Appendices

ACUCOBOL-GT®

Version 8.1.3

Micro Focus
9920 Pacific Heights Blvd
San Diego, CA 92121
858.790.1900

E-01-AP-100501-Appendix-8.1.3

Contents

Appendix A: Specifications

Appendix B: ACUCOBOL-GT Reserved Words

Appendix C: Changes Affecting Previous Versions

Appendix D: Compiler Error Messages

Appendix E: File Status Codes

Appendix F: Intrinsic Functions

Appendix I: Library Routines

Index

A Specifications

Key Topics

A.1 COBOL Modules

ACUCOBOL-GT is an ANSI-85 COBOL compiler and runtime system (ANSI X3.23-1985 and the ANSI X3.23-1989 supplement). ANSI COBOL is divided into a series of required and optional modules, each of which has various levels of implementation. ACUCOBOL-GT conforms to the following levels for each of the required modules (range of levels in parentheses):

Nucleus (1-2)	Level 2
Sequential I-O (1-2)	Level 2
Relative I-O (0-2)	Level 2
Indexed I-O (0-2)	Level 2
Inter-Program Communication (1-2)	Level 2
Sort-Merge (0-1)	Level 1
Source Text Manipulation (0-2)	Level 2
Segmentation (0-2)	Level 1

ACUCOBOL-GT does not support the optional modules: Report Writer, Communication, or Debug.

The following sections summarize various extensions and limitations ACUCOBOL-GT has with respect to the standard.

A.2 Limits and Ranges

ACUCOBOL-GT has the following limits:

Maximum Program Size: (compilation unit)	16 MB code, 2 GB data
Maximum Program Size: (run unit)	Limited only by machine memory
Maximum Record Size:	64 MB (67,108,864)
Number of Indexed Keys:	Primary + 119 alternates

Number of Segments per Key:	16
Maximum Indexed Key Size:	250 bytes
Maximum Sort Key Size:	32767 bytes
Maximum Number Sort Keys:	255
Maximum duplicate keys:	No limit (Vision)
Maximum File Size:	Host system dependent Logical limit: 128 terabytes, if Vision Version 5 or 4 is used; for all other Vision versions, the logical limit is 2048 MB
Maximum Data Item Size	
- Alphanumeric:	2 GB
- Numeric:	31 digits (default is 18, but can be set to 31 by using the "-Dd31" compiler option. See **Section 2.2.10** of the *ACUCOBOL-GT User's Guide* for details).
- Edited	255 bytes
Maximum Table Indexes:	15
Maximum Open Files/Process:	32767
Maximum Literal Size:	32767 characters
Maximum Paragraph Size:	32767 bytes
Maximum Picture String:	100 Characters
SPECIAL-NAMES Switches:	26
Maximum number of OCCURS:	2147483647
Maximum recursive CALL depth:	32767
Maximum number of parameters in a CHAIN statement	50
Maximum *number* in a "PERFORM *number* TIMES" statement	2,147,483,647

Maximum number of Linkage Section level-01 data items per program	255
Maximum number of ENTRY points per program	65536
Maximum number of characters in an alphanumeric data item used in a DISPLAY statement	2048

A.3 Extensions

ACUCOBOL-GT contains many extensions to the ANSI standard. These are summarized below:

- Terminal-oriented source format
- Compile-time modification of source by Identification Area flags
- The Identification Division is optional
- IS RESIDENT PROGRAM clause
- An index item may subscript a table other than the one it is associated with. Index data items may be used any place a numeric data item is allowed
- Apostrophes may be used to delimit nonnumeric literals. Hexadecimal literals are allowed
- A procedure name may be the same as a data item name
- Initial paragraph name not required
- Paragraph names allowed in Area B
- Multiple-word SOURCE-COMPUTER and OBJECT-COMPUTER names
- Data Division FILE SECTION header is optional

- The word ALPHABET is optional when you are declaring an alphabet-name in the SPECIAL-NAMES paragraph
- The ASSIGN TO clause may have a data item specified for the external file name. Also, the external file name is optional in the clause
- An optional device type may be specified in an ASSIGN clause
- WITH COMPRESSION, WITH ENCRYPTION added to ASSIGN clause
- LINE and BINARY options in ORGANIZATION clause
- COLLATING SEQUENCE clause
- COMPRESSION CONTROL clause
- LOCK MODE clause
- LENGTH OF clause for data literals
- RECORD-POSITION clause for data items
- RESERVE clause with the NO or ALTERNATE options
- Split key specification for indexed files
- FILE STATUS clause for sort files
- REDEFINES can reference an item that is itself a redefinition of an area
- Additional SPECIAL-NAMES clauses: CONSOLE IS CRT, CRT STATUS, CURSOR IS, EVENT STATUS, and NUMERIC SIGN SEPARATE
- SEGMENT-LIMIT clause (level 2 segmentation feature)
- VALUE OF FILE-ID clause
- USAGE COMP-1, COMP-2, COMP-3, COMP-4, COMP-5, COMP-6, COMP-N, COMP-X, FLOAT, DOUBLE, and HANDLE

- USAGE types:

SIGNED-SHORT	UNSIGNED-SHORT
SIGNED-INT	UNSIGNED-INT
SIGNED-LONG	UNSIGNED-LONG

- ADDRESS OF phrase in arithmetic expressions
- Tables may contain up to 15 dimensions
- A PICTURE string may contain up to 100 characters
- Level 78 constant names
- WHEN SET TO FALSE phrase for level 88 condition-names. A FALSE phrase added to the SET statement
- SCREEN SECTION
- SCREEN SECTION BEFORE, AFTER, and EXCEPTION embedded procedures
- SCREEN SECTION EVENT procedures
- IS SPECIAL-NAMES phrase in record description entry
- CHAINING phrase added to Procedure Division header
- Non-display data items may be specified in a NUMERIC class condition
- USE statements may reference sort files
- RETURN-CODE special register
- ACCEPT with screen control
- ACCEPT FROM SYSTEM-INFO, TERMINAL-INFO, INPUT STATUS, LINE NUMBER, COMMAND-LINE, ESCAPE KEY, CENTURY-DATE, CENTURY-DAY, STANDARD OBJECT, and WINDOW HANDLE
- ACCEPT FROM SCREEN

- ACCEPT CONTROL statement
- ACCEPT ALLOWING messages phrase
- ACCEPT *external-form-item* statement
- ADD TABLE statement
- CALL RUN statement
- CALL PROGRAM statement
- CALL THREAD statement
- Literals allowed in the USING portion of a CALL statement. Also, non-level 01 group items may be listed in the USING phrase
- BULK-ADDITION phrase for OPEN statement
- BY VALUE phrase for CALL statement
- OMITTED/NULL phrase for CALL statement
- NOT ON OVERFLOW accepted for CALL statement
- ALL option for CANCEL statement
- CHART option for CANCEL statement
- CHAIN statement
- CLOSE WINDOW statement
- COMMIT statement
- COPY RESOURCE statement
- CREATE statement
- DELETE FILE statement
- DESTROY statement
- DISPLAY with screen control

- DISPLAY SUBWINDOW/WINDOW statement
- DISPLAY FLOATING WINDOW statement
- DISPLAY SCREEN statement
- DISPLAY LINE statement
- DISPLAY BOX statement
- DISPLAY UPON WINDOW TITLE statement
- DISPLAY UPON COMMAND-LINE statement
- DISPLAY INITIAL WINDOW statement
- DISPLAY INDEPENDENT WINDOW statement
- DISPLAY TOOL-BAR statement
- DISPLAY *control-type* statement
- DISPLAY MESSAGE BOX statement
- DISPLAY *external-form-item* statement
- DRAW CHART statement
- ENTER CHART DATA statement
- ENTRY statement
- GOBACK statement
- INQUIRE CONTROL statement
- INQUIRE WINDOW statement
- LOCK THREAD statement
- TRAILING option on INSPECT statement
- MODIFY statement
- NEXT SENTENCE statement

- WITH LOCK and ALLOWING phrases added to OPEN statement
- MASS-UPDATE option on OPEN statement
- WITH NO LOCK and ALLOWING phrases on READ statement
- PERFORM THREAD statement
- PREVIOUS option on READ statement
- Literal allowed in FROM phrase of REWRITE and WRITE statements
- SEND message statement
- RECEIVE message statement
- SET CHART ATTRIBUTE statement
- SET FILE-PREFIX statement
- SET ENVIRONMENT statement
- SET EXCEPTION statement
- SET TO ADDRESS OF statement
- SET TO SIZE OF statement
- SET HANDLE statement
- SET THREAD statement
- SET WINDOW statement
- STOP THREAD statement
- SUBTRACT TABLE statement
- LESS THAN and LESS THAN OR EQUAL options on START statement
- UNLOCK statement

- DECLARATIVE procedures may reference procedures outside of DECLARATIVES
- Recursive CALLs
- Dynamically determined SORT keys
- EXIT PERFORM, EXIT PARAGRAPH and EXIT SECTION
- ROLLBACK clause for LOCK MODE phrase, on SELECT statement in FILE-CONTROL paragraph
- COMMIT statement may indicate end of transaction and cause changes to be written to transaction log file
- ROLLBACK statement
- SET statement with ADDRESS OF clause sets address of linkage data item to specified value
- START TRANSACTION statement
- SUPPRESS clause for the COPY statement
- USE *active_x_control_item* and USE *ole_object_item* statements
- USE FOR REPORTING statement
- UNLOCK THREAD statement
- WAIT statement

A.4 Restrictions

The current version of ACUCOBOL-GT has the following restrictions with respect to the standard. Many of these will be lifted in future versions of ACUCOBOL-GT.

- The Procedure Division is required.
- The ENTER statement is unsupported (obsolete feature).

- The RERUN clause is unsupported (obsolete feature).
- The COMMON clause of the PROGRAM-ID is unsupported.
- Nested source programs are unsupported.

B ACUCOBOL-GT Reserved Words

Key Topics

B.1 Conventions

This appendix lists all the reserved words used by ACUCOBOL-GT. Words that are reserved by ACUCOBOL-GT but not by the 1985 standard are indicated as follows:

(a) Indicates that the word is reserved by a special feature of ACUCOBOL-GT

(b) Indicates that the word is reserved by IBM DOS/VS. They are treated as reserved words by ACUCOBOL-GT only if you compile with the "-Cv" compiler option.

(h) Indicates that the word is reserved by HP COBOL. They are treated as reserved words by ACUCOBOL-GT only if you compile with the "-Cp" compiler option.

(i) Indicates that the word is reserved by both ACUCOBOL-GT and Data General ICOBOL

(r) Indicates that the word is reserved by both ACUCOBOL-GT and RM/COBOL

(s) Indicates that the word is reserved by ACUCOBOL-GT for use in the Screen Section

(v) Indicates that the word is reserved by both ACUCOBOL-GT and VAX COBOL

(8) Indicates that the word is reserved by the 1985 standard, but not by the 1974 standard

(*) Indicates that the word is reserved by the 1985 standard but not used by ACUCOBOL-GT. These words are treated as user symbols by the compiler. They may become reserved in the future as more features of the 1985 standard are implemented, so their use is not advised.

B.2 Reserved Word List

This section lists each reserved word in alphabetical order.A

ACCEPT	ACCESS
ACTUAL(b,h)	ADD
ADDRESS(a)	ADVANCING
AFTER	ALL
ALLOWING	ALPHABET(8)
ALPHABETIC	ALPHABETIC-LOWER(8)
ALPHABETIC-UPPER(8)	ALPHANUMERIC(8)
ALPHANUMERIC-EDITED(8)	ALSO
ALTER	ALTERNATE
AND	ANY(8)
APPLY(v)	ARE
AREA	AREAS
ASCENDING	ASSEMBLY-NAME(a)
ASSIGN	AT
ATTRIBUTE(a)	AUTHOR
AUTO(i,s)	AUTO-MINIMIZE(a)
AUTO-RESIZE(a)	AUTO-SKIP(s)
AUTOMATIC(a)	AUTOTERMINATE(v)

B

BACKGROUND-COLOR(s)
BACKGROUND-COLOUR(s)
BACKGROUND-HIGH(a)
BACKGROUND-LOW(a)
BACKGROUND-STANDARD(a)
BACKWARD(i)

BEEP(r,s)
BEFORE
BELL(v,s)
BIND(a)
BINARY(8,r)
BLANK
BLINK(i,r,s)
BLINKING(v)
BLOCK
BOLD(v)
BOTTOM
BOX(a)
BOXED(a)
BULK-ADDITION(a)
BY

C

CALL
CANCEL
CCOL(a)
CD(*)
CELL(a)
CELLS(a)
CENTERED(a)
CENTURY-DATE(a)
CENTURY-DAY(a)
CF(*)
CH(*)
CHAIN(a)
CHAINING(a)
CHARACTER
CHARACTERS
CHART(a)
CLASS(8)
CLASS-NAME(a)
CLINE(a)
CLINES(a)
CLOCK-UNITS(*)
CLOSE
COBOL(*)
CODE(*)
CODE-SET
COL(i,s)
COLLATING
COLOR(a)

COLOUR(a)
COLUMN
COM-REG
COMMA
COMMAND-LINE(a)
COMMIT(a)
COMMUNICATION(*)
COMP
COMP-1(r,v)
COMP-2(a)
COMP-3(r,v)
COMP-4(r)
COMP-5(a)
COMP-6(r)
COMP-N(a)
COMP-X(a)
COMPRESSION(a)
COMPUTATIONAL
COMPUTATIONAL-1(r,v)
COMPUTATIONAL-2(a)
COMPUTATIONAL-3(r,v)
COMPUTATIONAL-4(r)
COMPUTATIONAL-5(a)
COMPUTATIONAL-6(r)
COMPUTATIONAL-N(a)
COMPUTATIONAL-X(a)
COMPUTE
CONFIGURATION
CONSOLE(s)
CONSTRUCTOR(a)
CONTAINS
CONTENT(8)
CONTINUE(8)
CONTROL
CONTROLS(a)
CONVERSION(v)
CONVERT(r)
CONVERTING(8)
COPY
CORE-INDEX(b)
CORR
CORRESPONDING
COUNT
CREATE(a)
CRT(s)
CSIZE(a)
CULTURE(a)
CURRENCY
CURRENT-DATE(b,h)
CURSOR(r)
CYCLE(a)
CYL-INDEX(b)
CYL-OVERFLOW(b)

D

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DATE YYYYMMDD(a)
DAY
DAY-OF-WEEK(8)
DAY YYYYDDD(a)
DE(*)
DEBUG-CONTENTS(*)
DEBUG-ITEM(*)
DEBUG-LINE(*)
DEBUG-NAME(*)
DEBUG-SUB-1(*)
DEBUG-SUB-2(*)
DEBUG-SUB-3(*)
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFAULT(v)
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESCRIPTOR(v)
DESTINATION(8)
DESTROY(a)
DETAIL(*)
DISABLE(*)
DISPLAY
DISPLAY-ST(b)
DIVIDE
DIVISION
DOUBLE(a)
DOWN(a)
DRAW(a)
DUPLICATES
DYNAMIC

E

ECHO(r,v)
EGI(*)
EJECT(b)
ELSE
EMI(*)
EMPTY-CHECK(s)
ENABLED
ENCRYPTION(a)
END
END-ACCEPT(a,r,v)

END-ADD(8)
END-CALL(8)
END-CHAIN(a)
END-COMPUTE(8)
END-DELETE(8)
END-DISPLAY(a)
END-DIVIDE(8)
END-EVALUATE(8)
END-IF(8)
END-MODIFY(a)
END-MOVE(a)
END-MULTIPLY(8)
END-OF-PAGE
END-PERFORM(8)
END-READ(8)
END-RECEIVE(8)
END-RETURN(8)
END-REWRITE(8)
END-SEARCH(8)
END-START(8)
END-STRING(8)
END-SUBTRACT(8)
END-UNSTRING(8)
END-USE(a)
END-WAIT(a)
END-WRITE(8)
ENDING(a,b)
ENTER
ENTRY(a, b)
ENVIRONMENT
EOL(r,s)
EOP
EOS(r,s)
EQUAL
ERASE(r,v)
ERROR
ESCAPE(i,s)
ESI(*)
EVALUATE(8)
EVENT(a)
EVERY(*)
EXAMINE(b,h)
EXCEPTION
EXCLUSIVE(h,i)
EXIT
EXTEND
EXTENDED-SEARCH(r)
EXTERNAL(8)
EXTERNAL-FORM(a)

F

FALSE(8)
FD
FILE
FILE-CONTROL
FILE-ID(v)
FILE-LIMIT(r)
FILE-LIMITS(r)
FILE-PATH(a)
FILE-PREFIX(a)
FILLER
FINAL(*)
FIRST
FLOAT(a)
FLOATING(a)
FONT(a)
FOOTING
FOR
FOREGROUND-COLOR(s)
FOREGROUND-COLOUR(s)
FREE(h)
FROM
FULL(i,s)
FUNCTION

G

GENERATE(*)
GIVING
GLOBAL(8)
GO
GOBACK(r)
GRAPHICAL(a)
GREATER
GRID(s)
GROUP(*)

H

HANDLE(a)
HEADING(*)
HEIGHT(a)
HELP-ID(a)
HIGH
HIGH-VALUE
HIGH-VALUES
HIGHLIGHT(s)

I

I-O
I-O-CONTROL
ICON(a)
ID
IDENTIFICATION
IDENTIFIED(a)
IF
IN
INDEPENDENT(a)
INDEX
INDEXED
INDICATE(*)
INITIAL
INITIALIZE(8)
INITIATE
INPUT
INPUT-OUTPUT
INQUIRE(a)
INSPECT
INSTALLATION
INTO
INVALID
IS

J

JUST
JUSTIFIED

K

KEPT(a)
KEY

L

LABEL
LAST
LAYOUT-DATA(a)
LAYOUT-MANAGER(a)
LEADING
LEFT
LEFTLINE(s)
LENGTH
LENGTH-CHECK(s)
LESS
LIMIT(*)
LIMITS(*)

LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER(*)
LINES
LINK(a)
LINKAGE
LOCK
LOCK-HOLDING(v)
LOW(r)
LOW-VALUE
LOW-VALUES
LOWER(a)
LOWLIGHT(s)

M

MANUAL(a)
MASS-UPDATE(a)
MASTER-INDEX(b)
MEMORY
MENU(a)
MERGE
MESSAGE(a)
MESSAGES(a)
MODAL(a)
MODE
MODELESS(a)
MODIFY(a)
MODULE(a)
MODULES
MOVE
MULTIPLE
MULTIPLY

N

NAMESPACE(a)
NATIONAL
NATIONAL-EDITED
NATIVE
NEGATIVE
NEXT
NO
NO-ECHO(s)
NOLIST(h)
NOMINAL(b)
NOT
NOTE(b)
NULL(a)
NULLS(a)

NUMBER	NUMERIC
NUMERIC-EDITED(8)	NUMERIC-FILL(a)

○

OBJECT(a)	OBJECT-COMPUTER
OCCURS	OF
OFF	OMITTED
ON	ONLY(a)
OPEN	OPTIONAL(8)
OR	ORDER(8)
ORGANIZATION	OTHER(8)
OTHERS(v)	OTHERWISE(b)
OUTPUT	OVERFLOW
OVERLAPPED(a)	OVERLINE(s)

P

PACKED-DECIMAL(8)
PADDING(8)

PAGE
PAGE-COUNTER(*)
PARAGRAPH(a)
PASSWORD(b)
PERFORM
PF(*)
PH(*)
PIC
PICTURE
PIXEL(a)
PIXELS(a)
PLUS
POINTER
POP-UP(a)
POS(s)
POSITION
POSITIONING(b)
POSITIVE
PREVIOUS(a,i)
PRINT-CONTROL(v)
PRINTING(*)
PRIORITY(a)
PROCEDURE
PROCEDURES(*)
PROCEED
PROCESSING(b)
PROGRAM
PROGRAM-ID
PROMPT(r,s)
PROPERTY(a)
PROTECTED(i,v)
PURGE(*)

Q

QUEUE(*)
QUOTE
QUOTES

R

RANDOM
RD(*)
READ
READERS(v)
RECEIVE
RECORD
RECORD-POSITION(a)
RECORDING(i)

RECORDS
REDEFINES
REEL
REFERENCE(8)
REFERENCES(*)
RELATIVE
RELEASE
REMAINDER
REMARKS(a)
REMOVAL
RENAMES
REPLACE(8)
REPLACING
REPORTING
REQUIRED(i,s)
REPORT(*)
REPORTING(*)
REPORTS(*)
RERUN(*)
RESERVE
RESET(*)
RESIDENT(a)
RESIZABLE(a)
RETURN
RETURNING
RETURN-CODE
RETURN-UNSIGNED
REVERSE(r)
REVERSE-VIDEO(s)
REVERSED
REWIND
REWRITE
RF(*)
RH(*)
RIGHT
ROLLBACK
ROUNDED
RUN

S

SAME
SCREEN(i,s,v)
SCROLL(a)
SD
SEARCH
SECTION
SECURE(i,s)
SECURITY
SEEK(r,h)
SEGMENT(*)
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE

SEQUENTIAL
SET
SHADOW(a)
SIGN
SIGNED-INT(a)
SIGNED-LONG(a)
SIGNED-SHORT(a)
SIZE
SKIP1(b)
SKIP2(b)
SKIP3(b)
SORT
SORT-CORE-SIZE(b)
SORT-FILE-SIZE(b)
SORT-MERGE
SORT-MODE-SIZE(b)
SORT-RETURN(b)
SOURCE(*)
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2(8)
START
STATUS
STOP
STRING
STRONG-NAME(a)
STYLE(a)
SUB-QUEUE-1(*)
SUB-QUEUE-2(*)
SUB-QUEUE-3(*)
SUBTRACT
SUBWINDOW(a)
SUM(*)
SUPPRESS(b)
SYMBOLIC(8)
SYNC
SYNCHRONIZED
SYSTEM(a)
SYSTEM-INFO(a)

T

TAB(r)
TABLE(a)
TALLY(a)
TALLYING
TAPE
TERMINAL(*)
TERMINAL-INFO(a)
TERMINATE(*)

TEST(8)
TEXT
THAN
THEN(8)
THREAD(a)
THREADS(a)
THROUGH
THRU
TIME
TIME-OF-DAY(b,h)
TIMES
TITLE(a)
TITLE-BAR(a)
TO
TOOL-BAR(a)
TOP
TRACK-AREA(b)
TRACKS(b)
TRAILING
TRANSACTION(a)
TRANSACTION-STATUS(a)
TRANSFORM(b)
TRUE(8)
TYPE(a,*)

U

UN-EXCLUSIVE(h)
UNDERLINE(s)
UNDERLINED(v)
UNIT
UNLOCK(i,r,v)
UNSIGNED-INT(a)
UNSIGNED-LONG(a)
UNSIGNED-SHORT(a)
UNSTRING
UNTIL
UP
UPDATE(r,v)
UPDATERS(v)
UPON
UPPER(a)
USAGE
USE
USING

V

VALUE
VALUES

VARYING
VERSION(a)
VISIBLE(a)

W

WAIT(a)
WHEN
WHEN-COMPILED(b)
WIDTH(a)
WIDE(a)
WITH
WINDOW(a)
WORKING-STORAGE
WORDS
WRITE
WRAP(a)
WRITE-VERIFY(b)
WRITE-ONLY(b)
WRITERS(v)

Y

YYYYDDD(8)
YYYYMMDD(8)

Z

ZERO
ZERO-FILL(s)
ZEROES
ZEROS

C Changes Affecting Previous Versions

Key Topics

ACUCOBOL-GT is generally backwards compatible with prior versions of ACUCOBOL-GT and ACUCOBOL-85. There are, however, some changes that can affect existing programs. These changes are detailed in this appendix.

C.1 Changes Affecting Version 8.1.2

Compiler - Decimal math now the default behavior

In version 7.2, a new math package called Binary Math (--bin) was added to the compiler and became the default behavior over the previous Decimal Math (--dec). See **section C.5.** for details.

Starting with Version 8.1.2, the compiler's default behavior is reverted back to decimal math.

C.2 Changes Affecting Version 8.1.1

XFD format and -Fe compiler option

By default the compiler will now generate XFDs in XML format, as opposed to standard flat text, which was the default prior to Version 8.1.1. This change also affects the -Fe compiler option. This option will now direct the compiler to create XFDs in flat text format, as opposed to XML format, which was the case in versions prior to 8.1.1.

Read-only entry fields

In Version 8.1, read-only entry fields were changed to conform to standard Windows behavior in that the background color is always gray (regardless of the COBOL program's Color setting). This is still the case in Version 8.1.1; however, this behavior is now configurable. If you need the ability to change the color of read-only entry fields, set the runtime configuration variable "ECN_3699" to "0".

C.3 Changes Affecting Version 8.1

$ Symbol in source code

In Version 8.1, COBOL source code can now contain lines with a dollar sign ($) in the indicator area, which may be used with the IF, ELSE, END, DISPLAY, and SET statement to support conditional compiling.

The $ symbol is also a valid comment character. If a program uses $ as a comment, and it is immediately followed by IF, ELSE, END, DISPLAY an error will most likely be generated.

Read-only text fields

As of Version 8.1, read-only entry fields were changed to conform to standard Windows behavior in that the background color is always gray (regardless of the COBOL program's Color setting). If you need the ability to change the color of read-only entry fields, set the runtime configuration variable "ECN_3699" to "0".

C.4 Changes Affecting Version 8.0

Compiler

In previous versions, an END-PERFORM was required when the PERFORM was nested in an EVALUATE statement. With Version 8.0, the compiler accepts the WHEN verb as an implied END-PERFORM.

Interoperability

Comments in C$XML no longer include the expression:

```
'.* - generated by ACUCOBOL-GT v.*\n*'
```

If you depend on having those comments, you will need to rework your application in some way.

For .NET, the type checking rules for using overloaded methods in a COBOL program are more stringent now. You must use COBOL types that match the NETDEFGEN COPY file method declaration.

```
SIGNED-INT - int32.
   UNSIGNED-INT - uint32.
   SIGNED-LONG  - long.
   SIGNED-SHORT - int16.
   UNSIGNED-SHORT  - uint16.
PIC X(nn)- BSTR
pic 9- BOOLEAN
PIC X- BYTE
```

C.5 Changes Affecting Version 7.2

Compiler changes

In previous versions, ACUCOBOL-GT has performed the majority of its arithmetic operations using a 40-digit decimal format (68 digits if using the "-Dd31" compiler option). Starting with Version 7.3, ACUCOBOL-GT uses a binary math package as its default. The decimal package remains in place to handle certain high-precision cases and to maintain compatibility with existing programs. Users of ACUCOBOL-GT can choose which package they use: the binary package for enhanced performance or the decimal one for compatibility with prior compilers. Use the "--decimalMath" (or "--dec") compiler option to use the decimal math format. Refer to section 2.1.13, "Miscellaneous Options," in *ACUCOBOL-GT User's Guide* for more information about these compiler options.

Prior to Version 7.3, **cblutil** produced instructions that ran under both POWER and PowerPC architectures when generating PowerPC native code. Starting with Version 7.3, this is no longer true. The reason is that the code generator started using multiply and divide instructions in some important cases, and some of these instructions changed. The existing "--ppc" compiler option now produces 32-bit PowerPC code that is also compatible with POWER3, POWER4, and POWER5 processors. This code does not run correctly on POWER- or POWER2-based machines. A new "--power" option produces code that is compatible with POWER and POWER2

processors, as well as PowerPC and later POWER series processors. This code can be significantly slower than code generated with "--ppc", but it does run on a wider range of machines. Please refer to section 2.1.2, "Native Code Options," in *ACUCOBOL-GT User's Guide* for more information about native code generation.

Two compilation switches provide compatibility with Version 7.2:

-C72	Causes the compiler to generate code according to the rules used by Version 7.2.
-Z72	Creates object code that can be run with a Version 7.2 runtime.

C.6 Changes Affecting Version 7.1

Compiler and runtime changes

Beginning with Version 7.2, the wheel mouse can be used for scrolling in a center- or right-aligned entry field. To preserve the pre-7.2 behavior and prevent scrolling in these situations, set the V71_ALIGNED_ENTRY_FIELD configuration variable to "1" (on, true, yes) or compile your code for compatibility with a version older than Version 7.2. The default value of this variable is "0" (off, false, no).

In Version 7.2, the runtime uses a different font measuring algorithm when it computes font widths in Windows. With this change, the runtime now validates the data returned by the Windows GetTextMetrics function and corrects it when it is too large. The V71_FONT_WIDTHS configuration variable setting allows you to use the pre-Version 7.2 rules. This variable can have one of the following values:

-1	(default) The change is enabled for programs using Version 7.2 or later semantics. In other words, the program has been compiled with Version 7.2 or later and the command line does not contain a compiler option for pre-7.2 semantics.
0	The change is enabled.
1	The change is disabled and the Version 7.1 and earlier font measuring code is used.

Two compilation switches provide compatibility with Version 7.1:

-C71	Causes the compiler to generate code according to the rules used by Version 7.1.
-Z71	Creates object code that can be run with a Version 7.1 runtime.

C.7 Changes Affecting Version 7.0

The following sections describe changes that can affect programs originally written with ACUCOBOL-GT Version 7.0.

Compiler and runtime changes

Two compilation switches provide compatibility with Version 7.0:

-C70	Causes the compiler to generate code according to the rules used by Version 7.0.
-Z70	Creates object code that can be run with a Version 7.0 runtime.

In Version 7.1, the total size of parameters passed BY CONTENT is increased to 2GB. For Version 7.0 and earlier, the total size limit is 64K. If you compile with "-Z70", your program has the 64K limit for parameters passed BY CONTENT.

The maximum number of REPLACING elements in an INSPECT statement is increased to 256. For Version 7.0 and earlier, the limit is 30.

C.8 Changes Affecting Version 6.2

The following sections describe changes that can affect programs originally written with ACUCOBOL-GT Version 6.2.

UNIX: New default installation directory

On UNIX systems, the ACUCOBOL-GT development system (compiler, runtime, utilities, etc.) has a new default directory location. The new location is: "/opt/acucorp/720". This change has been made to conform with the Version 2.3 Filesystem Hierarchy Standard (FHS). Installing into the FHS standard location provides consistency and improves system integration. For more information about the FHS standard, please visit "www.pathname.com/fhs/pub/fhs-2.3.html".

Initialization of external data items

Versions of ACUCOBOL-GT prior to Version 7.0 had the behavior of initializing external data items to LOW-VALUES, even when the rest of Working-Storage was initialized to spaces. Beginning with Version 7.0, all Working-Storage data items are initialized to spaces or the value specified with the "-Dv" compile option. This includes external data items.

To maintain compatibility with programs that rely on the old behavior, you can compile for semantic compatibility with Version 6.2 or earlier. Use the "-C##" compile option to do this (for example, "-C62" for Version 6.2 compatibility). When you compile for compatibility with Version 6.2 or earlier, external data items are initialized to null bytes, regardless of how the rest of Working-Storage is initialized.

C functions

Starting with Version 7.0, ACUCOBOL-GT has added significant new features to the C interface, allowing you greater flexibility for calling COBOL programs from C and C++.

For existing programs, this means:

- The cobol() and cobol_no_stop() functions are still supported but have been deprecated. The new function, acu_cobol(), extends the options available in the C interface.

- Information on the deprecated cobol() and cobol_no_stop() functions is not documented in Version 7.0. Refer to Appendix F in previous versions of the ACUCOBOL-GT documentation set for this information.

Detailed descriptions of the current C functions are available in Chapter 6 of *A Guide to Interoperating with ACUCOBOL-GT.*

Compiler changes

Two compilation switches provide compatibility with Version 6.2:

-C62 Causes the compiler to generate code according to the rules used by Version 6.2.

-Z62 Creates object code that can be run with a Version 6.2 runtime.

Runtime changes

- The runtime now automatically corrects most reference modification range errors. It applies the following rules:

  a. A start reference less than 1 is treated as 1. For example, var(0:3) is treated as var(1:3).

  b. A length reference less than 0 is treated as 0. Moving a zero-byte item is equivalent to moving spaces to the destination item. A zero-byte destination is not affected by the move. In a STRING statement, a length of zero for a string source is treated as 1, not 0.

  c. A start plus length reference that is past the end of the item is treated as meaning to the end of the item. For example, if the var is a PIC X(5) item, var(4:23) is treated as var(4:2).

  The **WARNINGS** runtime configuration variable provides some control over how reference modification range errors are handled. See its entry in Appendix H.

- The behavior of the character-based tree view control has changed. In previous versions, the MSG-TV-SELCHANGE message was not sent if the COBOL program deleted an item. Nor was it sent when the COBOL program first ACCEPTed a tree view control. Beginning with Version 7.0, the MSG-TV-SELCHANGE message is now sent in both cases.

- Beginning with Version 7.0, when the runtime reduces the size of a window to fit the screen, it includes any fractional lines and columns that fit, provided the COBOL program attempted to create a window with fractional lines and columns. For example, if the program creates a 70.0

line window, but only a 66.4 line window fits on the display, the runtime detects that no fractional lines were attempted, and truncates the number of lines to 66.0. However, if you attempt to create a 70.1 line window, the runtime recognizes the fractional measurement and displays a 66.4 line window. In prior versions, the runtime always reduced the size of the window to a whole number. To preserve the old behavior, set the configuration variable V62_MAX_WINDOW to "1" (on, true, yes).

- Beginning with Version 7.0, the Web runtime uses ANSI_FIXED_FONT as the standard font. Because some systems may depend on the old font, this change is configurable. To use the font standard from Versions 6.2 and earlier (SYSTEM_FIXED_FONT), adjust the setting of the configuration variable **STD_FIXED_FONT**, described in Appendix H.

C.9 Changes Affecting Version 6.1

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 6.1.

Compiler changes

Programs that will be deployed on 64-bit Windows systems and that have USAGE POINTER data items must be recompiled with Version 6.1.1 or later. This is because, beginning with Version 6.1.1 (the introduction of ACUCOBOL-GT for 64-bit Windows), the compiler and runtime differentiate between USAGE LONG and USAGE POINTER data items. This is necessary for 64-bit Windows.

Two compilation switches provide compatibility with Version 6.1:

-C61	Causes the compiler to generate code according to the rules used by Version 6.1.
-Z61	Creates object code that can be run with a Version 6.1 runtime.

Runtime changes

- Beginning with Version 6.2, on the HP e3000, if a program is compiled with the "-Cp" option, OPEN OUTPUT statements create temporary files. This is consistent with the behavior of native HP COBOL on the platform. Prior to Version 6.2, OPEN OUTPUT statements created permanent files.
- Beginning with Version 6.2, C$OPENSAVEBOX makes use of the OPNSAV-FLAGS field of OPENSAVE-DATA. Prior versions ignored the field.

C.10 Changes Affecting Version 6.0

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 6.0.

Compiler changes

Two compilation switches provide compatibility with Version 6.0:

-C60	Causes the compiler to generate code according to the rules used by Version 6.0.
-Z60	Creates object code that can be run with a Version 6.0 runtime.

Alignment of literals

The compiler uses a new algorithm for aligning literals in memory. The alignment is the *smaller* of the alignment specified by the "-Da" option (which has a default value of "4") or the largest power of 2 that is less than or equal to the literal's size. For example, a literal that requires 3 bytes of memory will have an alignment of 2. You can use the "--noAlignLit" option to turn off the new algorithm. See section 2.1.9, "Data Storage Options," in Book 1, *ACUCOBOL-GT User's Guide*, for additional information on "--noAlignLit."

Runtime changes

- In Version 6.0 and earlier, the WIN$PRINTER functions WINPRINT-PRINT-BITMAP, WINPRINT-SET-CURSOR, and WINPRINT-GRAPH-DRAW, ignored the form feed status of a pending print job, causing images or text to print on the wrong page. In Version 6.1 and later, calls to these functions automatically test for a pending form feed before printing.

- In Version 6.1, when a program is compiled with the "-Cp" switch and run on an HP e3000, all OPEN OUTPUT statements create MPE files. In prior versions, byte stream files were created.

- In Version 6.0 and earlier, if a program argument was preceded by a double dash (two dashes) it was effectively treated as if preceded by a single dash. For example, "runcbl --dle errfile iobench" was executed as if it were "runcbl -dle errfile iobench". Beginning with Version 6.1, an argument preceded by two dashes generates a runtime startup error, unless you specifically modify exam_args (in "sub.c") to ignore command-line errors.

C.11 Changes Affecting Version 5.2

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 5.2.

Vision Version 5

Version 6.0 introduces a new Vision file format: Vision Version 5. Vision Version 5 supports records up to 64 megabytes in size, block sizes up to 8192 bytes, very large pre-allocate and extension factors, and a virtually unrestricted number of records that allow duplicates. Version 5 files cannot be read by ACUCOBOL-GT Version 5.2.1 or earlier runtimes. For a complete description of Vision Version 5, see section 6.1.3, "Indexed Files - Vision" in Book 1, *ACUCOBOL-GT User's Guide*.

Windows console runtime

Version 6.0 introduces a new runtime for the Windows operating environment that may be used to run applications originally deployed in the Extended DOS environment, as well as other character-based applications. The new runtime is called the *console* runtime. The name of the executable is “crun32.exe”. The console runtime uses the Windows Console API and runs in a virtual DOS window. The console runtime replaces the Extended DOS runtime and is sold separately.

The console runtime can run ACUCOBOL-GT applications developed for the Extended DOS environment provided that some minor changes are made. For example, the console runtime supports printing capabilities based on the Windows model. Program code that relies on DOS printing functions must be modified.

The following runtime configuration variables are MS DOS-specific and are not supported in Version 6.0:

132_MODE
A_WAIT_FOR_LICENSE
AUTO_BUFFER
DOS_OUTPUT_METHOD
DOS_WATCOM_10
LOCKED_RECORD_DELAY
USE_MOUSE

Web Plug-in discontinued

The browser industry has shifted away from its support of Internet plug-ins in favor of ActiveX controls. For this reason, we developed and released an ActiveX-based Web Runtime in ACUCOBOL-GT Version 5.2.1. Due to lack of browser support, the ACUCOBOL-GT Web Plug-in is not offered or supported in Version 6.0. For information on migrating from the Web Plug-in to the Web Runtime, see section 5.11 of *A Programmer’s Guide to the Internet*.

List box and combo box handling of VALUE

In Version 5.2 and earlier, setting the VALUE of a combo box or list box caused the first item in the list that started with the value of VALUE to be selected, regardless of case. Beginning with Version 6.0, when a box's VALUE is set, the list is searched for an exact, case sensitive match with the specified value. If the value is found, it is selected. If an exact match is not found, the list is searched for an exact match regardless of case. If a match is still not found, the list is searched again, this time for the first string that contains the passed VALUE as a leading substring, regardless of case.

This change could affect the behavior of an existing application. The configuration variable V60_LIST_VALUE allows you to select which search algorithm, new or old, to use. See **V60_LIST_VALUE** in Appendix H.

Area A in RM COBOL compatibility mode

Starting with Version 6.0, when you compile for RM COBOL compatibility ("-Cr"), in the Identification Division, Area A can start in either column 8 or 9 (ANSI format) or column 1 or 2 (terminal format). In prior versions, Area A in the Identification Division started precisely in column 8 (ANSI format) or column 1 (terminal format).

This change may cause warnings in programs that previously compiled without warnings. To revert to the old rule, you can use the "--noRmMargin" compiler option.

Image rendering for BITMAP controls

The image processing code used by Version 6.0 (and later) for BITMAP controls is device-dependent. This may affect image rendering in some programs, written for Version 5.2 or earlier, which rely on device-independent bitmaps. If BITMAP controls are displaying incorrectly, adjust the setting of the configuration variable, **V52_BITMAPS**, described in Appendix H.

Bitmap push button behavior change

If some event in the system forces the focus away from a text-based push button after a click has been initiated but not finished, the click is voided. Starting with Version 6.0, bitmap push buttons void the click just like a text-based push button. This change applies only to programs compiled for 6.0 semantics or later.

Changes to data items used by C$REDIRECT

The definitions of the HANDLER-PRE-ALLOCATE-AMOUNT, HANDLER-EXTENSION-AMOUNT, HANDLER-MAX-LREC-SIZE, HANDLER-MIN-LREC-SIZE, and HANDLER-SEGMENT-OFFSETdata items in “sample/handler.cpy” have changed.

Changes to data items used by I$IO

The definitions of the PRE-ALLOCATION-AMOUNT, EXTENSION-AMOUNT, MAX-REC-SIZE, MIN-REC-SIZE, and KEY-OFFSET data items in “sample/def/filesys.def” have changed.

Compiler changes

- Two compilation switches provide compatibility with Version 5.2:

  -C52 Causes the compiler to generate code according to the rules used by Version 5.2.

  -Z52 Creates object code that can be run with a Version 5.2 runtime.

- When compiling for Version 6.0 or later format, table indexes and USAGE INDEX data items are treated as 32-bit signed native binary data items. In versions prior to 6.0, the default for indexes is to act as 16-bit unsigned portable binary data items. In rare cases, this change from 16-bit to 32-bit indexes may cause problems with an existing program. One case where this could be a problem is if you place USAGE INDEX items in a data file. Another case would be if you rely on undefined overflow behavior with arithmetic on 16-bit indexes.

If you need to preserve indexes as 16-bit items, you can either compile for an object format prior to Version 6.0, or you can compile using the "--nodata32bit" option. This option inhibits the new data addressing features of 6.0 and causes indexes to be kept as 16-bit data items.

C.12 Changes Affecting Version 5.1

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 5.1.

Licensing changes

The licensing mechanism changed with the release of Version 5.0. In Version 5.2, this mechanism has been simplified:

- Node IDs are no longer used.
- When the license is installed, the Windows version of the Activator creates a separate license file for each product, in the same manner as UNIX.
- The Activator utility is not backwards compatible. You must use the version of the Activator utility that corresponds to the version of the product you are installing in order to create a proper license file.

A complete description of the current licensing mechanism is available in the *Getting Started* book.

Compiler changes

- Two compilation switches provide compatibility with Version 5.1:

  -C51 Causes the compiler to generate code according to the rules used by Version 5.1.

  -Z51 Creates object code that can be run with a Version 5.1 runtime.

- Compiler switch "-Zt" is not supported in Version 5.2.

- In versions prior to 5.2, the grid would not pass a MSG-GOTO-CELL-MOUSE event to the program when the user clicked on the cell containing the grid cursor. This was done to prevent extraneous messages from being sent to the program. However, this message can be useful in some cases, for example, to allow a user to deselect something that is already selected. Therefore, in Version 5.2 and later, the runtime no longer filters out MSG-GOTO-CELL-MOUSE messages just because the destination cell is the same as the current cell.

  Note: This change is active only for programs compiled for Version 5.2 or later. This means that the Version 5.2 runtime will use the old behavior when executing programs compiled with versions prior to 5.2, or compiled with the "-C51" or the "-Z51" switch. You can disable the new behavior by setting the configuration variable "V52_GRID_GOTO" to "0".

- Version 5.2 introduces "ENTRY" as a new reserved word in ACUCOBOL-GT. A program that compiled with a previous version of the compiler will not compile with Version 5.2 if it uses "entry" in certain places. For example:

  If "entry" appears in a paragraph name in your program, the compiler returns the error:

  "**Identifier expected, ENTRY found**"

If "entry" appears in a variable in your program, the compiler returns the error:

"syntax error scanning ENTRY"

See Book 3, *ACUCOBOL-GT Reference Manual*, section 6.6, "Procedure Division Statements," ENTRY Statement, for usage syntax and rules.

Runtime changes

- The ACUCOBOL-GT Version 5.2 runtime on SCO UNIX systems runs in the ELF binary format. Prior to Version 5.2, the runtime ran in the COFF format, but COFF does not support calling shared libraries so it

was changed to ELF. If you have your own C routines that you used to link to the runtime, you will need to recompile those C routines to create ELF objects to link to the 5.2 runtime. For details on calling shared library routines in UNIX environments, see Chapter 6 of *A Guide to Interoperating with ACUCOBOL-GT.*

- In versions prior to 5.2, the runtime would eliminate requests to resize a screen control if the new size and position matched the control's current size and position on the screen. With the current version, the runtime optimizes the control resize request using the SIZE and LINES indicated (or implied) by your program instead of the current size and position.

  These two ways of optimizing control resize requests produce nearly identical results. However, there are a few cases where the results can differ. For example, if you change the size of the subwindow that contains the control in such a way that the control would crop differently, then a comparison of the "actual" size shows a difference, while a comparison of the "requested" size does not. In this case, earlier versions of the runtime would resize the control, while the current version will not.

  This change was made to provide more predictable behavior and to improve efficiency when the display service is on a remote machine.

  If necessary, you can disable this behavior by setting OPTIMIZE_CONTROL_RESIZE to "0" (off, false, no). This prevents any optimization of control resizing operations. Note that this can result in additional screen painting (in which controls may appear to flicker) and should be used only as a short-term fix while any required coding changes are made.

- The way the runtime handles mouse click events in COBOL programs that contain both bitmap push buttons and multiple windows under the control of a single thread has changed.

  In Version 5.1 and earlier, in some cases, simply clicking down on the mouse button when a bitmap push button was selected, generated a CMD-CLICK event. This was not consistent with the way Microsoft Windows handles these events.

In Version 5.2 and later, clicking down on a bitmap pushbutton on a non-active window running in the same thread will cause the current ACCEPT to terminate with CMD-ACTIVATE event. The pushbutton is not considered clicked until the COBOL program performs some action that allows it to activate, such as ACCEPTing some control in the newly activated window. For self-activating pushbuttons, this allows the pushbutton to self activate. For non-self-activating pushbuttons, the new ACCEPT will terminate with a CMD-GOTO so that the COBOL program can ACCEPT the correct control.

This change is only available in COBOL objects compiled for Version 5.2 or later and run with a Version 5.2 or later runtime. COBOL objects compiled with Versions 5.1 or earlier will still exhibit the old behavior, even if they are run with a Version 5.2 or later runtime.

- The resolution of the ACCEPT BEFORE TIME timer has been substantially increased in Version 5.2. In rare cases, this could affect existing programs. To forestall any such problems, the runtime automatically uses the pre-5.2 resolution when running pre-5.2 objects and objects compiled for pre-5.2 compatibility (e.g. “-C51”).

C.13 Changes Affecting Version 5.0

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 5.0.

Compiler changes

Two compilation switches provide compatibility with Version 5.0:

-C50 Causes the compiler to generate code according to the rules used by Version 5.0.

-Z50 Creates object code that can be run with a Version 5.0 runtime.

Runtime changes

- In versions prior to 5.1, a CMD-ACTIVATE event would be generated only if there was an active ACCEPT statement running to receive it. Under some (unusual) circumstances, this could cause the runtime to enter a state where it believed the wrong window was active.

  In Version 5.1, this rule is modified so that CMD-ACTIVATE events *are* generated *unless* the window generating the event is in the process of being built. It no longer matters whether or not an ACCEPT statement is running. The new rule is needed to prevent the first ACCEPT in each window from immediately terminating due to a queued CMD-ACTIVATE event (generated by the window's own creation).

  Note: This is a change in the rules for when CMD-ACTIVATE is generated. As a result, it is possible for CMD-ACTIVATE events to occur in cases where they did not previously. In order to prevent this change from adversely affecting a working program, the new rule is used only for programs compiled for 5.1 semantics. This means that the 5.1 runtime will not behave any differently in this regard when executing programs compiled with 5.0 or earlier (or compiled with the "-C50" switch). You can explicitly enable this rule by setting the configuration variable "ECN-1660" to "1" or disable it by setting it to "0". When the variable is set to "-1" (the default), the program semantics apply as described above.

- In Version 5.1, you can assign pop-up menus to labels. This change has the side-effect that labels are now aware of mouse-clicks where previously they were not. This matters only if you happen to have a label and another control (like a push button) that overlap. Previously, the push button would always get all the mouse events. In Version 5.1, the label could start getting them. This can prevent the push button from working (because it is not "seeing" the mouse clicks). Normally, you would not overlap controls, but it can happen unintentionally if the label contains only spaces.

To correct this situation, make the controls not overlap or make label invisible instead of setting it to spaces if you want to hide it. You can inhibit this change by compiling for 5.0 or earlier semantics (this also means that you must compile for 5.1 or later semantics if you want to attach a pop-up menu to a label).

- The configuration variable V42_TRANSPARENT is now obsolete. Transparent labels always appear transparently. If this variable is set in your environment or in the runtime configuration file, it is simply ignored.

C.14 Changes Affecting Version 4.3

Version 5.0 of ACUCOBOL-GT contains significant changes in the internal workings of both the compiler and runtime. The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 4.3.

Licensing changes

With the Version 5.0 release, the licensing procedure for ***extend*** products has been changed. All products still require a license file, but you no longer need a separate license disk to install your products. Instead, you will receive a pair of alphanumeric strings (keys) that must be entered to activate your software. See your *Getting Started* book for details.

Compiler changes

Two compilation switches provide compatibility with Version 4.3:

-C43 Causes the compiler to generate code according to the rules used by Version 4.3.

-Z43 Creates object code that can be run with a Version 4.3 runtime.

- Prior to Version 5.0, the width of a printer cell was based on the average width of the selected printer font. Now, the width of a printer cell is computed in the same way that cells are computed for the screen, based on the width of the “0” (zero) character. Note that proportional fonts

may contain wider characters. This may affect the horizontal placement of a bitmap on the page, the width of bitmaps and margins if they are specified in cells, and the number of columns reported by the WINPRINT-GET-PAGE-LAYOUT call. See the **WIN$PRINTER** in Appendix I, "Library Routines", for details.

- The Arial font shipped with Windows 98 Version 2 is different from the Arial font shipped with earlier versions of Windows and Windows NT. The new font has a character width of 35 pixels, instead of the previous 23 pixel width. This can cause field overlap or screen distortions in programs that rely on the size of the Arial font. If you do not want to adjust your applications to accommodate the new wider version of the Arial font, a new configuration variable, **OLD_ARIAL_DIMENSIONS** will force the runtime to use the 23 pixel measurement. See **Appendix H, "Appendix H: Configuration Variables,"** for details.

- In previous versions, the command-line option for the **logutil** utility date filter "-d" had problems comparing dates when the specified 2-digit year was "00" or greater. Now, **logutil** requires that years be specified in a 4-digit format. If you enter a year less than 1900, **logutil** will report "logutil: use 4 digit year specification."

- Starting with ACUCOBOL-GT 5.0 release, the compiler no longer automatically assigns the "MULTILINE" style to an entry field with LINES value of "2" or greater. Although Version 5.0 correctly handles cases compiled with 4.x versions, in order for that to happen you need to specify an appropriate source-compatibility flag (such as "-Z43"). Note that the flag is not required if you had explicitly set the "MULTILINE" style in your 4.*x*-version program.

Runtime changes

- In previous versions, the UNIX runtime would use the name of the user that started a runtime process to identify the user to **acushare** and count the number of processes a user was executing simultaneously. In Version 5.0, the user is defined as a unique terminal name. Each terminal is counted as a unique user and requires one user license. Background processes adopt the name of the terminal which started them. There are 1024 processes per user allowed for each terminal name.

acushare 5.0 can be used with older runtimes and will report the maximum processes and processes per user settings for those runtimes as long as they have different serial numbers from the 5.0 runtime installed on the machine. If a Version 4.3 and a Version 5.0 runtimes on the same machine have the same serial number, **acushare** 5.0 supports both but does not report maximum processes or processes per user.

- Microsoft has changed standard input stream handling in Internet Information Server 4.0. When you are running with the "-f" option or when the A_CGI environment variable is set, the runtime reads only the number of bytes set in the CONTENT_LENGTH environment variable by the web server. The runtime no longer waits for an end of file condition.

- The runtime no longer differentiates between "UNIX-4" and "UNIX-V" in the OPERATING-SYSTEM field of the SYSTEM-INFORMATION structure. Instead, it reports "UNIX" for all UNIX operating systems.

C.15 Changes Affecting Version 4.2

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 4.2.

Compiler changes

Two compilation switches provide compatibility with Version 4.2:

-C42 Causes the compiler to generate code according to the rules used by Version 4.2.

-Z42 Creates object code that can be run with a Version 4.2 runtime.

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 4.2.

- The default ACCEPT size for a numeric-edited field now includes a space for the sign only if the field is signed. You can set the program to include a space for an implied sign by compiling for semantic compatibility with an earlier version of ACUCOBOL-GT using a compilation switch in the command line.

- One of the general rules for screen control entry has been modified: Prior to Version 4.3, an ACCEPT statement used to set ACCEPT-CONTROL to "1" if the event was a "message" ("MSG-…") event. Starting with version 4.3, if the reason for entry is a "notify" ("NTF-…") event, ACCEPT-CONTROL is set to "1"; otherwise it defaults to "0").

- The "-Fo" option has replaced the "-Zo" option. Both compiler options produce the same results, but the "-Zo" option should be considered obsolete.

- The compiler option "- Rw" has been expanded to allow, in addition to reserved words, the suppression of some non-reserved words, such as control names (e.g., "entry-field", "label") or property names (e.g., "max-text", "bitmap-number"). If you tell the compiler to suppress a non-reserved word, however, it will do so with the following warning: "Unknown reserved word: *non-reserved word*."

Runtime changes

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 4.2.

- If a program is in the event procedure for an active control, and the control activates and subsequently destroys another control, the control whose event procedure is executing is reactivated. In previous versions, the "active" control was left in an undefined state.

- When using the library routine WIN$PRINTER with the 32-bit runtime, the newer Windows PageSetup dialog box will appear by default. If you wish to use the old 16-bit PrintSetup dialog box, you must use the operation code WINPRINT-SETUP-OLD.

- In all releases up to and including the ACUCOBOL-GT 4.2 release, anytime you created an Entry-Field control with a LINES value of "2" or greater, it was treated as a multiline entry field. In version 4.3, this rule is modified so that a LINES value of "2" or greater implies MULTILINE only if the "CELLS" phrase is not also used or implied.

Calling COBOL from other languages

- Windows 95/98 and NT sites that are calling COBOL routines from C with the “cobol” routine need to be aware of a change to the calling convention. The calling convention has changed from “__cdecl” (the C calling convention) to “__stdcall” (the Pascal calling convention, used by Windows API routines). This was done to make integration with Visual Basic and Delphi more straightforward.

  Programs that call the “cobol” routine must be sure to include “sub.h” (included with ACUCOBOL-GT in the “lib” directory). This includes a declaration of the “cobol” routine for all platforms. This ensures that you use the correct calling convention when calling the “cobol” routine. If you have established routines that call “cobol”, these must be recompiled in order to use the new calling convention.

C.16 Changes Affecting Version 4.1

Two compilation switches provide compatibility with Version 4.1:

-C41 Causes the compiler to generate code according to the rules used by Version 4.1.

-Z41 Creates object code that can be run with a Version 4.1 runtime.

C.17 Changes Affecting Version 4.0

Two compilation switches provide compatibility with Version 4.0:

-C40 Causes the compiler to generate code according to the rules used by Version 4.0.

-Z40 Creates object code that can be run with a Version 4.0 runtime.

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 4.0.

- The compiler option "-Zx" (while still supported) has been replaced with "-Fx". Both options cause the compiler to generate XFD files in a new format (XFD Version 4). Version 4 XFD files include a list of all of the fields contained in a file's record description, including group items and REDEFINES items. Items that are excluded from use by the rules of XFD generation are marked with a condition number of 999. Versions of Acu4GL and alfred prior to Version 4.1 require the old format of XFD files. If you want to generate XFDs in the old format, use the new compiler option "-Fx3" or use "-Z40".

- The compiler option "-Za" now causes the compiler to test table indexes against the upper bound of the *current table size* when you compile for Version 2.4 or later semantics. Thus, if you have an OCCURS DEPENDING ON table that could hold 20 elements physically, but whose *current size* is 10 elements, the runtime will produce an error if you access elements 11--20 when compiling with "-Za".

  Compiling for Version 2.3 or earlier ("-C23") causes the compiler to test against *the physical upper bound*.

C.18 Changes Affecting Version 3.2

Two compilation switches provide compatibility with Version 3.2:

-C32 Causes the compiler to generate code according to the rules used by Version 3.2.

-Z32 Creates object code that can be run with a Version 3.2 runtime.

The following paragraphs describe changes that can affect programs originally written with ACUCOBOL-GT Version 3.2.

- Under 32-bit Windows, the user name returned in ACCEPT FROM SYSTEM-INFO is now retrieved from the system instead of from the USER environment setting. The name retrieved is the user's login name. This change could affect installed programs in that a user's name may appear differently than it did with earlier runtimes. The runtime does not provide a way to override this change, because doing so would present a security hole under Windows NT.

- The default compile output file has been changed from “cbl.out” to “*{source-name}*.acu”. This could affect the behavior of scripts used to compile programs. Also, if no CODE-SUFFIX is specified, the runtime tries a suffix of “.acu” before trying a blank suffix. This could affect programs if you happen to use a blank suffix for objects and have files named with the “.acu” extension in the same directory as your objects. To work around this, simply set CODE-SUFFIX explicitly in your configuration file. To specify a blank extension, simply add “CODE-SUFFIX” with no value.

- AUTO termination on a graphical screen now acts as if the “Tab” key had been pressed.

  Suppose a user is interacting with a screen that has an entry field followed by a radio button group. Normally, when the user tabs to the radio button group, control passes to the “group leader” (that is, the button that is selected, or the first button in the group, if none is selected). Prior to Version 4.0, if the entry field were defined with the AUTO style, then when the field was full, control passed to the very next item in the Screen Section. This might be a radio button that was not the “group leader.”

  The Version 4.0 runtime has been enhanced to treat this AUTO termination case as if the “Tab” key had been pressed, so that control passes to the “group leader” when the entry field becomes full.

  If the program is compiled with the “-C##” option, where “##” is a number less than 40 (such as “-C32” or “-C31”), this enhancement is disabled, and the behavior reverts to that of earlier versions.

- Starting with Version 4.0, the compiler uses a new rule when moving LOW-VALUES or HIGH-VALUES to a numeric item.

  Under standard COBOL, a MOVE of LOW-VALUES or HIGH-VALUES to a numeric item has undefined effects. Prior to Version 3.0, ACUCOBOL would treat these items as if they had legal numeric values, convert them accordingly, and move the result. This often results in a meaningless value, but can be useful for some numeric data items.

USAGE DISPLAY types, for example, would end up with LOW-VALUES in their storage. Non-DISPLAY types ended up with odd values. Some other COBOL systems would produce a value of zero in binary numeric items when LOW-VALUES were moved to them.

In order to improve compatibility with these systems, ACUCOBOL-GT was changed in Version 3.0 so that a MOVE of LOW-VALUES to a numeric item moved ZERO to that item. There were two concerns with this: (a) the compiler did not do this in every case, and (b) this changed the behavior of some programs that were functioning under prior versions of the runtime.

Starting with Version 4.0, the compiler uses the following rule when moving LOW-VALUES or HIGH-VALUES to a numeric item:

When the constant LOW-VALUES or HIGH-VALUES is the source of a MOVE statement whose destination is numeric, the move is treated as if the destination were defined as class alphanumeric. This results in the memory occupied by the numeric item being filled with LOW/ HIGH-VALUES.

This rule tends to produce the best results of both the pre-3.0 and post-3.0 behavior--the useful cases work out the same. Also, this rule expresses what most programmers believe should happen.

This new rule is used only for programs compiled for 4.0 semantics (this is the default). If you use the "-C##" or "-Z##" option to compile for earlier semantics, the compiler does not use this rule, and the runtime adjusts to use the semantics that were in place for version "##". For example, if you compile with "-C24", then the runtime will use the pre-3.0 semantics for the meaning of MOVE LOW-VALUES to a numeric item.

C.19 Changes Affecting Version 3.1

Two compilation switches provide compatibility with Version 3.1:

-C31 Causes the compiler to generate code according to the rules used by Version 3.1.

-Z31 Creates object code that can be run with a Version 3.1 runtime.

The following section describes changes that can affect programs originally written with ACUCOBOL-GT Version 3.1.

- The Vision Version 4 indexed file system uses a dual file format. Version 4 files cannot be read by ACUCOBOL-GT Version 3.1 or earlier runtimes. For a complete description of Vision Version 4, see section 6.1.3, “Indexed Files - Vision” in Book 1, *ACUCOBOL-GT User’s Guide*. Note that runtimes beginning with Version 3.2 are able to read any version of Vision file. To continue to use Vision Version 3 indexed files, see the entry for the **V_VERSION** configuration variable in Appendix H.

- Recursive PERFORMs are automatically enabled when you compile your programs with Version 3.2 or later. Recursive PERFORMs are *required* for the use of EVENT PROCEDURES. In very rare cases, this can affect the flow of control in a program. A program would be affected, for example, if it performs paragraph “A”, which performs paragraph “B” and then returns from “A” before returning from “B”. If you want, you can disable recursive PERFORMs with either the “-C31” (or earlier) flag or the “-Zr0” flag.

- Beginning with Version 3.2, data in a list box column can no longer overflow into the adjacent column (causing all columns to shift to the right). Instead, the data is truncated if it doesn’t fit in the allotted space for that column. There is no way to prevent this change.

- Beginning with Version 3.2, list box columns have a small buffer between them, so that the columns do not merge together when they are full. This can cause partial loss of the last character in a column if your columns are very close together. To correct this, set the configuration variable COLUMN-SEPARATION to zero.

- Beginning with Version 3.2, in environments that use system messages, such as Microsoft Windows, message processing during file I/O operations is no longer performed by default. This is due to problems that can occur in programs that use multithreading, modeless windows, or event procedures. To restore the old behavior, use the FILE-IO-PROCESSES-MESSAGES configuration variable. Enabling message processing should only be done under certain conditions. For a complete description, see the entry for **FILE_IO_PROCESSES_MESSAGES** in Appendix H.

- The IS NUMERIC test for COMP-3 fields is more rigorous beginning with Version 3.2. In prior versions, any bit pattern was allowed in the sign field. The runtime treated any bit pattern, other than 0x0D, as indicating a positive value. Starting with Version 3.2, only signs of 0x0C, 0x0D and 0x0F are treated as legal values in the IS NUMERIC test. These values are the normal values for signs (there are two positive values to match various other COBOLs). You can suppress this change by compiling for compatibility with Version 3.1 (i.e. "-C31").

- Beginning with Version 3.2, the DESTROY *handle-1* statement now sets the value of *handle-1* to NULL if the statement succeeds. In prior versions the value of *handle-1* was not changed. You can prevent the setting of *handle-1* to NULL by compiling for compatibility with Version 3.1 (i.e. "-C31").

C.20 Changes Affecting Version 2.4

The following section details changes that can affect programs originally written with ACUCOBOL-85 Version 2.4.

- Support for 16-bit MS-DOS compilers and runtimes has been eliminated. Support remains for 32-bit (*Extended*) DOS systems and for 32-bit and 16-bit Windows systems. This change does not affect the formal capabilities of ACUCOBOL-GT, but it does have some practical consequences.

  Primary among these is that ACUCOBOL-GT no longer supports the dynamic loading and linking of assembly language routines. Version 2.4 runtimes (and earlier) for 16-bit MS-DOS provided support for calling assembly language routines directly with the CALL verb. In

ACUCOBOL-GT Version 3.1 and later, if you want to use an assembly language routine you must link it directly into the ACUCOBOL-GT runtime in the same way that C routines are included.

Users who require 16-bit DOS support should use Version 2.4.

C.21 Changes Affecting Version 2.3

The following section details changes that can affect programs originally written for the Version 2.3 ACUCOBOL-85 compiler.

Compiler changes

New directory structure

Beginning with Version 3.0, a new directory structure is created when you load your media. See the "READ_ME" file for the location of all ***extend*** files. Note that the new directory structure may not be compatible with scripts you have in use at your site.

Runtime changes

Relinking the runtime

If you relink the runtime, be aware that the Makefile in the "lib" subdirectory leaves the rebuilt runtime in the "lib" subdirectory. This change allows you to rebuild the runtime system and test it without overwriting the original runtime, and without renaming it. However, be sure to move the rebuilt runtime to the "bin" subdirectory, or move it to a directory in your path.

Alternate file systems

Check the "RELEASE" notes to verify the compatibility of older versions of ***extend*** interfaces to alternate file systems such as Btrieve and INFORMIX.

C.22 Changes Affecting Version 2.1

The following section details changes that can affect programs originally written for the Version 2.1 ACUCOBOL-85 compiler.

Compiler changes

MS-DOS requirements

For machines using MS-DOS, ACUCOBOL-GT Version 3.0 requires MS-DOS version 3.0 or later. ACUCOBOL-85 Version 2.1 required only MS-DOS version 2.0.

Support for 64-bit architectures

Beginning with Version 3.0, ACUCOBOL-GT fully supports 64-bit machines without restriction. At the current time, the only machine that fits this classification is the DEC Alpha machine running OSF (Open/VMS also runs on the Alpha machine, but it runs in 32-bit mode). Version 2.1 of ACUCOBOL-85 also runs on 64-bit machines, but it contains some restrictions.

In Version 2.1, the following items are restricted:

- Since RETURN-CODE is only 32 bits in size, it cannot hold a “long” value properly. This makes it inappropriate for receiving “pointer” or “long” return values from a C subroutine.
- USAGE POINTER data items are also 32 bits, and so cannot actually hold a real machine address.
- The direct C interface cannot be used for “pointer” or “long” parameters since the 2.1 compiler does not allow you to pass a 64-bit item BY VALUE.

Beginning with Version 3.0, these restrictions do not apply. We made certain changes to the rules of ACUCOBOL-85, beginning with Version 2.3. These changes affect only a few existing COBOL programs, but they have the

potential of causing a working program to stop working. Because of this, there is a method available to inhibit these changes. See the "-Dw" option in section 2.1.9 of the *ACUCOBOL-GT User's Guide.*

The specific changes are:

- USAGE POINTER data items now occupy 8 bytes instead of 4 bytes. This allows a USAGE POINTER item to hold a full address on any machine architecture. On a machine that is smaller than 64-bits, only the first 32 bits of the POINTER item are used. The rest of the item is treated as FILLER.

  This is the change that is most likely to affect existing programs. You can be affected if you have POINTER data items as part of a group item, since the group item's size will change. If you have this case, then either allow the size of the group to change and adjust any external references or redefinitions of it, or use the option described below to keep POINTER items in 4 bytes.

- The special register RETURN-CODE was changed from PIC S9(9) COMP-5 to USAGE SIGNED-LONG. For a description of SIGNED-LONG, see the *ACUCOBOL-GT Reference Manual* section 5.7.1.8, "USAGE clause." This change allows RETURN-CODE to hold 64-bit values on 64-bit machines, and so it can be used to hold any return value from a called routine. This change should not affect any existing program.

- You may now pass 8-byte data items BY VALUE to a called routine. If you are on a 16- or 32-bit machine, then only the low-order 32 bits are actually passed. On a 64-bit machine, all 64 bits are passed. This provides a portable solution to the problem of passing "long" data. This change does not affect any existing programs.

For related topics, see the

- "-Dw" option in section 2.1.9 of the *ACUCOBOL-GT User's Guide.*

- "USAGE clause," section 5.7.1.8 of the *ACUCOBOL-GT Reference Manual.*

- CALL RETURNING syntax in the entry for the CALL statement in section 6.6 of the *ACUCOBOL-GT Reference Manual.*

- next section on “RETURN-CODE Changes.”

RETURN-CODE changes

As discussed in the previous section, the special register RETURN-CODE has changed. In versions of ACUCOBOL-85 prior to 2.3, RETURN-CODE was implicitly defined as:

```
77  RETURN-CODE   PIC S9(9)  COMP-5, EXTERNAL.
```

In Version 2.3 and later, it is defined as:

```
77  RETURN-CODE   SIGNED-LONG, EXTERNAL.
```

This change should have no noticeable effect on existing code, but it allows RETURN-CODE to be used sensibly on 64-bit machines. This change is inhibited if you compile for compatibility with a prior version of ACUCOBOL-85. For example, if you use “-C21” to maintain source compatibility with Version 2.1, then this change does not take place.

There is also a special register that redefines RETURN-CODE called RETURN-UNSIGNED. Its definition is:

```
77  RETURN-UNSIGNED
    REDEFINES RETURN-CODE  UNSIGNED-LONG, EXTERNAL.
```

You should use RETURN-UNSIGNED when handling pointer or “unsigned long” data types that are returned from an external routine. If you use RETURN-CODE in these cases, you can get errors if the value is large enough to set the high-order bit of RETURN-CODE. The problem is that these values are negative when interpreted as signed values, therefore COBOL will remove the sign if you move them to an unsigned destination.

The RETURN-UNSIGNED special register is not defined if you compile for compatibility with prior versions of ACUCOBOL-85.

Runtime changes

For machines using MS-DOS, ACUCOBOL-GT Version 3.1 and later versions do not run under standard 16-bit DOS but do support 32-bit Extended DOS. Version 3.0 requires MS-DOS version 3.0 or later. ACUCOBOL-85 Version 2.1 required only MS-DOS version 2.0.

C.23 Changes Affecting Version 2.0

The following section details changes that can affect programs originally written for the Version 2.0 ACUCOBOL-85 compiler.

Compiler changes

The "-Ca" compiler flag was still available in Version 2.0, but was not documented. (It was synonymous with the "-Va" flag.) Since Version 2.1, "-Ca" has a new and different meaning. So, you must now use "-Va" to cause opposite video intensities to be used for ACCEPT and DISPLAY statements.

C.24 Changes Affecting Version 1.5

The following sections detail changes that can affect programs compiled with the Version 1.5 ACUCOBOL-85 compiler.

Compiler changes

Note: All of the changes described in this section can be inhibited with the "-C5" compile-time option, which causes the compiler to use ACUCOBOL-85 Version 1.5 semantics. The "-Z5" option (which produces object files compatible with Version 1.5) will also inhibit these changes.

- Since the release of Version 2.0, indexed, relative, and binary sequential files can have variable-length records. You might have syntax in your existing programs that implies variable-length records, even though your files on disk are fixed-length. If this is the case, you will receive error "39" when you try to open your existing files after recompiling your programs. This type of error will occur most frequently with files that have multiple records declared for them (more than one "01" entry in the file's FD). In order to prevent the error, compile with either the "-C5" or "-Cf" compile-time option. The "-Cf" option causes the compiler to assume fixed-length records for these kinds of files.

- The function of the RETURN-CODE special register was expanded in Version 2.0. This register is used to return a status value to the operating system or calling program. The return status of the SYSTEM library routine is also stored here. This can cause an existing program to behave differently if you set RETURN-CODE to a particular value and then call the SYSTEM routine. This can also cause programs that return zero to the operating system (the default value of RETURN-CODE) to return a non-zero value if they call SYSTEM. Note that this change affects programs only after they have been recompiled with Version 2.0 or later. You can inhibit the change with the "-C5" compile-time option.

- The CALL PROGRAM verb behaves differently since Version 2.1. If you used CALL PROGRAM under Version 1.5, use the "-C5" option to maintain compatibility when you compile with Version 3.0. Also note, that since Version 2.1 the "-Ci" option implies the recursive PERFORM switch "-Zr".

- Under Version 1.5, the "-Vc" compile-time option caused ACCEPT statements that entered *numeric* fields to be treated as if the CONVERT phrase were specified for them. Since Version 2.1, this option also implies the CONVERT phrase for *numeric edited* fields.

- Under Version 1.5, the WRITE and REWRITE verbs did not check the length of the record for legality. Since Version 2.1, an illegally sized record returns error "44".

- The option "-Zz" causes spaces in a USAGE DISPLAY numeric item to be treated as the value zero. Because this action was formerly handled by the SPACES-ZERO runtime option, if you have a mix of object files from Version 3.0 or later and from any versions prior to Version 2.0, then you should use "-Zz" to create the new objects and should also add the SPACES-ZERO option to your runtime configuration file to handle prior versions.

Runtime changes

The changes described in this section take effect when you install the latest runtime system.

- **Important:** Beginning with Version 2.0 and continuing through Version 3.1, the ACUCOBOL-GT runtime was delivered with Version 3 of the Vision file system. Version 3.2 and later versions are delivered with

Vision Version 4. The Vision file system is used on all ACUCOBOL-GT implementations except VAX systems running VMS and Alpha Micro systems running AMOS. Vision Version 3 introduced a new file format that is portable across all machines, and is (generally) smaller. Vision Version 4 introduced a dual file format, in which the indexes are kept in a separate file from the data. When you are installing the latest version of the runtime system, you have three choices:

a. You can leave your existing data files in place. ACUCOBOL-GT will continue to use them. However, any new data files created by Version 3.2 or later will have the new Vision Version 4 format. This is the default behavior.

b. You can convert all of your files to the new format with the "rebuild" option of "vutil". In particular, running "vutil -rebuild -3" on your data files will convert them to the Vision Version 3 format, and running "vutil -rebuild -4" on your data files will convert them to the Vision Version 4 format.

c. You can continue to use the old format for all of your data files, including any newly created ones. To do this, add the line:

```
V-VERSION 2
```

to your "cblconfig" file. This will ensure that any newly created files use the old format.

- The default method of editing numeric and numeric edited fields on the screen changed slightly when Version 2.0 was released. In Version 1.5, when a user was editing an existing value, the user could type over the value. This left any trailing digits in place, and sometimes caused confusion. Beginning with Version 2.0, if the user starts typing over an existing field, the current contents are erased first. If the user instead starts by editing the field (by using an arrow key or an editing key), then the default value remains on the screen and the user can modify it.

  This behavior is controlled by the "NUMERIC-UPDATES" and "EDITED-UPDATES" configuration options. If you already have the following entries in your configuration file, then the default change will not affect you. If you do not have these entries and want to maintain exact compatibility with Version 1.5, then you should add the following to your configuration file:

```
SCREEN  Numeric-Updates=Converted
Edited-Updates=Converted
```

C.25 Changes Affecting Version 1.4

The following sections describe changes that can affect programs compiled with the Version 1.4 ACUCOBOL-85 compiler. These are the same changes that occur when you move from Version 1.4 to Version 1.5.

Compiler changes

The following changes can affect programs when they are re-compiled. **Note that all of these changes can be suppressed by the "-C4" compile-time option,** which causes the compiler to use Version 1.4 semantics. Also note, that the "-Z4" compile-time option (which produces 1.4 compatible object files) will also inhibit these changes. Note that there are several important changes, especially if you are using VAX COBOL compatibility mode. You should use "-C4" until you can evaluate the extent to which these changes affect your programs.

- **Important:** Under Version 1.4, USAGE BINARY data items are treated as identical to USAGE COMP-1 data items. Since Version 2.1, USAGE BINARY items are treated as defined by the ANSI standard. This results in data items that are different except for data items described as PIC S9, S9(2), S9(3) or S9(4). If you have any USAGE BINARY data items in files, you will need to specify "-C4" to maintain compatibility with your existing files until you can change your programs.

- **Important:** The internal format of COMPUTATIONAL data items is different under the following circumstances:

  a. You are using VAX COBOL compatibility mode; or

  b. You use the "-Zb" or "-Db" compile-time options.

  Under previous versions, a data item that fit one of these conditions is stored as a COMP-1 data item if it is small enough (PIC S9(4) or smaller), otherwise, it is stored as a COMP-2 data item. Since Version 2.1, these items are stored as BINARY. This is the same as COMP-1 for the small data items, but is different for the larger ones. If either of these

cases applies to your programs, and you store COMPUTATIONAL data items in files, then you should use "-C4" to maintain compatibility with your files until you can modify your programs.

- In previous versions of ACUCOBOL-85, COMP-3 data items are always treated as signed. They are also rounded up to an odd number of digits. Beginning with Version 2.0, they act as described by their picture clauses.

- In Version 1.4, COMP-6 data items always have an even number of digits. Since Version 2.1, they have the number of digits specified in their picture clauses.

- Since the release of Version 2.1, specifying CONVERT on a DISPLAY of a numeric edited data item causes that item to have its leading spaces stripped and causes the item to be justified according to the rules applied to numeric data items. Under Version 1.4, output conversion of numeric edited items has no effect.

- In Version 1.4, specifying the CONTROL KEY phrase or the ON EXCEPTION Key-Name phrase for an ACCEPT statement implies automatic termination of a field when that field is filled. Since the release of Version 2.1, this behavior is specified by the AUTO phrase. Because of the nature of the ACCEPT rules, this change does not affect programs using RM/COBOL compatibility mode.

- Versions of ACUCOBOL-85 prior to 2.1 do not support file errors 14 or 24 for relative files when the relative key data item is too small to hold the relative record number. Version 2.1 and all later versions return the appropriate error in this case.

- Since Version 2.1, assigning a file to the device name PRINTER without explicitly assigning an external file name causes the file to be assigned to "PRINTER" when you are using VAX COBOL compatibility mode. Under previous versions, the file is assigned to the same name as its internal file name.

- In Version 1.4, the SYNCHRONIZED clause has no effect. Since Version 2.1, data item synchronization occurs.

- The rules for the meaning of the ON EXCEPTION phrase of the ACCEPT statement have changed. For versions prior to Version 2.0, this phrase catches numeric conversion errors. If the Key-Name option is used, it also catches exception keys. Since Version 2.0, it always catches exception keys and does not catch numeric conversion errors (these errors are handled automatically by the terminal manager). Specifying "-C4" or "-Ve2" retains the original meaning of this phrase. Programs using RM/COBOL compatibility mode are unlikely to be affected by this change.

- Since Version 2.0, closing a window moves the cursor to the position it occupied when that window was created. Before Version 2.0, the cursor moved to the home position of the restored window.

- Many new reserved words have been added since Version 2.0. Most of these can be treated as user-defined words through use of the new "-Rs" and "-Ri" compile-time options. A few new words not covered by these options have also been added. If they conflict with your current programs, you can individually treat them as user-defined words with the "-Rw" option.

- Several compile-time options were renamed in Version 2.0. The original names are still supported, however, so this change does not affect existing programs or compile scripts, except for the "-Ca" option described earlier.

Runtime changes

The following changes occur when the latest runtime is installed.

- The default meaning of the Tab key has changed. Under Version 1.4, the Tab key is an exception key that has a key value of "9". Beginning with Version 2.0, the Tab key is a termination key with a key value of "9". The only difference is that under the previous version, the Tab key is allowed only when exception keys are allowed and it causes the ON EXCEPTION phrase to execute. If you depend on this behavior, you can add the following line to your configuration file:

  ```
  KEYSTROKE Exception=9 ^I
  ```

  This change was made so that the Tab key could function as a "next field" key when you are using the Screen Section.

- Several other keys were redefined in Version 2.0 for use with the Screen Section. These changes do not affect existing programs, however, because the new defaults have the same effects as the old ones when used with field-level ACCEPT statements.

- The maximum number of files that can be opened by the runtime was reduced from 64 to 32 in Version 2.0. This was done to save memory. If you need more than 32 files, you can set the maximum to any value you want (up to 255) with the **MAX_LOCKS** configuration option. See Appendix H for details.

- A subtle change has been made in the processing of the user's environment. In previous versions, an entry in the user's environment always takes precedence over an entry in the runtime's local environment. Beginning with Version 2.0, an entry in the user's environment takes precedence at the time the local environment is initially created. This change allows the SET ENVIRONMENT verb to have an affect on an entry initially defined in the user's environment.

- Since Version 2.1 the cursor does not leave the field when the field is filled. Instead, it stays in the last character position and inhibits further data entry. This difference is cosmetic, but if you prefer the method used by previous versions, you can add the following line to your configuration file:

```
KEYBOARD  Cursor-Past-End=Yes
```

C.26 Changes Affecting Version 1.3

If you are upgrading directly from Version 1.3, then several changes affect you. These changes are the same as those you encounter when you move from Version 1.3 to Version 1.4, except that the current runtime does not support linked object files produced by the Version 1.3 compiler.

Compiler changes

The following changes affect programs when they are re-compiled. **You can specify the "-C3" option to suppress these differences.** Note that specifying "-C3" also implies the "-C4" flag discussed above. You can also produce Version 1.3 object files with the "-Z3" compile-time option.

- Under Version 1.3, a line sequential file accessed by a program compiled with RM/COBOL compatibility mode automatically has short records padded with spaces to fill the record area. Beginning with Version 2.0, only line sequential files with automatic trailing space removal have their records padded with spaces. This change was made to accommodate the behavior of RM/COBOL-85.

- A numeric data item that is the object of a DISPLAY statement with the CONVERT option is left-justified when RM/COBOL compatibility mode is used under any version since 2.1. In Version 1.3, the data item is right-justified. This change was made to accommodate the behavior of RM/COBOL-85.

- Under Version 1.3, the default SIZE of an ACCEPT field is always equal to the number of assignable character positions in the data item, plus 1 if the data item is signed, and plus another 1 if the data item contains digits to the right of the decimal point. Beginning with Version 2.0, this amount is used only if the destination is numeric or edited and the CONVERT phrase is used. Otherwise, the default SIZE is the physical size of the receiving field. The difference is subtle and is unlikely to affect any current programs. This change was made to better simulate the behavior of RM/COBOL.

- In RM/COBOL compatibility mode, a field accepted with the ECHO phrase is redisplayed in a converted form only if the UPDATE phrase is also used. In Version 1.3, the field is redisplayed in a converted form only if the CONVERT phrase is used. This change was made to better simulate the behavior of RM/COBOL.

- Certain line sequential files now have automatic trailing-space removal applied to them. This depends on the device type specified in the file's ASSIGN clause. This will generally not affect existing programs except that files with automatic trailing space removal may not be opened for I/O (due to the unpredictable record size). This affects only those programs that do REWRITEs on sequential files. If you have a program

that does REWRITEs on a sequential file, you should check to make sure that the device type is not one that specifies automatic trailing space removal. For more information, see Book 3, *ACUCOBOL-GT Reference Manual*, section 4.3.1, "FILE-CONTROL Paragraph," under General Rules.

Runtime changes

The following changes occur when the latest runtime is installed. These changes can generally be compensated for by various configuration options.

- Since Version 3.0, the runtime does not support linked object files produced by the Version 1.3 compiler. If you have any linked object files, then you must convert them to the library format introduced in Version 1.4. Note that the normal object files produced by the 1.3 compiler are still supported.

- The default keyboard configuration has changed. The new default is very similar to the default RM/COBOL configuration. Also, the KEY-MAP and EDIT-MODE configuration variables are no longer supported. These have been replaced by the more powerful KEYBOARD and KEYSTROKE entries. Most users of Version 1.3 ACUCOBOL-85 reconfigured the keyboard with the KEY-MAP variable to simulate the RM/COBOL keyboard. Most will not need to make any changes since this is the new default.

- Users who used the default ACUCOBOL-85 keyboard under Version 1.3 will have to reconfigure the keyboard to meet the Version 1.3 standard. Other users may need to make minor changes to match their previous configuration. For details on the new default configuration and the KEYBOARD and KEYSTROKE variables, see the *ACUCOBOL-GT User's Guide*, section 4.3.2, "Redefining the keyboard." Also, see the sample configuration file supplied with the compiler.

- Under Version 1.3, files opened with the EXTEND phrase are automatically created if they do not exist. Beginning with Version 2.1, they are not. This change was made to match the ANSI standard. You can maintain the Version 1.3 behavior by setting the configuration variable "EXTEND-CREATES" to "1" in the configuration file.

- In VAX COBOL compatibility mode, a missing file opened for I/O is not automatically created. Under Version 1.3, it was. This change was made because the most recent release of the VAX COBOL compiler was changed this way.

- Several VAX COBOL file status codes have been changed. This change was made to match changes made to the VAX COBOL compiler.

- When you are using the RM/COBOL-85 or RM/COBOL version 2 file status codes, a corrupted indexed file is now returned as file error “98” instead of file error “30”.

- A single DISPLAY may now wrap around more than one screen row. Under Version 1.3, lines are truncated. If the 1.3 behavior is desired, set the configuration variable “WRAP“ to the value “0”.

- An ACCEPT or DISPLAY statement that references a row past the bottom edge of the window now causes that window to scroll. Under Version 1.3, the statement is (largely) ignored. You can cause a similar effect by setting the configuration variable “SCROLL“ to “0”.

- The syntax of the COLOR-MAP configuration variable has changed slightly. See the *ACUCOBOL-GT User’s Guide*, section 4.4.1, “Adding Color.”

- Object files produced by versions of ACUCOBOL-85 *prior* to Version 1.3 may not be executed by the latest runtime system. These programs must be recompiled with a 1.3 (or later) compiler. This change was made to reduce the size of the runtime system and to improve its performance. You can use the “-info” option of “ccbl” to locate object files created by a pre-1.3 version of ACUCOBOL-85. These will be object files that contain a “vers” value of “2” or less.

D Compiler Error Messages

Key Topics

D.1 Introduction

The ACUCOBOL-GT compiler produces a wide range of informative messages, including both Errors and Warnings. An Error message is more severe than a Warning and, unlike a Warning, inhibits production of an object file by the compiler.

The following list contains the Error messages produced by the compiler. In many cases, the meaning of an error message is clear from the message itself. Where this is not the case, a brief explanation follows the message. In this listing, the term "%s" represents some string that will replace the "%s" before you see the message. In most cases, the string will be a user-defined value, such as a file name, a record name or an item name.

The listing is in alphabetical order. Note, however, that the first few pages list messages that begin with dynamically generated strings. The alphabetical ordering ignores the string (which replaces the "%s" in the listings). Therefore, if an error message starts with a dynamic string, look it up in this list by using the generic portion of the message that follows the string.

D.2 List of Errors

$

"$ELSE without a corresponding $IF"

"$END without a corresponding $IF"

Either $ELSE or $END was encountered, and there is no corresponding $IF.

%

"%s: a section and a paragraph have the same name"

A section name may be the same as a data name, but must otherwise be a unique user-defined word.

"%s and %s must be the same size"

"%s cannot be moved to ALPHABETIC"

"%s cannot be moved to ALPHANUMERIC"

"%s cannot be moved to ALPHANUMERIC EDITED"

"%s cannot be moved to NUMERIC"

"%s cannot be moved to NUMERIC EDITED"

"%s contains no input fields"

You have attempted to ACCEPT a screen item that includes no TO or USING phrase.

"%s: Data item > 64K illegal here"

"%s: data item exceeds 2GB"

The maximum data item size in ACUCOBOL-GT is 2 GB. See **Section 5.1.6, "Large Data Handling,"**, in Book 3, *ACUCOBOL-GT Reference Manual*. For a list of compiler limits, see **Section A.2, "Limits and Ranges"**.

"%s expected, %s found"

"%s: File record exceeds 64MB"

The maximum record size allowed in ACUCOBOL-GTprograms compiled to Version 6.0 object format or later is 64 megabytes. See **Section A.2, "Limits and Ranges"**.

"%s: File record exceeds 32K"

The maximum record size allowed in ACUCOBOL-GT programs compiled to Version 5.2 or earlier object format is 32 kilobytes.

"%s ignored for OPEN INPUT"

"%s: illegal level 77"

Level-number 77 entries may not have subordinate items except for level 88 items.

"%s: incorrect number of arguments"

"%s: incorrect size for KEY AREA"

The KEY AREA must be in multiples of seven.

"%s is ambiguous"

The name here could be interpreted to be more than one thing.

"'%s' is an invalid destination"

Data cannot be stored in a literal value.

"%s is not a KEY of %s"

SEARCH ALL requires that the compared item be referenced in the KEY IS phrase in the OCCURS clause of the searched table.

"'%1' is not a property or method of 'CLASS %2' "

"%s is not a START key of %s"

"%s is not numeric"

"%s: key must not be in a table"

The data item specified in the KEY phrase of a SORT or MERGE statement may not be subordinate to an OCCURS clause.

"%s may not be used as a CODE-SET"

A Format 2 Alphabet entry may be used in a COLLATING SEQUENCE phrase, but not in a CODE SET phrase.

"%s may not belong to %s"

Key-table of the KEY AREA phrase of the SORT verb must name a data item that is not located in the record for sort-file.

In an INSPECT CONVERTING statement, the convert-string must be the same length as the compare-string.

"'%1' must be a 'get' property of '%2'"

"'%1' must be a 'put' property or method of '%2'"

"%s must belong to %s"

The data item specified in the KEY phrase of a SORT or MERGE statement must be a data item in the record description associated with sort-file.

"%s: must have only one value for SEARCH ALL"

A level 88 referenced in a SEARCH ALL statement may not specify a series or a range in its VALUE clause.

"%s: needs INDEXED BY phrase in declaration"

The subject of a SEARCH statement must be a data item that contains an OCCURS clause including an INDEXED BY phrase.

"%s: no FALSE value defined"

You cannot SET cond-name TO FALSE unless cond-name has a WHEN SET TO FALSE phrase associated with its defining level 88 entry.

"%s not a key of %s"

"%s: not a table"

The subject of a SEARCH statement must be a data item that contains an OCCURS clause including an INDEXED BY phrase.

"%s not allowed here"

"%s not an ALPHABET name"

You have attempted to use something in a place where an ALPHABET name must be specified. It has not been defined to be an ALPHABET name.

"%s: not defined"

%s was used in a $IF, but is not defined either with a level 78 item or with a "/CONSTANT" compile switch.

"%s not unique in first 18 characters"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This message occurs if a field name is not unique within the first 18 characters. The "%s" is the name found. You can either change the field name or apply the NAME directive.

"%s: Procedure name not unique"

The paragraph or section you are trying to ALTER, GO TO, or PERFORM has been defined more than once in the program.

"%s record larger than %s record"

In the USING phrase of the SORT or MERGE statement, in-file records may not be larger than sort-file records. In the GIVING phrase, sort-file records may not be larger than out-file records.

"%s: requires version %s runtime"

Some compiler options (like "-Z4" and "-Z5") cause the compiler to generate an object file that can be run on a version of the runtime that is older than the compiler you are using. These compiler options won't allow you to compile new features that the old runtime can't handle. When you attempt to compile such features into an object file for an older runtime, this error will be produced.

"%s: Screen name not allowed in this context"

You have attempted to use a form of the ACCEPT or the DISPLAY verb that does not allow the use of a screen name from the Screen Section.

"%s subject to DEPENDING ON phrase"

If the source or receiving item for a screen entry has an OCCURS clause, it may not include the DEPENDING phrase.

"%s: unknown XFD directive"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

The compiler did not recognize the directive you used. The "%s" is the directive found. Check for a typographical error.

"* %s overflow ***"**

"This table currently allows %s entries"

"Sorry, you cannot make this table any bigger!"

or

"You can increase this with the "-T%s" option"

"For example, you might try "-T%s %s""

The compiler uses several internal tables to which it has given an arbitrary maximum size. Your code requires a greater table size than the default. This message tells you which table maximum has been exceeded and whether you can try recompiling with an increased size. The tables are:

Table	Compile Flag	Default Value
Identifiers/statement - the maximum number of items in each statement	td	4096
Subscripts/statement - the maximum size for OCCURS	te	256

The compiler always suggests double the default value in the error message. Because higher values increase table size (using more memory), the values should not be set any bigger than they need to be.

A

"ACCEPT FROM DATE only returns two-digit year data"

"ACCEPT FROM DAY only returns two-digit year data"

"ALL expected"

"ALL ignored here"

"ALL index not allowed here"

"alphanumeric value expected"

"ALTER para must start with GO TO: %s"

"Ambiguous identifier: %s"

The identifier here could be interpreted to be more than one thing. If two group items use the same field name and the field is referred to in the program, the field name must be qualified by the name of the next higher group item with a unique name.

"Ambiguous symbol: %s"

"Arithmetic expression expected"

"AT value must be 4 or 6 digits"

B

"Bad CHART STATUS definition"

"Bad CRT STATUS definition"

"Bad CURSOR definition"

"Bad picture"

"Bad picture for DATE: *keyname*"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

The PICTURE in a DATE directive must be six or eight bytes in length, either alphanumeric or numeric with no sign.

"Bad SCREEN CONTROL definition"

"Badly formed condition"

"Badly formed ID: %s"

See the rules for COBOL Words, in **Section 2.1.1.1** of the *ACUCOBOL-GT Reference Manual*.

"Badly formed number: %s"

See the rules for Numeric literals, in **Section 2.1.2.1** of the *ACUCOBOL-GT Reference Manual*.

"BY CONTENT parameters exceed maximum size"

For Version 7.0 and earlier, the maximum parameter size is 64K. For later versions, the maximum limit is 2GB.

"BY expected"

The REPLACE statement and the REPLACING phrase require the word BY.

"BY VALUE parameter %s illegal size"

"BY VALUE parameter %s illegal type"

"BY VALUE parameter %s mis-aligned"

"BY VALUE parameter may not be a literal"

"BY VALUE parameter must be an integer"

C

"Can't recover from earlier error, Good bye!"

"Case sensitivity option repeated for same LIKE condition"

Only one of the CASE-SENSITIVE and CASE-INSENSITIVE options should be specified in any one LIKE condition. If the case sensitivity options are specified more than once, the last specification takes precedence.

"CELL phrase used inconsistently"

The CELL phrase appears in either the LINE or CLINE phrase, but not in both. Or, the CELL phrase appears in either the COL or CCOL phrase, but not in both. The CELL phrase must be specified in each of the LINE/CLINE or COL/CCOL phrases (or omitted from the pair).

"Class already specified"

The same category of data may not be specified more than once in the REPLACING phrase of an INITIALIZE statement.

"Class name not allowed here: %s"

"Clause repeated"

"COMP-X/N item too large"

"Compilation aborted"

"Compiled screen description too large"

"Compiler error: Picture"

"Condition name not allowed here: %s"

"Conditional expression expected"

"Configuration: %s"

D

"Data item exceeds 2GB"

The maximum data item size allowed in ACUCOBOL-GTprograms is 2 GB. For a list of compiler limits, see **section A.2, "Limits and Ranges."**

"Data item exceeds 64K"

The maximum data item size allowed in ACUCOBOL-GT programs compiled to Version 5.2 or earlier object format is 32 kilobytes.

"Data item not allowed here: %s"

"Data-item: Redefined data item with value moved"

The compiler generates this error when it detects that a data item with value, already written into the object code, is being redefined too large for the current data segment. If compiled, the resulting COBOL object would attempt to force the runtime to write to memory it has not allocated, likely resulting in a crash.

"Data missing from key segment keyname"

You have compiled with the "-Zx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

Some part of the named key could not be placed in the dictionary. This usually occurs because of filler. For example:

```
01  my-record.
    03  my-key.
        05  filler    pic xx.
        05  field-1   pic xx.
```

If *my-key* is declared as a record key, you will receive this error because the area of the key described by "filler" is not included in the dictionary.

To correct this error, ensure that every character that is part of the key is included in some field that is part of the dictionary. Use an XFD to give a field name to each filler, to ensure that fillers are included.

Example:

```
01  my-record.
    03  my-key.
*(( xfd name=myfiller ))
        05  filler   pic xx.
        05  field-1  pic xx.
```

"Dest may not be edited: %s"

"Different number of SYMBOLIC names and values"

There must be a one-to-one correspondence between occurrences of "name" and "number" in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

"Directive word too long: *keyname*"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

With one exception, the words contained in a directive, including field names, cannot exceed 30 characters. The value of a WHEN directive may consist of up to 50 characters. You have exceeded the limit.

"Disk full"

"Duplicate ACCESS"

"Duplicate ASSIGN"

"Duplicate ENTRY point name: %s"

The ENTRY point name has already been used in this program.

"Duplicate interface '%s'"

"Duplicate NOT %s phrase"

Various NOT phrases may be used in the context of specific statements (READ ... NOT AT END, COMPUTE ... NOT ON SIZE ERROR). The compiler has encountered more than one such NOT phrase in a statement.

"Duplicate ORGANIZATION"

"Duplicate paragraph name: %s"

The paragraph or section you are trying to ALTER, GO TO, or PERFORM has been defined more than once in the section.

"Duplicate RECORD KEY"

"Duplicate STATUS"

E

"edited item too large"

The maximum edited data item size in ACUCOBOL-GT is 255 bytes.

"ELSE, END-IF or '.' required after NEXT SENTENCE"

"END-PERFORM required"

The compiler has encountered code in which a scope delimiter is required for a PERFORM statement.

"Entry for product 'compiler' in file '%2' is corrupt"

The license file contains garbled information about the compiler product. The compiler cannot be executed until the license file has been repaired. Contact Technical Support.

"Error file name is the same as the source name"

Your **ccbl** command has instructed the compiler to name one of its output files with the same name you've given for the source code file.

"Error: input file is Vision format"

You have specified a Vision format file as input to "vutil -load". The *source* file must be the name of the binary, relative, or line sequential file. See **Section 3.3.10** of the User's Guide for details on this command.

"Errors found, size information suppressed"

"EVALUATE nesting level exceeded"

The maximum depth of EVALUATE statement nesting is 10 levels.

"Evaluation version - expires %1/%2/%3"

"Exception handlers require recursion (-Zr)"

"EXIT SECTION outside of SECTION"

EXIT SECTION must be used within a SECTION.

"Expecting condition after NOT"

"EXTERNAL file in SAME AREA illegal"

"EXTERNAL in REDEFINES"

The REDEFINES clause and the EXTERNAL clause may not be applied to the same data item.

"EXTERNAL name must be unique"

The same name may not be given to more than one file or data item that is declared EXTERNAL within a program.

"Extra segment exceeds 64K"

The "extra segment" is that part of the object file that contains descriptors and other miscellaneous elements. This category is restricted to 64 KB. The main factor here is the number of different items that are referenced in the Procedure division.

F

"FD already defined for file"

"Field xxx causes duplicate database data"

This is a warning message that can appear if you compiled with the "-Fx" option. The data dictionary was built, and the interface will operate correctly. The warning informs you that your record definition should be restructured. Your current definition is set up in such a way that:

- you have overlapping key fields, and
- both keys must be represented in the database as *separate items*.

The interface will handle this situation correctly. It will keep the overlapping keys updated simultaneously, so that they always have the same value. However, the warning alerts you that *you have the same data represented twice in the database*. This is dangerous, because someone at the site might access the database via SQL and accidentally change only one of the keys.

Here's an example of the problem, and a description of how to correct it (the example assumes that both **key-1** and **key-2** have been declared as keys):

```
01 order-record.
   03  key-1.
       05  field-a         pic x(5).
       05  field-b         pic 9(5).
       05  key-2
       redefines field-b    pic x(3).
```

This example will generate the warning message.

Because "key-2" is a key, it must also be represented in the XFD. It doesn't correspond exactly to any other data field, so it must be entered as a separate field in the XFD.

In the COBOL view of the file, "key-1" and "key-2" overlap. But the requirements of XFD storage force the same data (known to COBOL as "field-b") to be physically represented *twice* in the XFD. Any updates to the data from any ACUCOBOL-GT program will correctly update both fields. Updates from *outside* of ACUCOBOL-GT carry no such guarantee.

In this example, you can correct the situation by breaking "field-b" into two columns, so that "key-2" corresponds exactly to another data field:

```
01 order-record.
   03  key-1.
       05  field-a           pic x(5).
       05  field-b.
           07  field-b1      pic x(3).
           07  field-b2      pic 9(2)
       05  key-2
          redefines field-b  pic x(3).
```

"Figurative constant not allowed: %s"

A figurative constant (zero, space, quote, etc.) cannot be used in this context.

"Figurative constant not allowed here: %s"

"File %s in multiple areas"

The named file appears in more than one SAME RECORD AREA clause.

"File %s undefined"

A file named in the I-O-CONTROL paragraph must be defined by a SELECT clause in the FILE-CONTROL paragraph.

"File must be a SORT file"

"FILLER cannot be EXTERNAL"

"Floating-point literal not allowed here"

"Floating-point VALUE not allowed here"

"FOOTING larger than page size"

"FOOTING must be > 0"

"FROM/TO/USING error"

FROM, TO, and USING can be used only once each in a particular screen item description. USING cannot be used with either FROM or TO in the same description.

"Function argument %s must be alphanumeric"

"Function argument %s must be numeric"

G

"GIVING data item for file %s is too small"

"GROUP expected after USE"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

The "use group" directive must include both words.

I

"ID greater than 60 characters: %s..."

"%s" is truncated to 60 characters.

"Identifier expected, %s found"

"Identifier unresolved: %s OF %s"

An attempt to qualify an identifier, as in "field of record", failed. Check the spelling of the qualifier.

"Illegal ACCESS"

SEQUENTIAL is the only ACCESS MODE legal with ORGANIZATION SEQUENTIAL.

"Illegal arithmetic expression"

"Illegal BLANK ZERO"

The BLANK WHEN ZERO clause can be used only for numeric or numeric edited elementary items. The picture of the item may not include "*" or "s".

"Illegal class condition"

"Illegal clause(s) for sort file"

"Illegal color value"

The FOREGROUND-COLOR and BACKGROUND-COLOR phrases take a literal in the range 0-7.

"Illegal COMPRESSION value"

The COMPRESSION factor may not be greater than 100.

"Illegal condition"

"Illegal hex literal"

Hexadecimal literals must consist of the digits "0"-"9" or "A"-"F".

"Illegal indicator: '%s'"

The indicator area of a COBOL line may contain " ", "-", "*", "$", "/", "D", or, in Terminal mode, "\D".

"Illegal INITIALIZE item: %s"

The destination may not contain a RENAMES clause.

"Illegal JUSTIFIED clause"

The JUSTIFIED clause can be applied only to alphabetic and alphanumeric data items. It must be applied to an elementary item, rather than to the group item.

"Illegal KEY phrase"

The KEY phrase of a READ statement can be used only with ORGANIZATION INDEXED files and cannot be used with a READ NEXT or a READ PREVIOUS.

"Illegal level for OCCURS"

The OCCURS clause may not be used on a level 01 or a level 88 item.

"Illegal level number"

A level 77 item cannot be included in the File Section or in the Screen Section. The Screen Section allows only levels 1-49 and level 78.

"Illegal level number: %s"

"Illegal OCCURS value"

In a Format 1 OCCURS, the table-size must be more than zero. "Illegal or missing ASSIGN variable of %s"

"Illegal or missing BOTTOM variable of %s"

"Illegal or missing DEPENDING ON variable of %s"

"Illegal or missing FOOTING variable of %s"

"Illegal or missing KEY variable of %s"

"Illegal or missing LINAGE variable of %s"

"Illegal or missing PADDING variable of %s"

"Illegal or missing STATUS variable of %s"

"Illegal or missing TOP variable of %s"

"Illegal parameter: %s"

"Illegal parameter: literal"

Generated by the compiler if low-values or other figurative constants are passed to ActiveX or COM methods or properties as parameters where the method or property expects a "by reference" parameter: This is the same error message you get when passing a figurative constant as a USING parameter in a CALL statement. One way to tell that the ActiveX/COM method expects a "by reference" parameter is by viewing the entry in the COPY file for that control or object. If the type has "BYREF" or if the numeric value divided by 16384 is odd, then you may not pass a figurative constant.

"Illegal picture: %s"

"Illegal POINTER: %s"

"Illegal receiver for source type: %s"

The compiler has encountered an illegal MOVE.

"Illegal REDEFINES"

"Illegal RENAMES"

"Illegal replacement size: %s"

In an INSPECT REPLACING statement, the replace-string must be the same length as the target-string.

"Illegal SD clause: %s"

"Illegal sign condition"

The sign condition can be applied only to an arithmetic expression.

"Illegal SIGN/USAGE for file with CODE-SET"

"Illegal source type for CONVERSION"

The compiler will allow MOVE WITH CONVERSION only of alphanumeric items.

"Illegal statement in current declarative"

Occurs when a program attempts to execute a disallowed statement in the context of a USE FOR REPORTING declarative or a file declarative that has been triggered by a status "22" for a file open with BULK-ADDITION. In both of these cases, the declarative is triggered as part of the file operation (instead of after the operation completes) and several restrictions apply. The program may not perform any file operations or start or stop any run units (including chaining). In addition, the program that contains the declarative may not perform an EXIT PROGRAM.

Note: The program continues running after printing this statement (halting the program at this point would corrupt the data file).

This error message indicates a programming error that should be corrected. There is no way to disable the error message. You can find the offending statement by running the program under the debugger. When the statement executes, the runtime will break to the debugger with this message and place the cursor at the statement.

"Illegal table size: %s"

Your compiler command line has specified an illegal value for a user-resizable table ("-ta", "-tb", etc.). See the internal table list near the beginning of this section.

"illegal USAGE"

"Inconsistent picture"

"INDEXED key not in record: %s"

The key to an Indexed record must be defined within the record.

"INDEXED key outside of smallest record: %s"

In an Indexed file with variable record size, the offset of the end of the key must be within the bounds of the smallest possible record size. Multiple record definitions (01 levels) within a file description may generate a variable length record file.

"Indexing not allowed in this context"

"INSPECT TRAILING syntax error"

"Interface definition '%s' not found"

"* Internal error #%s ***"**

The compiler has encountered a syntax error for which it does not have a useful descriptive message. Anytime you get such a message from the compiler, notify Technical Support. If we are already aware of your particular syntax problem, we can tell you what to fix in your source. We may even have a more recent version of the compiler that detects the error more elegantly. If we have not been made aware of this oversight, your call will allow us to find and correct it.

"INTO identifier may not be reference modified"

In a STRING ... INTO statement, the destination may not be in the form "... INTO dest-field (2 : 4)". If you want to start modifying the destination field at a position other than the leftmost, use the POINTER phrase.

"Invalid CODE-SET file type"

CODE-SET may be specified in the FD of sequential files only.

"Invalid CLSID '%s'"

"Invalid directive syntax"

The $SET directive was used incorrectly.

"Invalid GIVING data item for file %s"

"INVALID KEY illegal in this context"

The INVALID KEY phrase of the DELETE statement may not be used with a file declared ACCESS MODE SEQUENTIAL. The INVALID KEY phrase of the REWRITE statement may not be used for ORGANIZATION SEQUENTIAL or ORGANIZATION RELATIVE files if either uses ACCESS MODE SEQUENTIAL.

"Invalid switch number: %s"

The switch named in the SPECIAL-NAMES paragraph must be one of SWITCH-1 through SWITCH-26, SWITCH 1 through SWITCH 26, or SWITCH "A" through SWITCH "Z".

"Invalid syntax in COPY statement"

K

"Key bigger than 250 bytes: %s"

The maximum indexed key size in ACUCOBOL-GT is 250 bytes.

"KEY must be first: %s"

More than one KEY IS phrase is allowed in each OCCURS clause. If one KEY IS phrase references the data-name of the entry that contains the OCCURS clause, it must be the first KEY IS phrase in the clause.

"KEY not found in table: %s"

The key named in the KEY IS phrase of the OCCURS clause must be contained within the table.

L

"Large REDEFINES of a regular variable with a value: %1 redefines %2"

"LENGTH ignored in this context"

"License file '%s' inaccessible"

The license file cannot be located. The message displays the name of the license file that the compiler is trying to locate. The compiler cannot execute without a valid license file.

"License file '%s' is invalid"

"LINAGE must be > 0"

"LINAGE required for END-OF-PAGE processing"

"LINAGE requires SEQUENTIAL organization"

"LINAGE-COUNTER is a reserved data item"

"LINKAGE not listed in USING: %s"

An item defined in the Linkage Section is not referenced in the USING phrase of the Procedure Division statement.

"Listing file name is the same as the source name"

Your **ccbl** command has instructed the compiler to name one of its output files with the same name you've given for the source code file.

"literal expected"

"Literal must be alphanumeric"

"Literal too long"

Prior to version 1.5, an ALL literal not associated with another data item had to be a single character.

M

"May not be a SEQUENTIAL file"

A Format 1 DELETE statement may not be used on a file with ORGANIZATION SEQUENTIAL. "May not be alphanumeric: %s"

"May not be alphanumeric edited: %s"

"May not be edited: %s"

"May not be floating-point: %s"

"May not be numeric: %s"

"May not be numeric edited: %s"

"May not INQUIRE on style %s"

You may not use a style name in the INQUIRE statement. You can only inquire the value of an element of a control. Because styles do not have values, using a style name with INQUIRE is not meaningful.

"May not modify or invoke ActiveX Controls in DISPLAY"

"May not specify both LINES and SIZE"

While it is acceptable to specify both height and length for a BOX, a LINE can have only one dimension.

"meaningless WHEN phrase"

"MERGE illegal in DECLARATIVES"

"Mismatching OCCURS structure"

If an OCCURS clause applies to a screen entry with TO or USING, the receiving item must have an OCCURS of the same number. With FROM, the source item must have an OCCURS of the same number or no OCCURS at all.

"Missing ')'"

"Missing ASSIGN clause"

"Missing closing quote"

A quoted string must have both opening and closing quotes.

"Missing continuation line quote"

If a continued line ends with a nonnumeric literal without a closing quotation mark, the first non blank character in Area B of the continuation line must be a quotation mark.

"Missing COPY file: '%s'"

The filename specified after the word COPY is not found in the directory in which it is expected. Consider whether it is spelled correctly, or check your COPYPATH environment variable.

"Missing COPY *filename*"

The filename specified after the word COPY is not found. Consider putting the file name in quotes.

"Missing directive"

The $SET directive was used incorrectly.

"Missing END-%s"

Several statements in COBOL, among them IF, SEARCH, PERFORM, and EVALUATE, can have their scope delimited by the END- (END-IF, END-SEARCH, END-PERFORM) phrase. The compiler has encountered code in which such a scope delimiter is required.

"Missing exponent"

A digit in the range 0-9 must follow the E in a floating point literal.

"Missing field name after WHEN"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

A valid field name, or the word OTHER, must be specified with the "when" directive.

"Missing KEY phrase in definition: %s"

"Missing library filename"

The OF phrase of the COPY statement has no pathname specified.

"Missing operand"

"Missing or invalid object expression"

"Missing PARA/SECTION" "Can't recover, good bye!"

"Missing period"

"Missing RECORD KEY clause"

"Missing SELECT for this file"

"Missing switch number"

The switch named in the SPECIAL-NAMES paragraph must be one of SWITCH-1 through SWITCH-26, SWITCH 1 through SWITCH 26, or SWITCH "A" through SWITCH "Z".

"Missing value"

"Missing WHEN phrase"

The SEARCH statement and the EVALUATE statement always require a WHEN phrase.

"Missing '=' in XFD directive"

You have compiled with the "-Fx" option. The object file was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

The "name" directive requires an "=" sign. The "when" directive requires a comparison operator.

"Mnemonic name required"

Each system-name in the SPECIAL-NAMES paragraph must be associated with a mnemonic name.

"Modification of %s not allowed"

LINAGE-COUNTER may never be explicitly modified by the program.

"Modification of screen item"

It is not legal to ACCEPT or DISPLAY an item defined in the Screen Section using reference modification.

"More subscripts needed: %s"

An item subordinate to an OCCURS clause has been referenced with a number subscripts or indexes less than its level of nesting.

"More than 64K of parameters illegal for this CALL"

"Multiple pictures"

"Multiple USE for %s mode"

The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified in only one USE statement in a Procedure Division.

"Multiple USE for file: %s"

A particular file may not appear in more than one USE statement in a program.

"Multiple USE for OBJECT"

"Must be a GROUP item: %s"

Both the source and the destination of a MOVE CORRESPONDING statement must be group items.

"Must be alphanumeric: %s"

"Must be in Working-Storage or Linkage: %s"

"Must be INDEXED file: %s"

"Must be integer: %s"

"Must be level 01 WORKING-STORAGE for EXTERNAL"

A data item can be declared EXTERNAL only if it is defined in the Working Storage section. The item must be at level 01 or level 77. (Except for the ability to take subordinate items, level 77 is often implied where level 01 is specifically mentioned.)

"Must be SEQUENTIAL"

A file must be ORGANIZATION SEQUENTIAL to CLOSE REEL, CLOSE UNIT, CLOSE NO REWIND, WRITE NO CONTROL, or WRITE ADVANCING.

"Must be size 1 in this context: %s"

"Must be USAGE DISPLAY: %s"

"Must not be subscripted: %s"

N

"Native character specified twice, ordinal value = %s"

When you specify an ALPHABET in the SPECIAL-NAMES paragraph, each character may appear only once.

"Needs DELIMITED BY to use COUNT"

"Needs DELIMITED BY to use DELIMITER"

"NEXT/PREVIOUS illegal for RANDOM ACCESS"

"No entry for product 'compiler' in file '%2'"

The license file does not contain an entry for the compiler product. The compiler cannot be executed until the license file is corrected. Contact Technical Support.

"No FD for %s"

"No records defined for file"

"No SELECT for file: %s"

"Non-native object file produced"

"Not a condition-name: %s"

Only level 88 items may be SET to TRUE or FALSE.

"Not a file: %s"

The USE statement must include INPUT, OUTPUT, I-O, and EXTEND or the name of a file described in the Data Division.

"Not a record of a file: %s"

"Not a record of a SORT file: %s"

The RELEASE statement may act only on the records of sort files.

"NOT, END-ACCEPT or '.' required after NEXT SENTENCE"

"Null ENTRY point name"

The program has used "" as an ENTRY point name.

"number too large"

The maximum numeric data item size in ACUCOBOL-GT is 18 digits.

"Number too large: %s"

"Numeric literal not allowed here"

"NUMERIC test is constant, because of the type of variable being tested"

The compiler issues this warning in cases where the item being tested is a binary item. This test should always return TRUE for binary items, since binary items can't be non-numeric. You can optimize your COBOL program by removing constant tests.

"numeric value expected"

"Numeric VALUE not allowed here"

○

"Object file name is the same as the source name"

Your **ccbl** command has instructed the compiler to name one of its output files with the same name you've given for the source code file.

"Object wrong type for subject"

If the subject of an EVALUATE statement is, or can be evaluated to be, TRUE or FALSE, the object must be a phrase that is, or can be evaluated to be, TRUE or FALSE (e.g., EVALUATE TRUE WHEN a = b ..., or EVALUATE a > b WHEN FALSE ...). Otherwise, the object must be something that would balance the subject in a conditional expression (e.g., EVALUATE field1 WHEN "a" ...).

"OCCURS DEPENDING illegal in Screen Section"

"OCCURS DEPENDING in OCCURS illegal"

"OCCURS DEPENDING must be last in group: %s"

"Offset too large: %s"

When you are using a subscript of the form "(data-item + integer)", the integer can be no greater than 32767.

"Only 1 level of OCCURS allowed"

Nested OCCURS are not permitted in the Screen Section.

"Operation has no effect"

"ORGANIZATION clash"

"* Out of Main Memory! ***"**

P

"Pic 'V' illegal in COMP-X/N"

COMP-X and COMP-N items can be defined with only '9' or only 'X' symbols.

"PICTURE and/or VALUE clash in Screen Section"

The PICTURE and VALUE clauses may not both be specified for the same screen description entry, either explicitly or implicitly by the use of FROM, TO, or USING.

"Picture required for floating-point in this context"

A screen item must have a picture. If you are using a FLOAT item with USING or FROM, give it a picture within the Screen Section.

"Picture too long"

The maximum picture string in ACUCOBOL-GT is 100 characters.

"Pixel AT value must be 8 digits"

When using the AT verb together with the PIXEL verb, 8 digits are mandatory to specify the position, or the variable being used must be a PIC 9(8).

"Positive integer required"

The value for this field must be greater than zero and include no decimal fraction. A subscript or index must be a positive integer.

"PREVIOUS illegal for sequential file"

This refers to ORGANIZATION SEQUENTIAL.

"Procedure name not allowed here: %s"

"Procedure name required"

GO TO must always be followed by a paragraph or section name.

"Program code exceeds 1MB"

The maximum size of the code portion of an ACUCOBOL-GT object file is 1 MB. The size of the program code is largely determined by the size of the Procedure division of the program. If you cannot streamline the instructions in the Procedure division to fit within this restriction, you might split the logic into two programs, one called by the other.

"Program data exceeds 32 segments"

The 1 MB restriction on the program data is monitored in terms of segments. The factors determining this size are as described above.

"Program data exceeds 64K"

The maximum size of the data portion of an ACUCOBOL-GT object file for any version prior to 1.5 is 64 KB. Thus, this restriction might be encountered when you are using the "-Z4" compiler option. Starting with Version 1.5, the maximum program data size is 1 MB. The size of the data is basically the sum of the sizes of the items in the data division (including File Descriptions) and of the literal strings used within the program (including the Procedure Division).

"Program-wide CURSOR already defined"

The CURSOR phrase of the ACCEPT statement may not be specified if a CURSOR phrase is specified in the program's Configuration Section.

R

"Radio Buttons cannot have array elements in a VALUE or USING phrase"

This error is generated at the occurrence of a radio button control in the screen section whose VALUE (or USING) is an array. This is not allowed due to the internal functionality of how data is copied to COBOL data items once the screen section terminates.

"Record belongs to SORT file: %s"

WRITE and REWRITE statements may not apply to records of files described with an SD rather than an FD in the File Section.

"REDEFINES not allowed in Screen Section"

"REDEFINES of an OCCURS item illegal under ANSI"

"Reference modification of numeric function is illegal"

"Reference modifier illegal in this context"

"Reference modifier out of range"

Using reference modification, either the start position is beyond the end of the referenced item, or the calculated end position would be.

"RELATIVE key in record: %s"

The key to a Relative record must be defined outside of the record.

"RELATIVE Key is required"

A relative key must be indicated if ACCESS DYNAMIC or ACCESS RANDOM MODE is specified.

"RELATIVE key must be PIC 9: %s"

"REMAINDER may not be used if any operand is External Floating-Point"

"Repeated OCCURS"

"REPLACING LEADING/TRAILING requires literals"

"REPLACING not allowed on nested COPY"

"RETURN-CODE is a reserved data item"

S

"Screen item subject to OCCURS"

It is not legal to ACCEPT or DISPLAY an item defined in the Screen Section with or subordinate to an OCCURS clause.

"SEARCH ALL must have only one WHEN"

"SEARCH statement missing WHEN phrase"

A SEARCH statement must have a WHEN phrase.

"SECTION required"

The use of Declaratives is part of a Format 1 Procedure Division. The Format 1 Procedure Division requires the use of Sections.

"Segment %s exceeds 64K"

When you are using segmentation, an individual segment may not be larger than 64 KB.

"Segments must be in order"

When segment numbers are used on the SECTION header, they must be used in ascending order.

"SELECT for this file inconsistent with a SORT file"

"SIZE or LINES phrase required"

A DISPLAY LINE statement requires that either the length or height be specified.

"Sorry, multiple TALLYING counters not supported"

"Sorry, this compiler may not be used on a stand-alone basis"

Some of our customers are licensed to include a limited-use version of the ACUCOBOL-GT compiler in their software application for sale to their own customers. Any attempt to activate such a compiler from the command line, rather than from inside the application, will produce an error.

"SORT file not allowed here"

A Sort file is a file described with an SD rather than an FD in the File Section.

"SORT illegal in DECLARATIVES"

"Source name too long: %s"

"Special name not allowed here: %s"

"START illegal for RANDOM ACCESS files"

"START illegal for SEQUENTIAL files"

"Statement too large at code address %s"

The maximum Paragraph size in ACUCOBOL-GT is 32767 bytes. (A statement cannot be larger than the maximum paragraph.)

"Status name not allowed here: %s"

"STATUS variable %s should be X(2)"

"String must be 1 character in context: '%s'"

"Style name not allowed here"

"Subscript may not be table item: %s"

A data item used as a subscript may not itself be subordinate to an OCCURS clause.

"Subscript out of bounds: %s"

The subscript or index on a table entry is less than 1 or greater than the number in the OCCURS that defines the table.

"Subscript required: %s"

An item subordinate to an OCCURS clause has been referenced without a subscript or index.

"Symbol not in LINKAGE: %s"

"Symbol not in WORKING-STORAGE: %s"

"SYMBOLIC name expected"

The SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph must include at least one "name" naming a symbolic character.

"SYMBOLIC value must be between 1 and %s"

The "number" in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph must be in the range of ordinal positions in the alphabet being referenced.

"SYNC not allowed in Screen Section"

"Syntax error"

"Syntax error: %s"

"syntax error scanning %s"

T

"This constant not allowed: %s"

"This evaluation copy of ACUCOBOL has expired!"

"Please call customer support if you would like to upgrade to a full version or if you wish to extend your evaluation period."

"TO value too small in OCCURS"

The maximum value cannot be less than the minimum value for a Format 2 OCCURS clause.

"Too few parameters: %1 required, %2 found"

"Too many ALPHABETS (max 100)"

"Too many delimiters (max 30)"

"Too many destinations (max 30)"

"Too many destinations (max 50)"

"Too many ENTRY points (max 65536)"

The program has more than 65536 ENTRY statements.

"Too many ENTRY point pages (max 65536)"

It would take more than 65536 object file pages to write out the ENTRY point table.

"Too many errors, compilation aborted"

The compiler has a limit on the number of errors it will track on any one compile cycle. Please correct some of the errors encountered to this point, and try again.

"Too many EXTERNAL items (max 256)"

"Too many files open by the current process"

Vision returns this system error (30) when its attempt to create an additional file segment is stopped because the limit imposed by MAX_FILES has been reached. Error code is one of the following: 94,10; 97; or 97,10; depending on the setting of FILE-STATUS-CODES.

"Too many INITIALIZE destinations (max 50)"

"Too many key segments (max 6)"

"Too many keys (max 120)"

"Too many level 01 linkage items (max 255)"

The program has more than 255 level 01 linkage items.

"Too many operands (max 60)"

"Too many parameters: %1 is the maximum, %2 found"

"Too many REPLACING operands (max %s)"

For Version 7.0 and earlier, the maximum number is 30. For later versions, the maximum limit is 256.

"Too many sending items (max 100)"

"Too many source items (max 50)"

"Too many subscripts: %s"

An item subordinate to an OCCURS clause has been referenced with a number of subscripts or indexes greater than its level of nesting.

"Too Many <symbols> (max <symbols>)"

Generated if compiling for debug and the number of symbols is larger than 65535, or the number of bytes in all symbols is larger than 1048560. This latter limit only happens if compiling with -Znn with nn < 80.

"Too many SYMBOLIC CHARACTERS in this clause (max 100)"

"Too many table dimensions (max 15)"

"Too many USING parameters (max 255)"

U

"Unable to find ''%s''"

"Undefined data item: %s"

The data item referred to has not been defined in the Data Division.

"Undefined procedure: %s"

The paragraph or section you are trying to ALTER, GO TO, or PERFORM has not been defined in the program.

"Undefined procedure: %s OF %s"

The paragraph or section you are trying to ALTER, GO TO, or PERFORM does not exist in the program within the qualifier you have specified for it.

"Unknown mode: %s"

As part of our compatibility with other dialects of COBOL, the ACUCOBOL-GT compiler allows the use of the RECORDING MODE clause. Only "F", "V", "S" and "U" modes are permitted.

"Unknown reserved word: %s"

"Unknown special name: %s"

The mnemonic-name in a Format 6 ACCEPT statement or in a Format 9 DISPLAY statement has not been defined in the Special-Names paragraph.

"Unknown switch: %s"

The switch named in the SPECIAL-NAMES paragraph must be one of SWITCH-1 through SWITCH-26, SWITCH 1 through SWITCH 26, or SWITCH "A" through SWITCH "Z".

"Unmatched ELSE"

The ELSE phrase must always be used in a one-to-one relationship with IF in an IF statement.

"Unmatched END-%s"

Several statements in COBOL, among them IF, SEARCH, PERFORM, and EVALUATE, can have their scope delimited by the END- (END-IF, END-SEARCH, END-PERFORM) phrase. Such END- phrases must exist in matched pairs with their companion verbs. The compiler has encountered such a scope delimiter, but found no matching verb preceding it.

"Unmatched NOT %s phrase"

Various NOT phrases may be used in the context of specific statements (READ ... NOT AT END, COMPUTE ... NOT ON SIZE ERROR). The compiler has encountered such a NOT phrase outside of its proper statement.

"Unsupported operation"

"USAGE conflict"

"USAGE must be DISPLAY"

"USE statement missing"

USING parameter <name> not aligned and may cause problems in the called subprogram

This is a warning message that can be generated if compiling with the "-Wa" option. This warns that a passed parameter is a group or is binary, and whose alignment is not an even multiple of the alignment specified by the "-Da#" option.

USING parameter <name> is not an 01-level item

This is a warning message that can be generated if compiling with the "-W1" option. The ANSI COBOL standard requires that parameters passed to subprograms be 01-level items. ACUCOBOL-GT does restrict them as such; however, there are valid reasons for restricting their use. See the *ACUCOBOL-GT User's Guide*, chapter 2 for details on this warning message.

V

"VALUE illegal on item > 64K"

"VALUE in EXTERNAL"

External data items may not have a VALUE phrase.

"VALUE in REDEFINES"

A Format 1 VALUE clause may not appear on a data item that is subordinate to a REDEFINES clause.

"Value must be 80 or 132"

The DISPLAY SCREEN SIZE statement must specify either an 80-column or a 132-column display.

"Value should be a name: %s"

You have compiled with the "-Fx" option. The object code was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This error occurs when the item to the right of an "=" should be a name and it isn't. For example, it would be an error to use a quoted string with the "name" directive: $XFD NAME="some text".

The "**%s**" in the message is the value found.

"Value should be numeric: %s"

You have compiled with the "-Fx" option. The object code was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This error occurs when the item to the right of an "=" should be numeric and it isn't. The "**%s**" in the message is the value found.

"Value should be a literal: %s"

You have compiled with the "-Fx" option. The object code was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This error occurs when the item to the right of an "=" should be a literal and it isn't. The "**%s**" in the message is the value found. A literal is either a quoted string or a numeric integer.

"VALUE size error: %s"

All literals used in a VALUE clause must have a value which falls within the range of allowed values for the item's PICTURE clause. Nonnumeric literals may not exceed the size of the item. Numeric items must have numeric literals. Alphabetic, alphanumeric, group, and edited items must have nonnumeric literals.

"VALUE specified for group"

When a VALUE clause is applied to a group item, no subordinate item may contain a VALUE clause.

"Value too large for context: %s"

The number you are using is too large. There are many cases in which 64 KB is the maximum size.

"VALUE too long: %s"

The maximum length for a floating point literal is 30.

"VALUE type error: %s"

All literals used in a VALUE clause must have a value within the range of allowed values for the item's PICTURE clause. Nonnumeric literals may not exceed the size of the item. Numeric items must have numeric literals. Alphabetic, alphanumeric, group, and edited items must have nonnumeric literals.

"Variable file name requires "File" directive"

You have compiled with the "-Fx" option. The object code was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This message occurs when the compiler cannot assign a name to the ".xfd" file because the ASSIGN phrase for the file names a variable file name. In this case, you must use a "file" directive to name the ".xfd" file.

"Verb expected, %s found"

W

"Warning: -Dcm ignored when using -Z%s"

The data storage option "-Dcm" is being ignored because you are generating object code for a runtime version that does not support that storage convention.

"Warning: -Dcn ignored when using -Z%s"

The NCR sign coding convention indicated by "-Dcn" requires a Version 2.4 or later runtime. The "-Dcn" flag has been ignored by the compiler.

"Warning: cannot generate native code from pre-5.0 object, '-Z%1' flag ignored"

"Warning: COLLATING SEQUENCE ignored for non-INDEXED files"

"Warning: native code not supported on current host, '-n' ignored"

"Warning: PADDING CHARACTER ignored for non-SEQUENTIAL files"

"Warning: Paragraph Name found in Area B."

"WHEN OTHER must be last"

No other WHEN phrase may follow WHEN OTHER in an EVALUATE statement.

"WHEN subject may not be reference modified"

SEARCH ALL does not allow the compared item to be reference modified.

"WHEN unexpected"

"WHEN variable xxx not found in record"

You have compiled with the "-Fx" option. The object code was generated, but a data dictionary could not be built, for the reason listed below. Remove the error condition and recompile to obtain a data dictionary.

This happens if you have a "when" directive that mentions a variable that doesn't exist in the record.

"WHEN, END-SEARCH or '.' required after NEXT SENTENCE"

"Writing %s code"

"Wrong number of parameters: %1 expected, %2 found"

E File Status Codes

Key Topics

E.1 Introduction

ACUCOBOL-GT conforms to five different standards regarding the values of file status codes. These codes are those used by RM/COBOL-85 (ANSI 85), RM/COBOL version 2 (ANSI 74), Data General ICOBOL, VAX COBOL, and IBM DOS/VS COBOL. By default, ACUCOBOL-GT uses the RM/COBOL-85 set. You can change the current set by changing the configuration variable **FILE_STATUS_CODES** (see also the *ACUCOBOL-GT User's Guide*, **Section 2.8.3, "File Status Codes"**).

The table in the next section describes the various file status codes returned by each condition. Some of the status values in the table have a second two-character code listed. This code distinguishes between different causes for the same FILE STATUS code. You can obtain this second code value by calling the ACUCOBOL-GT library routine **C$RERR** described in Appendix I. Where a second code is not listed, its value is "00".

For file systems that support READ PREVIOUS, wherever READ NEXT is mentioned, you may assume that READ PREVIOUS is also implied. An *end of file* for READ NEXT is analogous to a *beginning of file* for READ PREVIOUS.

E.2 Table of Codes

Regardless of which set of status codes is being used:

- Any code that starts with a "0" is considered successful.
- Any code that starts with a "1" is considered to be an "at end" condition.
- Any code that starts with a "2" is considered to be an "invalid key" condition.

85	74	Vax	DG	IBM	Condition
00	00	00	00	00	Operation successful.

85	74	Vax	DG	IBM	Condition
02	02	00	00	00	The current key of reference in the record just read is duplicated in the next record. (read next)
02	02	02	00	00	The operation added a duplicate key to the file where duplicates were allowed. (write, rewrite)
05	00	05	00	10	Optional file missing. If the open mode is I-O or EXTEND, then the file has been created. This is also returned by DELETE FILE if the file is not found. (open, delete file)
07	00	07	00	00	A CLOSE UNIT/REEL statement was executed for a file on a non-reel medium. The operation was successful.
0M	0M	0M	0M	00	The operation was successful, but some optional feature was not used. For example, if you opened a file that specified an alternate collating sequence, but the host file system did not support that feature, then the open would succeed, but it would return this status.
10	10	10	10	10	End of file. (read next)
14	00	14	00	00	A sequential READ statement was attempted for a relative file, and the number of digits in the relative record number is larger than the size of the relative key data item. (read next)
21	21	21	21	21	Primary key was written out of sequence, or the primary key on a rewrite does not match the last record read. This error occurs only for an indexed file open with the sequential access mode. (write, rewrite)
22	22	22	22	22	Duplicate key found but not allowed. (write, rewrite)
23	23	23	23	23	Record not found.
24	24	24	24	24	Disk full for relative or indexed file. (write)

85	74	Vax	DG	IBM	Condition
24, 01	00	24, 01	00	24	A sequential WRITE statement was executed for a relative file, and the number of digits in the relative record number was larger than the size of the relative key data item. (write)
30, xx	30, xx	30, xx	30, xx	30	Permanent error. This is any error not otherwise described. The secondary code value is set to the host system's status value that caused the error. See your operating system user manual for an explanation, and C$RERR in Appendix I.
34	34	34	34	34	Disk full for sequential file or sort file. (write, sort)
35	94, 20	35	91	93	File not found. (open, sort)
37, 01	95, 01	37, 01	91, 01	93	The file being opened is not on a mass-storage device which is required for the file type or the requested open mode. (open)
37, 02	95, 02	37, 02	91, 02	93, 02	Attempt to open a sequential file with fixed-length records as a Windows spool file.
37, 07	90, 07	39, 07	91, 07	93	User does not have appropriate access permissions to the file. (open)
37, 08	95, 08	37, 08	91, 08	93	Attempt to open a print file for INPUT. (open)
37, 09	95, 09	37, 09	91, 09	93	Attempt to open a sequential file for I/O and that file has automatic trailing space removal specified. (open)
37, 99	95, 99	37, 99	91, 99	93, 99	A Windows or Windows NT runtime that is not network-enabled tried to access a file on a remote machine.
38	93, 03	38	92	93	File previously closed with LOCK by this run unit. (open)

85	74	Vax	DG	IBM	Condition
39, xx	94, xx	39, xx	9A, xx	95	Existing file conflicts with the COBOL description of the file. (open) The secondary error code may have any of these values: **01** - mismatch found but exact cause unknown (this status is returned by the host file system) **02** - mismatch found in file's maximum record size **03** - mismatch found in file's minimum record size **04** - mismatch found in the number of keys in the file **05** - mismatch found in primary key description **06** - mismatch found in first alternate key description **07** - mismatch found in second alternate key description The list continues in this manner for each alternate key.
41	92	41	91	93	File is already open. (open)
42	91	42	92	92	File not open. (close)
42	91	94	91	92	File not open. (unlock)
43	90, 02	43	92	23	No current record defined for a sequential access mode file. (rewrite, delete)
44	97	44	92	21	Record size changed. The record being rewritten is a different size from the one existing in the file, and the file's organization does not allow this. (rewrite) This status code can also occur if the record is too large or too small according to the RECORD CONTAINS clause for the file. (write, rewrite)

85	74	Vax	DG	IBM	Condition
46	96	46	92	21	No current record. This usually occurs when the previous operation on the file was a START that failed, leaving the record pointer undefined. (read next)
47, 01	90, 01	47, 01	92, 01	13	File not open for input or I-O. (read, start)
47, 02	91, 02	47, 02	92, 02	13	File not open. (read, start)
48, 01	90, 01	48, 01	92, 01	13	A file that is defined to be access mode sequential is open for I-O, or the file is open for INPUT only. (write)
48, 02	91, 02	48, 02	92, 02	13	File not open. (write)
49, 01	90, 01	49, 01	92, 01	13	File not open for I-O. (rewrite, delete)
49, 02	91, 02	49, 02	92, 02	13	File not open. (rewrite, delete)
93	93	91	94	93	File locked by another user. (open)
94, 10	94, 10	97	97, 10	93	Too many files open by the current process. (open)
94, 62	94, 62	39, 62	92, 62	93	One of the LINKAGE values for this file is illegal or out of range. (open, write)
94, 63	94, 62	39, 62	92, 62	93	Key not specified (specifying a table whose size is zero) in a SORT or MERGE statement
98, xx	98, xx	30, xx	9B, xx	93	Indexed file corrupt. An internal error has been detected in the indexed file. The secondary status code contains the internal error number. The file should be reconstructed with the appropriate utility.
99	99	92	94	23	Record locked by another user.
9A	9A	9A	9A	23	Inadequate memory for operation. This most commonly occurs for the SORT verb, which requires at least 64K bytes of free space. (any)

85	74	Vax	DG	IBM	Condition
9B	9B	9B	9B	23	The requested operation is not supported by the host operating system. For example, a deferred file system initialization failed, or a READ PREVIOUS verb was executed and the host file system does not have the ability to process files in reverse order. (any) If you are using AcuXML, this error results when the program tries to open a file EXTEND or I-O. With AcuXML, programs are able to open files INPUT or OUTPUT only.
9C	9C	9C	9C	23	There are no entries left in one of the lock tables. The secondary error code indicates which table is full: **01** - operating system lock table **02** - internal global lock table (see the **MAX_LOCKS** configuration variable) **03** - internal per-file lock table (see the **LOCKS_PER_FILE** configuration variable)
9D, xx	9D, xx	9D, xx	9D, xx	92	This indicates an internal error defined by the host file system. The “xx” is the host system’s error value. This is similar to error “30”, except that “xx” is specific to the host file system instead of the host operating system. For example: **02** - In Acu4GL or AcuXML, 9D,02 indicates that an XFD file is corrupt. This could be the result of a parsing error. **03** - In Acu4GL or AcuXML, 9D,03 indicates that an XFD file is missing. This could be the result of a parsing error. **05** - In AcuXML, 9D,05 indicates that there was an XFD parsing error, so AcuXML was unable to read a record. Refer to the specific product documentation for more details on the host file system’s error codes.

85	74	Vax	DG	IBM	Condition
9E, xx	9E, xx	9E, xx	9E, xx	92	This indicates an error occurred in the transaction system. The exact nature of the error is shown by the contents of TRANSACTION-STATUS. See **section E.4, "Transaction Error Codes."**
9Z	9Z	9Z	9Z	92	This indicates that you are executing the program with a runtime that has a restriction on the number of records it can process. You have exceeded the record limit.

E.3 Vision Secondary Error Codes for Error 98s

Following is a brief description of the secondary error codes for error 98s for the Vision file system.

01 The file size listed in the file's header does not match the actual file size.

02 The header's next record pointer points to an area that is invalid.

03 Unique ID used to distinguish duplicate keys has already been used and cannot be used with a new key.

04 Missing tree terminator key.

05 An error was detected while performing a bulk read of a record.

06 The key being deleted from the tree was not found in the tree.

07 A child node was not found in its parent.

08 An I/O error occurred when the runtime was trying to read key information out of the file's header.

09 A pointer in a node points past the end of the file.

12 A node in the free node list was not marked as a free node.

13 A record in the deleted record list was not marked as a deleted record.

20 Non-zero key prefix on first key in node.

21 Key prefix larger than key size.

22 Key prefix or key size larger than maximum key size.

31 A record pointer in a Vision Version 3 file points to a record-chain value. In a Version 3 file, record pointers should always point to the start of a record, never to a record-chain value.

42 The unique record counter has been exhausted. Rebuild the file to correct the error.

68 A Vision 4 or 5 data segment is not found during an open.

69 A Vision 4 or 5 index segment is not found during an open.

81 Invalid data found in record header when a compressed record was read.

82 Invalid data found in record header when a non-compressed record was read.

83 When a record was read, an I/O error occurred or the record was too short.

84 When a record link was read, an I/O error occurred or the link was too small.

85 Record contains invalid record compression codes--the record would uncompress into a record that was larger than the maximum record size.

86 During a record write, a read of a record-chain value failed, probably due to an end-of-file condition.

87 Vision Version 4 or 5 detects that it is about to write a record to an area of a file that does not contain an appropriate record header. An appropriate record header indicates that a record currently does not exist at this address.

89 In Vision Version 4 or 5, on open, a data segment's internal revision number does not match the internal revision number stored in the header of the first data segment.

90 In Vision Version 4 or 5, on open, an index segment's internal revision number does not match the internal revision number stored in the header of the first data segment.

99 Vision Version 4 or 5 has tried to open the 65,537th data or index segment for this file. Vision can only support 65,536 data segments and 65,536 index segments per logical file.

E.4 Transaction Error Codes

A transaction management error is one that follows a START TRANSACTION, COMMIT, ROLLBACK or call to C$RECOVER, or one that occurs during some other file operation within a transaction (resulting in an error 9E). Error codes associated with these are stored in the TRANSACTION-STATUS register. This section lists and describes the primary and secondary transaction error codes.

E.4.1 Primary Error Codes

Following is a list of the primary error codes for the transaction management system.

01 This is returned from a ROLLBACK statement or call to **C$RECOVER** when an error occurs in an external routine. For a list of the secondary codes for this error, see **section E.4.2, "Secondary Error Codes for Error 01."**.

02 An attempt to open the log file failed because the maximum number of files per process would be exceeded. This is returned from a START TRANSACTION or call to C$RECOVER.

03 An attempt to open the log file failed because some element of the specified directory path is non-existent. This is returned from a START TRANSACTION statement or call to C$RECOVER.

04 An attempt to open the log file failed because the user has insufficient access privileges for the file. This is returned from a START TRANSACTION statement or call to C$RECOVER.

05 This indicates an operating system error that is not otherwise covered by one of the standard error conditions. You can determine the exact nature of this error by examining the value of the secondary error code.

06 This indicates that the log file is corrupted. The error is returned when the program encounters an unexpected end of file, or when an invalid transaction type code is found during recovery.

07 An attempt to open the log file failed because the file is locked (MS-DOS only). This is returned from a START TRANSACTION statement or a call to **C$RECOVER**.

08 This indicates that the system ran out of dynamic memory.

09 This indicates that a write failed because the disk is full.

10 This is returned from a START TRANSACTION statement or call to C$RECOVER when no log file was specified in the LOG-DIR configuration variable.

11 This is returned from a ROLLBACK or COMMIT statement when an unexpected end of file is reached while the rollback log file is being read.

12 A START TRANSACTION, ROLLBACK or COMMIT failed because the last transaction in the log file is incomplete.

13 This error is returned in the TRANSACTION-STATUS register from a WRITE, REWRITE, CLOSE, or DELETE if the file was not opened *within a transaction.* Note that, if the FILE-CONTROL paragraph for the file contains the WITH ROLLBACK phrase, all OPENs are automatically performed within a transaction.

14 This is a file-system specific error that is not one of the standard errors, and not an error returned by the operating system. The secondary and tertiary error codes indicate the exact meaning, which is file-system dependent.

16 This error is returned when the runtime is executing a START TRANSACTION while another transaction is already active.

99 This warning indicates that the requested transaction operation is not supported by a host file system. The transaction operation is still attempted for other file systems.

E.4.2 Secondary Error Codes for Error 01

The following is a list of the secondary error codes for transaction error 01.

	Secondary Error	**Corresponding file-status error**
01	operating system error (see tertiary code for system-specific error code)	30
02	illegal parameter	39,01
03	attempt to open more files than system allows	94,10
04	open mode does not allow operation	48,01 or 49,01
05	requested record is locked	99
06	index file is corrupt	98,xx
07	duplicate key where duplicates not allowed	22
08	requested record not found	23
10	disk became full while adding a new record	24
11	file locked against requested open mode	93
12	record size mismatch during rewrite	44

	Secondary Error	**Corresponding file-status error**
14	out of dynamic memory	9A
15	requested file does not exist	35
16	inadequate access permissions to file	37,07
17	requested operation not supported	9B
18	out of lock-table entries	9C
19	file-system specific error	9D

E.5 IBM DOS/VS Error Codes

IBM DOS/VS COBOL has a form of the USE statement in the DECLARATIVES section that is not normally recognized by ACUCOBOL-GT:

```
USE AFTER STANDARD ERROR PROCEDURE ON file-name GIVING
      data-name-1 [data-name-2]
```

This form is accepted by ACUCOBOL-GT when the "-Cv" option is in effect.

When an error handler introduced by this statement is invoked, the runtime puts special error codes into the eight-byte data item *data-name-1*. For more information and the list of codes, see Chapter 5, "IBM DOS/VS COBOL Conversions," in the *Transitioning Your COBOL Applications to ACUCOBOL-GT* book.

F Intrinsic Functions

Key Topics

F.1 Introduction

Intrinsic functions are subprograms that are built into the ACUCOBOL-GT library. They save time by simplifying common tasks that your COBOL programs might need to perform. For example, intrinsic functions can perform statistical calculations, convert strings from upper to lower case, compute annuities, derive values for trigonometric functions such as sine and cosine, and perform general utility tasks such as determining the compile date of the current object file.

Intrinsic functions are sometimes called built-in or library functions.

To access an intrinsic function, you include it inside a COBOL statement (typically a MOVE or COMPUTE statement). Here's an example of a statement that uses the "min" intrinsic function:

```
move function min(3,8,9,7) to my-minimum.
```

This COBOL statement can be translated into: move the result derived from performing the "min" function on the literals "3, 8, 9, and 7" to the variable "my-minimum."

Note the presence of the required word "function," followed by the name of the function ("min") and then its parameters.

Each intrinsic function is evaluated to a data value. This value is stored in a temporary storage area that you cannot access directly in your program. The only way to get the derived value of an intrinsic function is to provide the name of a data item into which the resulting value should be placed. In the example shown above, the variable "my-minimum" receives the derived value of the "min" function.

In the example above, the parameters passed to the "min" function are literals. It is also permissible to pass data items, as shown here:

```
compute my-sine = function sin(angle-a).
```

Note: When the return value of a function is a double, the precision of the return value is limited to that supported by the underlying hardware.

However, if your COBOL program is compiled for 31-digit support ("-Dd31"), numeric functions are computed using special floating point arithmetic that is accurate to approximately 33 digits, regardless of the floating-point representation on the host machine.

The functions that return a double include: ABS, ABSOLUTE-VALUE, ACOS, ANNUITY, ASIN, ATAN, COS, LOG, LOG10, MEAN, MEDIAN, MIDRANGE, NUMVAL, NUMVAL-C, PRESENT-VALUE, RANDOM, REM, SIN, SQRT, STANDARD-DEVIATION, TAN, and VARIANCE.

F.2 Function Definitions and Returned Values

The definition of a function identifies the following:

- For alphanumeric functions, the size of the returned value
- For numeric and integer functions, the sign of the returned value and whether the function is an integer
- For some other cases, the value returned

Data item functions are elementary data items and return alphanumeric, numeric, or integer values. Data item functions are treated as elementary data items and cannot be receiving operands. Types of data item functions are as follows:

- Alphanumeric functions--these are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage of DISPLAY.
- Numeric functions--these are of the class and category numeric. A numeric function is always considered to have an operational sign.
- A numeric function may be used only in an arithmetic expression.

- A numeric function may not be referenced where an integer operand is required, even though a particular reference may yield an integer value.
- Integer functions--these are of the class and category numeric. A numeric function is always considered to have an operational sign.
- An integer function may be used only in an arithmetic expression.
- An integer function can be referenced where an integer operand is required and where a signed operand is allowed.

F.2.1 Function Definitions

The table below summarizes the functions that are now available.

The Arguments column defines the type and number of arguments as follows:

A alphabetic

I integer

N numeric

X alphanumeric

The Type column defines the type of the function as follows:

I integer

N numeric

Z alphanumeric

Function-name	Arguments	Type	Value returned
ABSOLUTE-VALUE (or ABS)	N1	N	Absolute value of the argument passed
ACOS	N1	N	Arccosine of N1
ANNUITY	N1, N2	N	Ratio of annuity paid for I2 N2 periods at interest rate of N1 to initial investment of one
ASIN	N1	N	Arcsine of N1
ATAN	N1	N	Arctangent of N1
CHAR	I1	X	Character in position I1 of program collating sequence
COS	N1	N	Cosine of N1
CURRENT-DATE	None	X	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	I1	I	Standard date equivalent (YYYYMMDD) of integer date
DAY-OF-INTEGER	I1	I	Julian date equivalent (YYYYDDD) of integer date
FACTORIAL	I1	I	Factorial of I1
INTEGER	N1	I	The greatest integer not greater than N1
INTEGER-OF-DATE	I1	I	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	I1	I	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	N1	I	Integer part of N1

Function-name	Arguments	Type	Value returned
LENGTH	A1 or N1 or X1	I	Length of argument
LOG	N1	N	Natural logarithm of N1
LOG10	N1	N	Logarithm to base 10 of N1
LOWER-CASE	A1 or X1	X	All letters in the argument are set to lowercase
MAX	A1... or I1... or N1... or X1...	Depends on arguments.*	Value of maximum argument
MEAN	N1...	N	Arithmetic mean of arguments
MEDIAN	N1...	N	Median of arguments
MIDRANGE	N1...	N	Mean of minimum and maximum arguments
MIN	Al... or I1... or N1... or X1...	Depends on arguments*	Value of minimum argument
MOD	I1, I2	I	I1 modulo I2
NUMVAL	X1	N	Numeric value of simple numeric string
NUMVAL-C	X1, X2	N	Numeric value of numeric string with optional commas and currency sign
ORD	A1 or X1	I	Ordinal position of the argument in collating sequence
ORD-MAX	A1... or N1... or X1...	I	Ordinal position of maximum argument
ORD-MIN	A1... or N1... or X1	I	Ordinal position of minimum argument

Function-name	Arguments	Type	Value returned
PRESENT-VALUE	N1, N2...	N	Present value of a series of future period-end amounts, N2n at a discount rate of N1
RANDOM	I1	N	Random number
RANGE	I1... or N1...	Depends on arguments	Value of maximum argument minus value of minimum argument
REM	N1, N2	N	Remainder of N1/N2
REVERSE	A1 or X1	X	Reverse order of the characters of the argument
SIN	N1	N	Sine of N1
SQRT	N1	N	Square root of N1
STANDARD-DEVIATION	N1...	N	Standard deviation of arguments
SUM	I1... or N1...	Depends on arguments	Sum of arguments
TAN	N1	N	Tangent of N1
UPPER-CASE	A1 or X1	X	All letters in the argument are set to uppercase
VARIANCE	N1...	N	Variance of argument
WHEN-COMPILED	None	X	Date and time program was compiled

*A function that has only alphabetic arguments is type alphanumeric.

F.3 ABSOLUTE-VALUE (ABS) Function

The ABSOLUTE-VALUE (or ABS) function returns a single numeric value which is the absolute value of the argument passed.

Usage

```
FUNCTION ABSOLUTE-VALUE (argument-1)
```

or

```
FUNCTION ABS (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Value

The returned value is a single numeric value which is the absolute value of *argument-1*.

F.4 ACOS Function

The ACOS function returns a numeric value in radians that approximates the arccosine of argument-1. The type of this function is numeric.

Usage

```
FUNCTION ACOS (argument-1)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be greater than or equal to "-1" and less than or equal to "+1".

Returned Value

The returned value is the approximation of the arccosine of *argument-1* and is greater than or equal to zero and less than or equal to pi. This function will produce results accurate to only about 17 digits, even when *argument-1* contains more than 18 digits (for example, if you have compiled your program for 31-digit support.)

F.5 ANNUITY Function

The ANNUITY function (annuity immediate) returns a numeric value representing the amount of each payment in a series of equal periodic payments whose total value is 1.0, where *argument-1* is the interest rate per period, *argument-2* is the number of periods (usually 12), and each payment is applied at the end of its period. The type of this function is numeric.

a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by *argument-1* and is applied at the end of the period before the payment. The type of this function is numeric.

Usage

```
FUNCTION ANNUITY (argument-1 argument-2)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be greater than or equal to zero.
3. *Argument-2* must be a positive integer.

Returned Values

1. When the value of *argument-1* is zero, the value of the function is the approximation of:

   ```
   1 / argument-2
   ```

2. When the value of *argument-1* is not zero, the value of the function is the approximation of:

   ```
   argument-1 / (1 - (1 + argument-1) ** (- argument-2))
   ```

Note: This function will produce results accurate to only about 17 digits, even when *argument-1* contains more than 18 digits (for example, if you have compiled your program for 31-digit support.)

F.6 ASIN Function

The ASIN function returns a numeric value in radians that approximates the arcsine of *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION ASIN (argument-1)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be greater than or equal to "-1" and less than or equal to "+1".

Note: This function will produce results accurate to only about 17 digits, even when *argument-1* contains more than 18 digits (for example, if you have compiled your program for 31-digit support.)

Returned Value

The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to "-pi/2" and less than or equal to "+pi/2".

F.7 ATAN Function

The ATAN function returns a numeric value in radians that approximates the arctangent of *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION ATAN (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Value

The returned value is the approximation of the arctangent of *argument-1* and is greater than “-pi/2” and less than “+pi/2”.

F.8 CHAR Function

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position that corresponds to the value of *argument-1*. The type of this function is alphanumeric.

Usage

```
FUNCTION CHAR (argument-1)
```

Parameters

1. *Argument-1* must be an integer.
2. The value of *argument-1* must be greater than zero and less than or equal to the number of positions in the collating sequence.

Returned Values

1. If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.
2. If the current program collating sequence was not specified by an ALPHABET clause, the value returned will be the character in the ASCII character set occupying the ordinal position of the argument.

F.9 COS Function

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians that is specified by argument-1. The type of this function is numeric.

Usage

```
FUNCTION COS (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Value

The returned value is the approximation of the cosine of *argument-1* and is greater than or equal to "-1" and less than or equal to "+1".

F.10 CURRENT-DATE Function

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated. The type of this function is alphanumeric.

Usage

```
FUNCTION CURRENT-DATE
```

Returned Values

1. The character positions returned, numbered from left to right, are described in the table below.

Character Position	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range of 00 through 23.

Character Position	Contents
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past a second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	The character '0'. This is reserved for future use.
18-19	The characters '00'. This is reserved for future use.
20-21	The characters '00'. This is reserved for future use

2. If the system does not have the facility to provide fractional parts of a second, the value 00 is returned in character positions 15 and 16.

3. If the system does not have the facility to provide the local time differential factor, the value 00000 is returned in character positions 17 through 21.

4. Currently, we do not support the information contained in positions 17-through 21. These fields will contain 0.

5. The returned value can be reference modified. For example:

```
MOVE FUNCTION CURRENT-DATE (1:4) TO YEARDATE.
```

F.11 DATE-OF-INTEGER Function

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD). The type of this function is integer.

Usage

```
FUNCTION DATE-OF-INTEGER (argument-1)
```

Parameter

Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600 in the Gregorian calendar.

Returned Values

1. The returned value represents the ISO standard date equivalent to the integer specified in *argument-1*.
2. The returned value is in the form YYYYMMDD, where YYYY represents a year in the Gregorian calendar; MM represents the month of that year; and DD represents the day of that month.

F.12 DAY-OF-INTEGER Function

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD). This type of function is integer.

Usage

```
FUNCTION DAY-OF-INTEGER (argument-1)
```

Parameter

Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Returned Values

1. The returned value represents the Julian equivalent of the integer specified in *argument-1*.
2. The returned value is an integer of the form YYYYDDD, where YYYY represents a year in the Gregorian calendar, and DDD represents the day of that year.

F.13 FACTORIAL Function

The FACTORIAL function returns an integer that is the factorial of *argument-1*. The type of this function is integer.

Usage

```
FUNCTION FACTORIAL (argument-1)
```

Parameter

Argument-1 must be an integer greater than or equal to zero.

Returned Values

1. If the value of *argument-1* is zero, the value "1" is returned.
2. If the value of *argument-1* is positive, its factorial is returned.

F.14 INTEGER Function

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

Usage

```
FUNCTION INTEGER (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned values

1. When standard arithmetic is specified, *argument-1* is not rounded.
2. The returned value is the greatest integer less than or equal to the value of *argument-1*. For example, if the value of *argument-1* is " -1.5", "-2" is returned. If the value of *argument-1* is "+1.5", "+1" is returned.

F.15 INTEGER-OF-DATE Function

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form. The type of this function is integer.

Usage

```
FUNCTION INTEGER-OF-DATE (argument-1)
```

Parameter

Argument-1 must be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
- MM represents a month, and must be a positive integer less than 13.
- DD represents a day, and must be a positive integer less than 32 provided that it is valid for the specified month and year combination.

Returned Value

The returned value is an integer that is the number of days the date represented by *argument-1* succeeds December 31, 1600 in the Gregorian calendar.

F.16 INTEGER-OF-DAY Function

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form. The type of this function is integer.

Usage

```
FUNCTION INTEGER-OF-DAY (argument-1)
```

Parameter

Argument-1 must be an integer of the form YYYYDDD, whose value is obtained from the calculation (YYYY * 1000) + DDD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
- DDD represents the day of the year. It must be a positive integer less than 367 provided that it is valid for the year specified.

Returned Value

The returned value is an integer that is the number of days the date represented by *argument-1* succeeds December 31, 1600 in the Gregorian calendar.

F.17 INTEGER-PART Function

The INTEGER-PART function returns an integer that is the integer portion of *argument-1*. The type of this function is integer.

Usage

```
FUNCTION INTEGER-PART (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Values

1. If the value of *argument-1* is zero, the returned value is zero.
2. If the value of *argument-1* is positive, the returned value is the greatest integer less than or equal to the value of *argument-1*. For example, if the value of *argument-1* is “+1.5”, then “+1” is returned.

3. If the value of *argument-1* is negative, the returned value is the least integer greater than or equal to the value of *argument-1*. For example, if the value of *argument-1* is "-1.5", then "-1" is returned.

F.18 LENGTH Function

The LENGTH function returns an integer equal to the length of the argument in character positions. This type of function is integer.

Usage

```
FUNCTION LENGTH (argument-1)
```

Parameters

1. *Argument-1* may be a non-numeric literal or a data item of any class or category.
2. If *argument-1* (or any data item subordinate to *argument-1*) is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

Returned Values

1. If *argument-1* is a non-numeric literal or an elementary data item, or if *argument-1* is a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.
2. If *argument-1* is a group data item containing a variable occurrence data item, the returned value is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause dealing with the data item as a sending data item.
3. The returned value includes implicit FILLER characters, if any.

Note: This function is similar in functionality to the Format 8 Set SET statement: “SET result-item TO SIZE OF data-item”.

F.19 LOG Function

The LOG function returns a numeric value that approximates the logarithm to the base e (natural log) of *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION LOG (argument-1)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be greater than zero.

Returned Value

The returned value is the approximation of the logarithm to the base e of argument-1.

F.20 LOG10 Function

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION LOG10 (argument-1)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be greater than zero.

Returned Value

Returned value is the approximation of the logarithm to the base 10 of *argument-1*.

F.21 LOWER-CASE Function

The LOWER-CASE function returns a character string that is the same length as *argument-1* with each uppercase letter replaced by the corresponding lowercase letter. The type of this function is alphanumeric.

Usage

```
FUNCTION LOWER-CASE (argument-1)
```

Parameter

Argument-1 must be class alphabetic or alphanumeric, and must be at least one character in length.

Returned Values

1. The same character string as *argument-1* is returned, except that each uppercase letter is replaced by the corresponding lowercase letter.
2. The character string returned has the same length as *argument-1*.
3. If the computer character set does not include lowercase letters, no changes take place in the character string.
4. This function only translates characters with a numeric value of 0-128. Anything above that (such as é, with a value of 130) must be mapped to its associated upper- or lower-case character using the configuration variable UPPER-LOWER-MAP.

Note: This function is similar to the library routine C$TOLOWER except that the original data is not modified, and the entire string is converted.

5. The returned value can be reference modified. For example:

```
MOVE FUNCTION LOWER-CASE(FILE-NAME)(1:4) TO TMP-STRING.
```

F.22 MAX Function

The MAX function returns the content of the *argument-1* that contains the maximum value. The type of this function depends upon the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric	Numeric (some arguments may be integer)

Usage

```
FUNCTION MAX ({argument-1} ... )
```

Parameters

If more than one *argument-1* is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the content of the *argument-1* having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions.

2. If more than one *argument-1* has the same greatest value, the content of the *argument-1* returned is the leftmost *argument-1* having that value.

3. If the type of the function is alphanumeric, the size of the returned value is the same as the size to the selected *argument-1*.

F.23 MEAN Function

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments. The type of this function is numeric.

Usage

```
FUNCTION MEAN ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

1. The returned value is the arithmetic mean of the *argument-1* series.

2. The returned value is defined as the sum of the *argument-1* series divided by the number of occurrences referenced by *argument-1*.

F.24 MEDIAN Function

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order. The type of this function is numeric.

Usage

```
FUNCTION MEDIAN ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

1. The returned value is the content of the *argument-1* having the middle value in the list formed by arranging all the *argument-1* values in sorted order.
2. If the number of occurrences referenced by *argument-1* is odd, the returned value is such that at least half of the occurrences referenced by *argument-1* are greater than or equal to the returned value, and at least half are less than or equal. If the number of occurrences referenced by *argument-1* is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.
3. The comparisons used to arrange the *argument-1* values in sorted order are made according to the rules for simple conditions.

F.25 MIDRANGE Function

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument. The type of this function is numeric.

Usage

```
FUNCTION MIDRANGE ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

The returned value is the arithmetic mean of the greatest *argument-1* value and the least *argument-1* value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions.

F.26 MIN Function

The MIN function returns the content of the *argument-1* value that contains the minimum value. The type of this function depends upon the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

Usage

```
FUNCTION MIN ({argument-1} ... )
```

Parameters

If more than one *argument-1* is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the content of the *argument-1* having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions.
2. If more than one *argument-1* has the same least value, the content of the *argument-1* returned is the leftmost argument-1 having that value.
3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected *argument-1*.

F.27 MOD Function

The MOD function returns an integer value that is *argument-1* modulo *argument-2*. The type of this function is integer.

Usage

```
FUNCTION MOD (argument-1 argument-2)
```

Parameters

1. *Argument-1* and *argument-2* must be integers.
2. The value of *argument-2* must not be zero.

Returned Values

1. The returned value is *argument-1* modulo *argument-2*. The returned value is defined as:

   argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))

2. The following illustrates the expected results for some values of *argument-1* and *argument-2*:

argument-1	**argument-2**	**Return**
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

F.28 NUMVAL Function

The NUMVAL function returns the numeric value represented by the character string specified by *argument-1*. Leading and trailing spaces are ignored. The type of this function is numeric.

Usage

```
FUNCTION NUMVAL (argument-1)
```

Parameters

1. *Argument-1* must be a non-numeric literal or alphanumeric data item whose content has one of the following two formats:

```
[space] [+] [space] {digit [ . [digit]]} [space]
        [-]         {. digit           }
```

or

```
[space] {digit [ . [digit]]} [space] [+ ] [space]
        {. digit           }         [- ]
                                     [CR]
                                     [DB]
```

where *space* is a string of zero or more spaces, and *digit* is a string of one to 18 digits.

2. The total number of digits in *argument-1* must not exceed 18. If your program has been compiled for 31-digit support ("-Dd31"), *argument-1* must not exceed 31.

3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in *argument-1* rather than a decimal point.

Returned Values

1. The returned value is the numeric value represented by *argument-1*.

2. The number of digits returned is 18. If your program has been compiled for 31-digit support ("-Dd31"), up to 31digits may be returned.

F.29 NUMVAL-C Function

The NUMVAL-C function returns the numeric value represented by the character string specified by *argument-1*. Any optional currency sign specified by *argument-2* and any optional commas preceding the decimal point are ignored. The type of this function is numeric.

Usage

```
FUNCTION NUMVAL-C (argument-1 [argument-2])
```

Parameters

1. *Argument-1* must be a non-numeric literal or alphanumeric data item whose content has one of the following formats:

```
[space] [+] [space] [cs] [space] {digit [, digit] ... [. [digit]]} [space]
        [-]                      {. digit                        }
```

or

```
[space] [cs] [space] {digit [, digit] ... [. [digit]]} [space] [+ ] [space]
                     {. digit                        }         [- ]
                                                               [CR]
                                                               [DB]
```

where *space* is a string of zero or more spaces, *cs* is the string of one or more characters specified by *argument-2*, and *digit* is a string of one or more digits.

2. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in *argument--1* are reversed.

3. The total number of digits in *argument-1* must not exceed 18. If your program has been compiled for 31-digit support (“-Dd31”), *argument-1* must not exceed 31.

4. *Argument-2*, if specified, must be a non-numeric value represented by *argument--1*.

5. If *argument-2* is not specified, the character used for cs is the currency symbol specified for the program.

Returned Values

1. The returned value is the numeric value represented by *argument-1*.

2. The number of digits returned is 18. If your program has been compiled for 31-digit support (“-Dd31”), up to 31digits may be returned.

F.30 ORD Function

The ORD function returns an integer value that is the ordinal position of argument-1 in the collating sequence for the program. The lowest ordinal position is "1". The type of this function is integer.

Usage

```
FUNCTION ORD (argument-1)
```

Parameter

Argument-1 must be one character in length, and must be class alphabetic or alphanumeric.

Returned Value

The returned value is the ordinal position of *argument-1* in the collating sequence for the program.

F.31 ORD-MAX Function

The ORD-MAX function returns a value that is the ordinal number of the *argument-1* that contains the maximum value. The type of this function is integer.

Usage

```
FUNCTION ORD-MAX ({argument-1} ... )
```

Parameters

If more than one *argument-1* is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the ordinal number that corresponds to the position of the *argument-1* having the greatest value in the *argument-1* series.

2. The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions.

3. If more than one *argument-1* has the same greatest value, the number returned corresponds to the position of the leftmost *argument-1* having that value.

F.32 ORD-MIN Function

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value. The type of this function is integer.

Usage

```
FUNCTION ORD-MIN ({argument-1} ... )
```

Parameters

If more than one *argument-1* is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the ordinal number that corresponds to the position of the *argument-1* having the least value in the *argument-1* series.

2. The comparisons used to determine the least valued *argument-1* are made according to the rules for simple conditions.

3. If more than one *argument-1* has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

F.33 PRESENT-VALUE Function

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by *argument-2* at a discount rate specified by *argument-1*.

Usage

```
FUNCTION PRESENT-VALUE (argument-1 {argument-2} ... )
```

Parameters

1. *Argument-1* and *argument-2* must be of the class numeric.
2. The value of *argument-1* must be greater than -1.

Returned Value

The returned value is an approximation of the summation of a series of calculations with each term in the following form:

argument-2 / (1 + argument-1) ** n

There is one term for each occurrence of *argument-2*. The exponent, *n*, is incremented increased from one by in increments of one for each term in the series.

Example

```
COMPUTE RSULT = FUNCTION PRESENT-VALUE (DISCOUNT-RATE, 2000).
```

Note: In this example, DISCOUNT-RATE and RSULT are numeric data items. If DISCOUNT–RATE has the value “0.08”, the value returned and stored in RSULT is approximately “1851.85”.

F.34 RANDOM Function

The RANDOM function returns a numeric value that is a pseudo-random number (one of a sequence of numbers generated by an algorithm so as to have an even distribution over a range of values and minimal correlation between successive values) from a rectangular distribution. The type of this function is numeric.

Usage

```
FUNCTION RANDOM [(argument-1)]
```

Parameters

1. If *argument-1* is specified, it must be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.
2. If a subsequent reference specifies *argument-1*, a new sequence of pseudo-random numbers is started.
3. If the first reference to this function in the run unit does not specify *argument-1*, a seed value will be provided by the runtime.
4. In each case, subsequent references without specifying *argument-1* return the next number in the current sequence.

Returned Values

1. The returned value is greater than or equal to zero and less than one.
2. For a given seed value on a given implementation, the sequence of pseudo-random numbers will always be the same.

Example

```
   77  random_num  pic s9(4)v99.
...
   move function random() to random_num.
```

F.35 RANGE Function

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument. The type of this function depends on the argument types as follows:

Argument Type	Function Type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

Usage

```
FUNCTION RANGE ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Value

The returned value is equal to the greatest value of argument-1 minus the least value of *argument-1*. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions.

F.36 REM Function

The REM function returns a numeric value that is the remainder of *argument-1* divided by *argument-2*. The type of this function is numeric.

Usage

```
FUNCTION REM (argument-1 argument-2)
```

Parameters

1. *Argument-1* and *argument-2* must be class numeric.
2. The value of *argument-2* must not be zero.

Returned Value

The returned value is the remainder of *argument-1* / *argument-2*. It is defined as the expression:

Argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1 / argument-2))

F.37 REVERSE Function

The REVERSE function returns a character string of exactly the same length as *argument-1* and whose characters are exactly the same as those of *argument-1*, except that they are in reverse order. The type of this function is alphanumeric.

Usage

```
FUNCTION REVERSE (argument-1)
```

Parameter

Argument-1 must be class alphabetic or alphanumeric, and must be at least one character in length.

Returned Value

If *argument-1* is a character string of length *n*, the returned value is a character string of length *n* such that for $1 \leq j \leq n$, the character in position j of the returned value is the character from position n-j+1 of *argument-1*.

F.38 SIN Function

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION SIN (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Value

The returned value is the approximation of the sine of *argument-1* and is greater than or equal to "-1" and less than or equal to "+1".

F.39 SQRT Function

The SQRT function returns a numeric value that approximates the square root of *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION SQRT (argument-1)
```

Parameters

1. *Argument-1* must be class numeric.
2. The value of *argument-1* must be zero or positive.

Returned Value

The returned value is the absolute value of the approximation of the square root of *argument-1*.

F.40 STANDARD-DEVIATION Function

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments. The type of this function is numeric.

Usage

```
FUNCTION STANDARD-DEVIATION ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the standard deviation of the *argument-1* series.
2. The returned value is calculated as follows:
   a. The difference between each *argument-1* value and the arithmetic mean of the *argument-1* series is calculated and squared.
   b. The values obtained are then added together. This quantity is divided by the number of values in the argument-1 series.
   c. The square root of the quotient obtained is then calculated. The returned value is the absolute value of the square root.
3. If the *argument-1* series consists of only one value, or if the *argument-1* series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

F.41 SUM Function

The SUM function returns a value that is the sum of the arguments. The type of this function depends upon the argument type as follows:

Argument Type	**Function Type**
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

Usage

```
FUNCTION SUM ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

1. The returned value is the sum of the arguments.
2. If the *argument-1* series are all integers, the value returned is an integer.
3. If the *argument-1* series are not all integers, a numeric value is returned.

F.42 TAN Function

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by *argument-1*. The type of this function is numeric.

Usage

```
FUNCTION TAN (argument-1)
```

Parameter

Argument-1 must be class numeric.

Returned Value

The returned value is the approximation of the tangent of *argument-1*.

F.43 UPPER-CASE Function

The UPPER-CASE function returns a character string that is the same length as *argument-1* with each lowercase letter replaced by the corresponding uppercase letter. The type of this function is alphanumeric.

Usage

```
FUNCTION UPPER-CASE (argument-1)
```

Parameter

Argument-1 must be class numeric or alphanumeric and must be at least one character in length.

Returned Values

1. The same character string as *argument-1* is returned, except that each lowercase letter is replaced by the corresponding uppercase letter.
2. The character string returned has the same length as *argument-1*.
3. This function only translates characters with a numeric value of 0-128. Anything above that (such as é, with a value of 130) must be mapped to its associated upper- or lower-case character using the configuration variable UPPER-LOWER-MAP.

   Note: This function is similar to the library routine C$TOUPPER except that the original data is not modified, and the entire string is converted.

   ---

4. The returned value can be reference modified. For example:

   ```
   MOVE FUNCTION UPPER-CASE(FILE-NAME)(1:4) TO TMP-STRING.
   ```

F.44 VARIANCE Function

The VARIANCE function returns a numeric value that approximates the variance of its arguments. The type of this function is numeric.

Usage

```
FUNCTION VARIANCE ({argument-1} ... )
```

Parameters

Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the variance of the *argument-1* series.
2. The returned value is defined as the square of the standard deviation of the *argument-1* series.
3. If the *argument-1* series consists of only one value, or if the *argument-1* series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

F.45 WHEN-COMPILED Function

The WHEN-COMPILED function returns the date and time the program was compiled as provided by the system on which the program was compiled. The type of this function is alphanumeric.

Usage

```
FUNCTION WHEN-COMPILED
```

Returned Values

1. The character positions returned, numbered from left to right, are described in the table below.

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.

Character Positions	Contents
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range of 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past a second, in the range 00 through 99. The value 00 will be returned for all systems
17	The character '0'. This field is reserved for future implementation.
18-19	The characters '00'. This field is reserved for future implementation.
20-21	The characters '00'. This field is reserved for future implementation.

2. The returned value is the date and time of compilation of the source program that contains this function. If the program is a contained program, the returned value is the compilation date and time associated with the separately compiled program in which it is contained.

Note: The returned value must denote the same time as the compilation date and time if provided in the listing of the source program and in the generated object code for the source program, although their representations and precisions may differ.

---

3. The returned value can be reference modified. For example:

```
MOVE FUNCTION WHEN-COMPILED (1:4) TO YEARDATE.
```

G Reserved for Future Use

As a convenience to long-time ACUCOBOL-GT programmers and users, we have retained this empty appendix so that Appendix H, “Runtime Configuration Entries”, and Appendix I, “ACUCOBOL-GT Library Routines” can continue to be located in their historic positions.

H Configuration Variables

Key Topics

H.1 Introduction

Many aspects of the runtime system can be controlled through *runtime configuration variables*. This mechanism provides a great deal of flexibility, because these variables can be modified by each **runcbl** site as well as directly by an ACUCOBOL-GT program.

H.1.1 Variable Syntax

Configuration variables are maintained in a *runtime configuration file*. This standard text file can be modified by the host system's text editor. Each entry in the runtime configuration file consists of a single line. All entries start with a keyword, followed by one or more spaces or tabs, and then one or more values. The limit for each configuration value entry is 4095 characters.

Some examples of runtime configuration variables are:

```
AUTO_PROMPT   0
BELL   1
COMPRESS_FACTOR   70
CURSOR_TYPE   3
MENU_ITEM   Edit=Delete   200
SCROLL   on
```

For all runtime configuration variables, "=" placed between the keyword and the first value is optional, and is interchangeable with a space.

For the following configuration variables, a colon (:) may be used instead of an equals sign (=) in the value portion of the entry:

COLOR-TABLE	COLOR-MAP
FILE-CONDITION	KEYBOARD
KEYSTROKE	SCREEN
MENU-ITEM	MOUSE
HOT-KEY	

In the above cases, allowing a colon instead of an equals sign in the value portion of the entry makes it possible to specify these values in environment variables. This accommodates systems that do not allow an equals sign in the environment variable.

For some runtime configuration variables, the words “on”, “true”, and “yes” are synonyms for “1”, and the words “off”, “false”, and “no” are synonyms for “0”. The entry for each variable in this appendix indicates when these synonyms are allowed.

In the keyword, all lower-case characters are treated as upper-case and all hyphens are treated as underscores. Keywords longer than 60 characters are truncated to 60 characters.

H.1.2 Variable Usage

The configuration file is optional, as are all of its contents. For this reason, no errors in the configuration file are ever reported. The “-l” **runcbl** option can help debug configuration file problems.

In the descriptions of some runtime configuration variables, you will find comments about behavior under the Windows environment; unless otherwise noted, these comments apply to all 32-bit versions of the Windows operating system.

Runtime configuration variables may be placed in either the runtime configuration file or the machine’s environment. When they are placed in the runtime configuration file, upper- and lower-case names are equivalent, as are hyphens and underscores. When placed in the machine’s environment, the keywords must be all upper case and must use underscores instead of hyphens. For more details about the configuration process, see the *ACUCOBOL-GT User’s Guide*, **section 2.8, “Runtime Configuration.”**

All configuration variables that have a default value are used by and affect the runtime in the same way that they would if they were in the configuration file. That is to say, a configuration variable that has a default value is treated as if it appears in the configuration file set to the default value.

The values of many runtime configuration variables may be changed at runtime with the SET ENVIRONMENT verb. The syntax is:

```
SET ENVIRONMENT env-name TO env-value
```

Env-name may specify either the literal name of the variable or a data-item whose value is the name of the variable. If you specify the actual name of the variable, such as CODE_CASE, then you must enclose the name in quotes. *Env-value* is the value to which *env-name* will be set. If it is a numeric data item, then it is treated as if it were redefined as an alphanumeric data item.

Most configuration variables can be read with the ACCEPT FROM ENVIRONMENT statement. If the variable to be read is numeric, then the receiving field must be defined either as a numeric field or as an alphanumeric field of five or more characters. If it is defined as alphanumeric and is longer than five characters, then the value that is read from the environment will occupy the leftmost five characters of the field and the remainder will be space-filled.

H.1.3 Configuration filename Resolution

runcbl uses the following rules to decide what the configuration file is called:

1. If the "-c" runtime option is used, the configuration file is the one named by that option; otherwise,
2. If the operating system environment variable "A_CONFIG" is defined, its value is the name of the configuration file; otherwise,
3. The configuration file is named according to the host operating system. This depends on the operating system used by the machine, as outlined in the following table.

System	Configuration File
Windows	\etc\cblconfi
UNIX/Linux	/etc/cblconfig
MPE/iX	/etc/cblconfig
VMS	SYS$LIBRARY:A_CONFIG.DAT

Caution: Do not give a data file a name that is the same as a configuration variable name. Doing so can cause problems if you map the data filename through a configuration entry. For example, if you have a data file named "CURRENCY", the runtime may confuse the data file with the configuration variable of the same name, inadvertently changing the default currency character.

H.1.4 Nested configuration files

It is possible to use multiple configuration files by nesting one inside another. Within the configuration file, you can specify another file to process with the following syntax:

```
!COPY filename
```

No name expansion is done to *filename* (for example FILE_PREFIX is not applied) so you must specify a file that the runtime can find. You can include remote name syntax if you are using AcuServer® or AcuConnect®. Otherwise, the file must be an absolute path or a path relative to the current directory.

For example, if you have some configuration variables in a global place such as "/etc/cblconfi", then individual users can execute the runtime using this configuration file instead of the usual one. The settings in the usual configuration file take effect also, because their settings are copied in with !COPY:

```
#Get all the standard variables
!copy /etc/cblconfi

#Now set personal settings
COMPRESS_FILES 1
```

H.2 Configuration Variables

This section contains an alphabetical list of the runtime configuration file variables. Many of these variables are also described in other parts of the documentation set.

3D_LINES

This variable has meaning only on graphical systems such as Windows. Set this variable to “1” (on, true, yes) to cause the runtime to display lines and boxes with 3-D shading. This makes the lines appear to be inscribed into the surface of the screen. The variable is especially helpful in giving a 3-D look to a program originally designed on a character system. Only black lines on a non-black background are shown with shading. Other lines are displayed normally.

The set of colors available to ACUCOBOL-GT significantly impacts how effective the shading will be. Normally, the shading is most effective when the background is low-intensity white. The other low-intensity colors are next best.

The shading is only marginally effective with a high-intensity background. For this reason, the 3D_LINES setting is not used when a high-intensity background is drawn. Note that, by default, ACUCOBOL-GT shows background colors in high-intensity, so you will need to use at least one other configuration variable to arrange for a low-intensity background color. For example, the **BACKGROUND_INTENSITY** variable could be set to “1” to force a low-intensity background.

You may freely change the way lines are displayed in COBOL by using the SET ENVIRONMENT verb to set 3D-LINES prior to displaying a line or a box.

- Setting it to “1” (on, true, yes) gives you the 3-D effect.
- Setting it to “0” (off, false, no) gives you normal lines.

The runtime remembers which lines are drawn with 3-D, so you don’t need to keep track of this yourself. Note, however, that if you attach a 3-D line to a non-3-D line, the intersection will use the 3D-LINES setting currently in effect.

The default value is “0”.

4GL_COLUMN_CASE

When set to “unchanged”, this variable causes the runtime to leave the case and hyphen usage of the field names found in XFDs unchanged. XFDs are used with the Acu4GL interface, AcuXDBC, or AcuXML. They are also required for international character mapping with AcuServer and they provide useful information to the **alfred** record editor. By default, the runtime converts all field names to lower case and all hyphens to underscores.

For AcuXML, the case and hyphen usage of the XFD must match the XML file exactly, and 4GL_COLUMN_CASE should be set to “unchanged”. For Acu4GL, however, you should be aware that most databases do not accept hyphens in column names. If you set this variable to “unchanged” to protect case, you may need to modify the XFD by hand to replace hyphens with underscores.

7_BIT

When this configuration variable is set to “1” (on, true, yes), ACUCOBOL-GT supports 7-bit communications instead of 8-bit. This variable is designed specifically for machines that use 7-bit communications with parity enabled. When 7_BIT is set to the default of “0” (off, false, no), 8-bit communications are used.

A_CHECKDIV

This variable allows you to specify an alternate runtime response to a divide by zero condition when the statement does not include a SIZE ERROR clause.

In COBOL, a division by zero produces a size error condition. The SIZE ERROR clause allows the programmer to specify actions to take when this condition occurs. If there is no SIZE ERROR clause, by default in ACUCOBOL-GT the results are undefined. You can use the A_CHECKDIV configuration variable to specify alternate handling.

A_CHECKDIV can be set to:

NONE	or:	"0"	The default setting. This setting retains the default behavior of the runtime: the results are undefined.
ABEND	or:	"1", STOP, ABORT	This setting causes the runtime to catch the divide by zero condition and exit with the error message: "Attempt to divide by zero".
MOVE_ZERO	or:	"2", ZERO_RESULT , MOVE_ZEROS	This setting causes the runtime to move zeroes to the destination item(s) and continue.

A_DEBUG

This variable is available for applications such as online transaction servers that call ACUCOBOL-GT through the C API (see Chapter 6 of *A Guide to Interoperating with ACUCOBOL-GT*). The default value is "0". With the default setting, the debugger launches when the debug_method flag in the C interface is set to "1".

Set this variable to "1" to turn on the ACUCOBOL-GT debugger in an xterm window the first time you call the C interface. The debugger shuts down when the program that caused it to launch shuts down.

A_DISPLAY

This variable is available for applications such as online transaction servers that call COBOL through the C API. The value of A_DISPLAY overrides the value of the DISPLAY environment variable. Set A_DISPLAY to the value of your X server host name or IP address in the runtime configuration file (or /etc/cblconfig). For example:

```
A_DISPLAY myvpn123.myhostname.com:0
```

A_EXTFH_FUNC

The value of this variable is an EXTFH function name needed for the EXTFH interface. If you are using a library that contains an EXTFH function name other than “cics_xfh”, “cobol_extfh”, or “EXTFH”, you also need to set one or more of these variables to specify the function name:

A_EXTFH_FUNC	Specifies a function to be used by all file types (indexed, relative, and sequential).
A_EXTFH_IDX_FUNC	Specifies a function name to be used by indexed file types.
A_EXTFH_REL_FUNC	Specifies a function name to be used by relative file types.
A_EXTFH_SEQ_FUNC	Specifies a function name to be used by sequential file types.

For example, to specify a function name to use for all file types:

```
A_EXTFH_FUNC=myExtfh
```

Or, to specify a different function for indexed, relative, and sequential files:

```
A_EXTFH_IDX_FUNC=myIdxExtfh
A_EXTFH_SEQ_FUNC=mySeqExtfh
A_EXTFH_REL_FUNC=myRelExtfh
```

If the library is a DLL, you can specify both the name of the DLL and the calling convention to use. Any calling convention specified this way overrides the **DLL_CONVENTION** variable setting. For information about specifying DLLs and calling conventions, see section 3.3.2, “Loading DLLs with Configuration Variables,” in *A Guide to Interoperating with ACUCOBOL-GT.*

A_EXTFH_LIB

The value of this variable is an EXTFH shared library or DLL file name. You can use this variable to dynamically load an EXTFH library without relinking the ACUCOBOL-GT runtime. For example:

```
A_EXTFH_LIB libraryname.so
```

You can also use the following variables to specify library names for indexed, relative, and sequential files. The ACUCOBOL-GT runtime uses A_EXTFH_LIB as the default EXTFH library for all three file types. If one or more of these three variables is also set, the runtime uses its value instead of A_EXTFH_LIB for the corresponding file type.

A_EXTFH_IDX_LIB	Specifies the EXTFH library to use for indexed files.
A_EXTFH_REL_LIB	Specifies the EXTFH library to use for relative files.
A_EXTFH_SEQ_LIB	Specifies the EXTFH library to use for sequential files.

You can specify these variables in the runtime configuration file or as operating system environment variables.

If the library is a DLL, you can specify both the name of the DLL and the calling convention to use. Any calling convention specified this way overrides the **DLL_CONVENTION** variable setting. For information about specifying DLLs and calling conventions, see section 3.3.2, “Loading DLLs with Configuration Variables,” in *A Guide to Interoperating with ACUCOBOL-GT.*

See section 11.6, “Working With an EXTFH Interface,” in *A Guide to Interoperating with ACUCOBOL-GT*, for information on specifying EXTFH library and function names to use with the EXTFH interface.

A_EXTFH_SIMPLE_OPEN_OUTPUT

This variable is only used in UniKix environments, and the UniKix application automatically sets this variable to “1” (TRUE). When set to “1” (TRUE), an OPEN OUTPUT statement will cause the EXTFH functions to bypass the “make” process, and will open the file as OUTPUT. When set to “0” (FALSE), or not set at all, the EXTFH functions will execute the “make” process, and will open the file as EXTEND.

A_EXTFH_VARIABLE_IDX, A_EXTFH_VARIABLE_REL, A_EXTFH_VARIABLE_SEQ

These variables indicate whether the filesystem you are accessing with the the EXTFH interface can or cannot handle variable length files. Setting this variable to the default of "1" (on, true, yes) causes the EXTFH interface to pass the minimum and maximum record lengths to the file system for variable length files as defined in the COBOL program. Setting this variable to "0" (off, false, no) causes the EXTFH interface to ignore the variable record length defined in the COBOL program and instead pass a record length equal to the maximum record length.

You can specify the variable separately for indexed, relative, and sequential files. For example:

```
A_EXTFH_VARIABLE_IDX=0
A_EXTFH_VARIABLE_REL=0
A_EXTFH_VARIABLE_SEQ=1
```

When the file system does not process variable length files, set these configuration variables to "0" and the EXTFH interface treats variable length records as fixed lengths.

If the file system does process variable length files, set the configuration variables to "1" (or do not set them at all).

A_JAVA_CHARSET

This variable specifies the character set that the runtime should use when mapping Java strings or PIC X data items containing characters outside of the ISO-8859-1 range. The default setting is "IS0-8859-1". If you have data outside the IS0-8859-1 range (for example, an umlaut or Euro symbol), specify a different character set that contains those characters.

Be aware of a common misconception that ISO-8859-1 is equivalent to Windows-1252. This is mostly true, but there are characters in the range 0x80 – 0x9F that differ. Windows-1252 uses these numbers for letters and punctuation, while the ISO-8859-1 uses these for control codes.

A_JAVA_GC_COUNT

A_JAVA_GC_COUNT is a 32-bit value that determines how often the runtime calls the JVM garbage collector. The JVM garbage collector will run at unknown times, in order to deallocate memory which is no longer being used. Setting this to a non-zero value allows you to be a little more intentional about running the garbage collector. The value is the number of times C$JAVA is called before the runtime calls the JVM garbage collector. The default value is 9883, so every 9883 calls to C$JAVA will explicitly call the JVM garbage collector. (For more info on the JVM garbage collector, see your JVM documentation.)

A_JAVA_TRACE_FILENAME

A_JAVA_TRACE_FILENAME is the name of the file where the trace information is sent. This filename can include all of the format specifiers that the runtime error file can include. If this file can't be opened for writing (for any reason), no trace information is collected.

A_JAVA_TRACE_VALUE

To track calls to the JVM made on behalf of the COBOL program, you can set one of the following three configuration variables: A_JAVA_TRACE_VALUE, A_JAVA_TRACE_FILENAME, and A_JAVA_GC_COUNT.

A_JAVA_TRACE_VALUE is a 32-bit value that determines the types of calls to trace. Add any of the following values together to create a single value to set.

1 - Show calls that return simple types (boolean, byte, character, short, integer, long, float, double).

2 - Show method calls that return simple types.

4 - Show string calls that return string references (that must be released).

8 - Show string calls that return simple types.

16 - Show calls that return references to a Java object (that must be released).

32 - Show method calls that return references to a Java object (that must be released).

64 - Show calls that return references to a Java array or array elements (that must be released).

128 - Show calls that return other array information.

256 - Show calls to the exception routines (some of which must be released).

512 - Show calls to get IDs (Method Identifiers or Field Identifiers).

1024 - Show calls to field functions that return references to a Java object (that must be released).

2048 - Show calls to field functions that return simple types.

4096 - Show other types of calls that return references to a Java object (that must be released).

8192 - Show calls to release a reference to a Java object.

16384 - Show other calls to the Java runtime.

Note for there to be no memory leaks, any call that returns a reference to a Java object (that must be released) needs to be paired with a call to release that reference. If the COBOL program gets that reference, it is responsible for releasing the reference. If the runtime gets the reference for internal purposes, the runtime is responsible for releasing the reference.

For example, setting A_JAVA_TRACE_VALUE to 13684 shows all calls to the JVM that obtain or release a reference to a Java object. Setting A_JAVA_TRACE_VALUE to -1 is equivalent to setting it to 32767 (which is the sum of all the above values), and has the added benefit of tracing new options that may be added in the future. However, for finding memory leaks, this may be too much information.

A_LICENSE_RETRIES

This variable affects UNIX networks with multiple-user licenses for the runtime. When set to a positive, non-zero value, this entry causes the runtime to retry ("value" times) any failed attempt to register with the network license manager, acushare. The configuration variable **A_RETRY_DELAY** specifies how many seconds the runtime will wait between retries.

The default value is "0" (no retries).

A_OPERATING_SYSTEM

As of Version 5.0, the runtime no longer differentiates between "UNIX-V" and "UNIX-4" in the OPERATING-SYSTEM field of the SYSTEM-INFORMATION data item. Instead, the value "UNIX" is used for all UNIX platforms. If you have an existing program that depends on one of the older values, set A_OPERATING_SYSTEM to a value of "UNIX-V" or "UNIX-4". Then, when an ACCEPT FROM SYSTEM-INFO statement is executed, this value overrides the value returned by the function. The default value is empty.

A_REMOVE_EMPTY_ERROR_FILE

Use this variable to prevent the accumulation of 0 byte files when using format specifiers such as "%p**"** (to include the process id) in the error file name. When this variable is set to "1" (on, true, yes), the runtime deletes its error file if the runtime has never written to that file. Note that on some operating systems, if your error file is shared by multiple processes (i.e., the file name does not include the process id or some other unique session information), setting A_REMOVE_EMPTY_ERROR_FILE to "1" may cause error messages to be lost. For example, on UNIX if the error file is empty when one runtime exits, that runtime would delete the file. The file will remain deleted even if another runtime process subsequently writes a message to it. The default value for this variable is "0" (off, false, no).

A_RETRY_DELAY

This variable affects UNIX networks with multiple-user licenses. If **A_LICENSE_RETRIES** is set to a positive integer value, then the value of A_RETRY_DELAY determines how many seconds the runtime will wait between repeated attempts to register itself with the network license manager, **acushare**.

The default value is “10”.

A_SEQ_DEFAULT_BLOCK_SIZE

This configuration variable determines the size of the buffer to use when accessing a sequential file whose definition has no BLOCK CONTAINS clause. When set, A_SEQ_DEFAULT_BLOCK_SIZE specifies the size of the buffer in characters, rounded up to the nearest power of 2 that is greater than or equal to that value. The default value is “0”, which sets the block size to one record. Note that this variable does not apply to print files or to files with names that start with a hyphen followed by “D” or “P”.

You can set A_SEQ_DEFAULT_BLOCK_SIZE in the environment to allow the “**vutil** -load” command to buffer the input file according to the variable’s value. The maximum buffer size is 1 GB. If this variable is not set, the default buffer block size is 4096 bytes. If it is set to “0”, “**vutil** -load” performs record-based I/O on a sequential file.

A_SYSLOG_HOSTNAME

This variable applies only on Windows and works in conjunction with the **A_SYSLOG_ON_RUNTIME_ERROR** configuration variable. Set A_SYS_HOSTNAME to the server name or IP address on which the event log is located. Do not include any slashes with the server name. The default value for this variable is empty. Then set A_SYSLOG_ON_RUNTIME_ERROR to “1” (on, true, yes). Shutdown messages will be sent to the event log on the local machine.

A_SYSLOG_ON_RUNTIME_ERROR

When this variable is set to "1" (on, true, yes), on a fatal error, the runtime will send its shutdown error message to the UNIX syslog daemon, console, or Windows event log. The runtime uses the same logic as the C$SYSLOG routine. (See **C$SYSLOG** in Appendix I for more information). The error message also includes the name of the runtime error file so that the administrator can view it for more information. The default value for this variable is "0" (off, false, no).

ACCEPT_AUTO

This configuration variable applies only when running in HP COBOL compatibility mode (with the "-Cp" compiler option). The ACCEPT_AUTO configuration variable causes the runtime to treat all Format 1 ACCEPT statements as if the AUTO phrase is used, whether or not AUTO appears in the statement. Set this variable to "1" (on, true, yes) to enable this behavior. The default value is "0" (off, false, no).

ACCEPT_TIMEOUT

This variable causes all ACCEPT statements to time out just as if there was a BEFORE TIME phrase present in the ACCEPT statement. The value assigned to ACCEPT_TIMEOUT is the timeout period, in seconds. This timeout value is applied to every ACCEPT statement that can have a BEFORE TIME phrase specified for it. If a particular ACCEPT statement has a BEFORE TIME phrase explicitly coded for it, that phrase takes precedence and ACCEPT_TIMEOUT does not apply to that statement. The default value of ACCEPT_TIMEOUT is "0", which indicates no timeout value.

ACTIVE_BORDER_COLOR

This variable is used on character-based hosts to specify the color and video attributes of the characters used to form the border (box) around the active floating window. ACTIVE_BORDER_COLOR can be set to a variety of

numeric values that express combinations of color and video attributes. See the documentation for the COLOR phrase in the "Common Screen Options" section of the *ACUCOBOL-GT Reference Manual* (**Section 6.4.9**).

If ACTIVE_BORDER_COLOR is set to "0", the active window's border is drawn with the colors and video attributes specified in the COBOL program when the window is initially created. The default value is "0".

ACU_DUMP, ACU_DUMP_FILE, ACU_DUMP_WIDTH, ACU_DUMP_TABLE_LIMIT

These configuration variables are used to enable and configure the Abend Diagnostic Report (ADR) facility. For a complete description of the ADR, see **Section 3.1.9**, in Book 1, *ACUCOBOL-GT User's Guide*.

ACU_DUMP

This variable enables the Abend Diagnostic Report. The default value is "0" (off, false, no). Set ACU_DUMP to "1" (on, true, yes) to turn on the ADR.

ACU_DUMP_FILE

This variable specifies the name of the report file. It allows the following special parameters:

- If the file name starts with a plus sign ("+"), the report is appended to the specified file. By default, a new report overwrites the specified file.
- If the name contains the string "%p", when the report is generated that string is replaced with the process ID (PID) of the runtime from which the report originates.
- If the name contains the string "%d", that string is replaced with the current date in the form YYYYMMDD where YYYY is the year, MM month and DD day.
- If the name contains the string "%t", that string is replaced with the current time in the form HHMMSSTTT where HH is the hour, MM minute, SS second and TTT milliseconds.

- If the name contains the string “%u”, that string is replaced with the username.
- If the name contains the string “%h”, that string is replaced with the hostname.

The default value for ACU_DUMP_FILE is “acudump.%p”.

ACU_DUMP_WIDTH

This variable controls the width of the report and has a default value of 80 characters. The minimum allowed value is 79 and the maximum is 2048. Note that because the report uses dynamically computed columns for its hexadecimal data, making the report very wide can reduce readability by introducing excessive white space.

ACU_DUMP_TABLE_LIMIT

This variable limits how many elements of each table item to list. The default value is 1000. Note that if you increase this value substantially, and if you have tables that allow for large numbers of elements, you may get very large reports.

In the following example, ACU_DUMP_TABLE_LIMIT is set to 5:

```
01 MY-TABLE-R                    = (group)
05 TABLE-ENTRY(1)                =     1          h20202020 31
05 TABLE-ENTRY(2)                =     2          h20202020 32
05 TABLE-ENTRY(3)                =     3          h20202020 33
05 TABLE-ENTRY(4)                =     4          h20202020 34
05 TABLE-ENTRY(5)                =     5          h20202020 35
Remaining table items suppressed due to ACU-DUMP-TABLE-LIMIT setting
```

ACU_USER_DIR

The ACU_USER_DIR configuration variable specifies the default location of a user debugger settings file. In the past, the ACUCOBOL variable has been used for this purpose. When set, ACU_USER_DIR specifies the directory for the user’s debugger settings (“.adb”) file. The default value is “NULL”, which causes the runtime to use the ACUCOBOL variable.

ACUCOBOL

This variable holds the full path to the ACUCOBOL-GT installation directory. For example, if the runtime is installed in "C:\Program Files\Acucorp\Acucbl8xx\AcuGT\bin", you would set this configuration variable to:

```
ACUCOBOL C:\Program Files\Acucorp\Acucbl8xx\AcuGT
```

This variable is used to locate extensions to the runtime.

AGS_BLOCK_SLEEP_TIME

This variable is used to specify the amount of wait time before attempting to retry writing data to a socket. It can improve file I/O peformance times for large files (32K or larger). The value of this variable by default is 10 milliseconds. The default value should provide performance at par with pre 7.3 versions.

This variable only has impact on UNIX/Linux.

AGS_MAX_SEND_SIZE

This variable allows you to control the size of a basic socket packet exchanged between ***extend*** applications that use sockets to communicate. The default value is 16000. In the vast majority of cases, the default value provides excellent results. However, when performance problems are traced to packet size, you can change the size with AGS_MAX_SEND_SIZE. The value of this variable is checked every time that data is sent to the socket. When a program changes the value, the new value is applied the next time that data is sent to the socket.

AGS_RECEIVE_BUFFER_SIZE

This variable determines the size of the low-level receive buffer for a socket connection. For the value to have an affect, it must be set before any sockets have been created. The default value is 16384. The default value should be sufficient for most cases. The receive-buffer-size is passed directly to a call to setsockopt.

Note: The value of this variable is sent to a lower-level socket layer not controlled by ACUCOBOL-GT. It may not have any noticeable effect. Changes in this value are not seen in response to a "U" debugger command listing the memory usage of the runtime.

AGS_SEND_BUFFER_SIZE

This variable determines the size of the low-level send buffer for a socket connection. For the value to have an affect, it must be set before any sockets have been created. The default value is 16384. The default value should be sufficient for most cases. The send-buffer-size is passed directly to a call to setsockopt.

Note: The value of this variable is sent to a lower-level socket layer not controlled by ACUCOBOL-GT. It may not have any noticeable effect. Changes in this value are not seen in response to a "U" debugger command listing the memory usage of the runtime.

AGS_SOCKET_COMPRESS

This variable determines the type of data compression performed at the internal socket layer. AGS_SOCKET_COMPRESS must be set before any socket communication is done, and cannot be changed via SET ENVIRONMENT. This variable has three possible values:

NONE	This is the default setting. When AGS_SOCKET_COMPRESS is set to this value, no compression is performed.
ZLIB	When AGS_SOCKET_COMPRESS is set to this value, socket data is compressed using the same algorithm as the **gzip** compression utility.
RUNLENGTH	When AGS_SOCKET_COMPRESS is set to this value, simple compression is done, based on counting repeated bytes of data.

RUNLENGTH compression tends to be very fast, while ZLIB compression tends to compress the data more, but is slower as a result.

Windows supports ZLIB compression, but not all UNIX machines do. For those machines that do not, RUNLENGTH compression will be used whether this variable is set to ZLIB or RUNLENGTH. When the compression algorithm is being negotiated with a server, the method that both machines support will be used.

AGS_SOCKET_ENCRYPT

To turn on encryption at the internal socket-layer, set the configuration variable AGS_SOCKET_ENCRYPT to “1” (on, true, yes). It must be set before any socket communication is performed, and cannot be changed via a SET ENVIRONMENT statement.

Note: If the variables AS_CLIENT_ENCRYPT and/or THIN_CLIENT_ENCRYPT are set to “1”, AGS_SOCKET_ENCRYPT is also set to “1” automatically.

AGS_TCP_NODELAY

This variable determines whether the Nagle algorithm is used when sending socket buffer messages. This algorithm automatically delays sending small socket packets for a short period of time in order to increase network efficiency by sending them in a batch. Setting this variable to the default of “1” (on, true, yes) causes socket packets to be sent immediately (not using the algorithm), while setting this variable to “0” (off, false, no) causes socket packets to be delayed (using the algorithm). The TCP-NODELAY socket option is used as follows:

```
setsockopt(s, IPPROTO_TCP, TCP_NODELAY, &tcp_nodelay, sizeof(int));
```

The value of this variable is sent to a lower-level socket layer not controlled by ACUCOBOL-GT. It may not have any noticeable effect.

alfred Configuration variables

As of Version 8.0, the Indexed File Record Editor (alfred) is provided as a sample program and is located in the “sample” folder under “AcuGT”. You can download detailed information on using and configuring alfred in PDF format from the **Support > Examples & Utilities >Acucorp Samples > Acucorp Technical Articles and Tips** section of the Micro Focus website (www.microfocus.com).

ALLOW_FS_OVERRIDE

This variable enables you to determine if the actual EXTFH return status will be returned, or if the return status should be translated by the runtime. The default setting is “True” or “1” and will cause the actual EXTFH return status to be returned to the user. Setting this variable to “False” or “0” will cause the EXTFH return status to be translated by the runtime

ANSI_OUTPUT_IN_DEBUG

This variable prevents a COBOL program that uses ANSI-style DISPLAY statements from interfering with the runtime debugger window. This variable accepts two possible values: “CANVAS” or “TERMINAL”.

When set to “CANVAS” (the default setting) the runtime constructs a default canvas on which to place the ANSI output. This prevents the ANSI output from interfering with the debugger window. Note that if your COBOL program sends escape sequences to the terminal, this mode will cause those escape sequences to not have the intended result.

When set to “TERMINAL”, the runtime will send ANSI output to the terminal, possibly interfering with the view of the debugger window. This is how the runtime behaved before the implementation of this new feature.

Note that this configuration variable must be set before the runtime initializes the terminal manager, which means you cannot set this variable from a COBOL program.

APPLY_CODE_PATH

When set to “1” (on, true, yes), this variable causes the **CODE_PREFIX** variable to be applied to object files with full path names (those beginning with a “/” (forward slash). Otherwise, CODE_PREFIX is not applied to files with full path names. For example, if your application specifies the file:

```
/accounting/objects/payroll
```

and your CODE_PREFIX variable is set to:

```
CODE_PREFIX  /master_obj
```

and APPLY_CODE_PATH is set to “on”, the runtime will look for your file in:

```
/master_obj/accounting/objects/payroll
```

The default value of APPLY_CODE_PATH is “0” (off, false, no).

APPLY_FILE_PATH

When set to “1” (on, true, yes), this variable causes the **FILE_PREFIX** variable to be applied to data files with full path names (those beginning with “/”, forward slash). Otherwise, FILE_PREFIX is not applied to files with full path names. For example, if your application specifies the file:

```
/accounting/data/ind.dat
```

and your FILE_PREFIX variable is set to:

```
FILE_PREFIX  /master_data
```

and APPLY_FILE_PATH is set to “on”, the runtime will look for your file in:

```
/master_data/accounting/data/ind.dat
```

The default value of APPLY_FILE_PATH is “0” (off, false, no).

AUTO_DECIMAL

When set to “1” (on, true, yes), this variable checks the data item descriptions of numeric entry fields with a decimal point for the number of digits that must be filled to the right of the decimal point. When all the digits after the decimal point are entered, the field will terminate if the AUTO_TERMINATE phrase is specified. The number of digits to the right of the decimal point can vary, depending on how many are indicated in the picture of each numeric entry field. You must specify AUTO_TERMINATE phrase for this feature to work.

The exception to this is when an entry field has an AUTO_DECIMAL property specified, in which case, the coded value will be used.

The default value of this variable is “0” (off, false, no).

AUTO_PROMPT

When set to “1” (on, true, yes), this variable causes all ACCEPT statements without a PROMPT clause to be treated as if they had a PROMPT SPACES clause. This causes the screen to be erased at the field position prior to the data’s being entered. This variable is provided for compatibility with ACUCOBOL-85 Version 1.3 and earlier, which behaved this way. The default setting is “0” (off, false, no).

AXML_CREATE_SCHEMA

This variable is designed for use with AcuXML for instances when you want to include a schema or schema name with your XML output. In order for this variable to have an effect, **AXML_CREATE_STYLE** must be set to “schema” and **AXML_SCHEMA_NAME** must name the schema file. Once these conditions are met, this variable tells AcuXML whether to create a schema file with XML output, or simply include the name of a schema file in the output.

By default, when AXML_CREATE_STYLE is set to schema, AcuXML creates a schema file for all XML output. Because only one schema is typically required, you should set AXML_CREATE_SCHEMA to “FALSE” after the first time a schema is created. Then, only the name of the schema file will be included in the output XML file. Similarly, if you already have a schema file and don’t want AcuXML to overwrite it, set this variable to “FALSE.”

AXML_CREATE_STYLE

This variable is designed for use with AcuXML. Use it to define the type of XML output that ACUCOBOL-GT should generate when it creates XML files. It can be set to “DTD”, “SCHEMA” or “NONE”. Set this variable to “NONE” if you want the resulting XML file to be raw XML. Set it to “DTD” if you want the output to include a Document Type Definition of the elements in the document. Often, the party with whom you trade data may require that your XML document include a DTD.

Set this variable to “SCHEMA” if you want ACUCOBOL-GT to create a schema to describe the XML documents that it writes. Schemas provide the highest level of detail about the contents of the associated XML document, and are typically required for development purposes. If you set this variable to “SCHEMA”, you must use the **AXML_SCHEMA_NAME** variable to name the schema file.

Please note that creating a schema for a file that was run through the **xml2fd** utility with a schema won’t result in an identical schema. In addition, note that setting this variable to “schema” causes a schema to be created for every XML output file by default. Once the first schema is created, you should set **AXML_CREATE_SCHEMA** to “FALSE” to prevent schemas from being created on subsequent XML outputs.

AXML_ENCODING

This variable is designed for use with AcuXML. Use it when you want to specify a character encoding method for the XML files that ACUCOBOL-GT creates. By default, the XML output generated by ACUCOBOL-GT is mapped to the UTF-8 encoding system (compatible with the US-ASCII character set). If you want to use a different encoding system, for instance a European encoding system that includes the British pound character (£), change this variable to reflect the new system name. For example:

```
AXML_ENCODING IS0-8859-1
```

This variable causes encoding information to be added to the header of XML files created by ACUCOBOL-GT. With the configuration file entry shown above, the following header would be included:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

This header causes the ISO-8859-1 Latin encoding system to be applied to the data file as desired.

AcuXML supports the following encoding systems:

- UTF-8, default [8-bit Unicode Transformation Format, backwards compatible with US-ASCII]

- US-ASCII
- UTF-16 [16-bit Unicode Transformation Format]
- ISO-8859-1 [Latin 1, European encoding]

AXML_EXACT_TABLE_MATCH

This variable affects the behavior of AcuXML. By default, all tables in an FD must match data in the XML file with respect to the values of the indices. Therefore, AXML_EXACT_TABLE_MATCH is set to "1" (on, true, yes) by default. To disable this requirement, set AXML_EXACT_TABLE_MATCH to "0" (off, false, no).

AXML_IGNORE_EMPTY_DATA

Set this variable to "TRUE" to omit empty and zero-filled data from AcuXML's output file. In this case, AcuXML will not write tags for alphabetic data items that are all blank or numeric data items that are "0". When you set this variable from your COBOL program, it affects any records written via AcuXML from that point on. Note that setting this variable to "TRUE" could cause AcuXML to generate parts of an XML file that are not consistent with any DTD or schema associated with the file. As a result, use this variable with care.

The default value of "FALSE" causes AcuXML to generate tags for all data items in the file. If your records are mostly empty, this may be overkill.

AXML_SCHEMA_DOC

This variable is designed for use with AcuXML. Use it when you want to add a documentation element to the schema that ACUCOBOL-GT creates when it writes an XML file (such as whenever a sequential file is OPEN OUTPUT).

If you do not require specific documentation in the schema file, or if you did not request that schemas be created for XML output, you can omit this variable.

If this variable is set, its value is included in the documentation element of the resulting schema. For example, if you set this variable as follows:

```
AXML_SCHEMA_DOC This is the documentation to be
included in the file...
```

The schema will include the following data:

```
<xs:annotation>
  <xs:documentation>
    This is the documentation to be included in the file.
    Created by AcuXML version 6.0.0 on 2002/05/16
  </xs:documentation>
</xs:annotation>
```

Note: For information on working with XML data, see section 11.2 in *A Guide to Interoperating with ACUCOBOL-GT.*

AXML_SCHEMA_NAME

This variable is designed for use with AcuXML. Use it to define the name of the schema file that ACUCOBOL-GT writes, if any, when it creates an XML file. If this variable is not set, or if it is set to a file name that cannot be created (for whatever reason), a schema is not created.

Note: To tell ACUCOBOL-GT to create a schema, use the **AXML_CREATE_STYLE** and **AXML_CREATE_SCHEMA** variables.

AXML_SCHEMA_NAMESPACE_DATA

This variable is designed for use with AcuXML for instances when you want to include a schema or schema name with your XML output and you want precise control over the schema namespace string shown in the output. The default value of this variable is:

```
xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
    xs:noNamespaceSchemaLocation=\"%s\"
```

By default, when ACUCOBOL-GT writes XML output (and a schema has been requested), it substitutes the "%s" in this variable with the name of the schema file specified with the **AXML_SCHEMA_NAME** configuration variable. For instance, if AXML_SCHEMA_NAME is set to "myschema", ACUCOBOL-GT will include the following line in the XML output:

```
xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
    xs:noNamespaceSchemaLocation=\"myschema.xsd\"
```

If you need something different than "myschema.xsd" written in the namespace output, add this variable to your configuration file and alter the namespace value in the quotes to meet your requirements.

Note: If you want to include a single "%" character in the namespace, add a second percent sign "%%" to the definition of this variable.

In general, the value of this variable is used in the standard C library printf() function as the first argument, and all printf() rules apply.

AXML_STYLESHEET_HREF and AXML_STYLESHEET_TYPE

These variables are designed for use with AcuXML. Use them when you want to associate an XML style sheet with the XML documents that ACUCOBOL-GT creates. When you set these variables to a non-blank value, ACUCOBOL-GT includes an XML-stylesheet comment in the beginning of the resulting XML files. For instance, the following entry:

```
AXML_STYLESHEET_TYPE text/css
```

causes the following comment to be added to the beginning of the XML file:

```
<?xml-stylesheet type="text/css"?>
```

If you set both of these variables, as in the following example,

```
AXML_STYLESHEET_TYPE text/css
AXML_STYLESHEET_HREF mystyle.css
```

then a comment like the following is added to the file:

```
<?xml-stylesheet type="text/css" href="mystyle.css"?>
```

If you do not require specific stylesheet data in the XML file, you can omit these variables.

These variables may be toggled on and off during program execution. If you have set these variables and want to generate a xml file without a xml style sheet association, you can do this by adding these statements before opening the xml file:

```
SET ENVIRONMENT "AXML_STYLESHEET_TYPE" TO ""
SET ENVIRONMENT "AXML_STYLESHEET_HREF" TO ""
```

Note: Setting these variables to NULL or SPACE will NOT toggle the variable off.

BACKGROUND_INTENSITY

This variable is used to choose a background intensity. Use one of these values:

0	The runtime uses the default intensity, which is based on your hardware and operating environment. Under Windows, the default background intensity is high-intensity. The default value is “0”.
1	The runtime uses low-intensity.
2	The runtime uses high-intensity.

There are two important exceptions:

- The runtime always assigns low-intensity to the background if the background color is black. Using high-intensity would cause the background to be dark gray, which tends to make the screen look muddy.
- Many devices do not support a background intensity independent from the foreground intensity (most terminals, for example). When that is the case, the runtime declares the background intensity to be low-intensity.

BELL

When set to “0” (off, false, no), this variable will inhibit all bells generated by ACCEPT and DISPLAY statements. Note that this will override explicit WITH BELL clauses as well as implicit bells. The default setting is “1” (on, true, yes).

BOXED_FLOATING_WINDOWS

When this variable is set to “1” (on, true, yes) all floating windows displayed on character-based hosts are drawn with a border (box). If this variable is set to “0” (off, false, no), floating windows are drawn with a border only when the BOXED phrase appears in the statement that creates the window. The default value for this variable is “0” (off, false, no). This variable has an effect only on character-based host systems.

BTRV_MASS_UPDATE

When this variable is set to “1” (on, true, yes), a Btrieve file is opened in exclusive mode. No other processes may open the file at the same time. When this variable is set to “0” (off, false, no), a Btrieve file is opened in accelerated mode, and other processes may open the file.

BTRV_NOWRITE_WAIT

When a user tries to write to a locked file, the Btrieve interface performs a 15-second “wait and retry” operation before it reports an error condition (99) to the runtime. Setting the BTRV_NOWRITE_WAIT configuration variable to “TRUE” (the default) prevents this operation from occurring, and the error condition is reported immediately. Setting BTRV_NOWRITE_WAIT to “FALSE” causes the interface to perform the wait and retry operation.

BTRV_USE_REPEAT_DUPS

This variable controls whether duplicate keys are created as LINKED duplicates (the Btrieve default) or REPEATING duplicates. When set to the default value of “FALSE”, the Btrieve interface creates all duplicate keys as LINKED duplicates. When set to “TRUE”, the Btrieve interface creates all duplicate keys as REPEATING duplicates.

In cases where a large number of users are accessing files, you may experience better performance if you set this variable to “TRUE”. See the Pervasive documentation for information on REPEATING duplicates and why you may want to use them.

BUFFERED_SCREEN

This variable controls how the Terminal Manager should buffer its output on UNIX systems. Normally, all queued output is sent to the screen after each DISPLAY statement. If this value is set to “1” (on, true, yes), then output is sent only when the internal buffer is full, an ACCEPT statement is executed, or an internal 1-second timer expires. This can speed up output on some systems by reducing the number of times the operating system is called. It will also cause a short delay before messages are seen. We recommend keeping this setting at the default “0” (off, false, no) unless you are experiencing poor screen performance.

CALL_HASH_SIZE

The setting of this variable controls the size of the hash table that tracks CALL statements to COBOL subprograms. Each CALL statement tracks its last resolution (target object, entry point, and owning thread). When the resolution is unchanged in a subsequent execution of the CALL statement, the CALL uses the saved information, contributing to improved performance. Each program contains its own copy of this table, so the size should generally be set to a small value.

The default value for CALL_HASH_SIZE is “31”. The only reason to change this setting is if your programs contain hundreds of individual CALL statements that target distinct objects. In this case, you may see a small

performance improvement by setting CALL_HASH_SIZE to a larger value. You can disable the tracking of these CALL statements by setting the value of CALL_HASH_SIZE to “0”.

Note that this mechanism consumes a small amount of memory for each CALL statement. This memory is recovered when the calling object is removed from memory. The amount is machine-specific, but is normally well under 100 bytes per CALL.

CANCEL_ALL_DLLS

This variable is used to change the default behavior of a CANCEL ALL statement. The default behavior is for CANCEL ALL to free all DLLs and UNIX/Linux shared object libraries loaded with a prior CALL statement. Setting CANCEL_ALL_DLLS to “0” (off, false, no) indicates that CANCEL ALL should not free any DLLs or shared object libraries. If you want to free a particular DLL or shared library when CANCEL_ALL_DLLS is set to “0”, you must specify the DLL’s name in a CANCEL statement.

The default value of CANCEL_ALL_DLLS is “1” (on, true, yes).

CARRIAGE_CONTROL_FILTER

The value of this variable affects how carriage control characters are treated when found in LINE SEQUENTIAL data files.

RM/COBOL version 2 handles carriage control characters in a line sequential file differently on different systems. By default, both ACUCOBOL-GT and RM/COBOL-85 remove carriage control characters from input records for line sequential files. This is the ANSI standard. RM/COBOL version 2, however, does not remove form-feed characters on MS-DOS machines and does not remove form-feed or carriage return characters on UNIX systems. Some existing RM/COBOL version 2 programs depend on this behavior.

You can retain any or all of these characters in the input record by setting CARRIAGE_CONTROL_FILTER to a value as follows:

1	form-feed characters are retained
2	carriage return characters are retained
4	line-feed characters are retained

You can specify two or three characters to be retained by adding the appropriate values together. For example, a value of "6" retains carriage returns and line feeds (2 plus 4). Setting the variable to "0" causes the default action of removing all three characters.

The default value is "0".

Note: On VMS systems, carriage control information is not placed directly into data records and is instead maintained separately. For this reason, the CARRIAGE_CONTROL_FILTER setting has no effect on VMS systems and should not be considered portable to those machines.

CBLHELP

Define the CBLHELP configuration variable to the location of the "cblhelp" debugger help file. The definition must include the path and filename. For example:

```
CBLHELP /home/acucobol8/etc/cblhelp
```

CGI_AUTO_HEADER

This variable is used when you are writing a Common Gateway Interface (CGI) program in COBOL. It allows you to suppress the output of the HTML header.

Set CGI_AUTO_HEADER to "0" (off, false, no) if you want to suppress the output of the HTML header. This can be useful when you want to execute a CGI program and include its output into an existing flow of HTML text. For example, with server-side includes, or SSI, you can instruct the Web server

to execute a subprogram in the manner of CGI and then incorporate its output right into the HTML document before sending it to the requesting client. The default value is "1" (on, true, yes).

For information about writing a CGI program in COBOL, refer to Chapter 4 in *A Programmer's Guide to the Internet*.

CGI_CLEAR_MISSING_VALUES

This variable is used when you are writing a Common Gateway Interface (CGI) program in COBOL. It allows you to control the behavior of the ACCEPT statement when CGI variables do not exist in the CGI input data.

By default, ACCEPT sets the value of numeric data items to zero and non-numeric data items to spaces if a CGI variable does not exist. Set the CGI_CLEAR_MISSING_VALUES configuration variable to "0" (off, false, no) if you do not want ACCEPT to change the value of the data item if the corresponding CGI variable is missing from the CGI input data.

CGI_CONTENT_TYPE

By default, the output generated by your CGI program is mapped as HTML content. To associate your CGI output with a MIME content type other than "text/html", use the CGI_CONTENT_TYPE configuration variable. This variable lets you control the content type information in the header of output files created by ACUCOBOL-GT. Such information informs recipients of the type of content that they are about to receive.

Using this variable, you can configure your CGI program for many types of output, including eXtensible Markup Language (XML) or Wireless Markup Language (WML) for Wireless Application Protocol (WAP) devices like mobile phones.

Whichever format you choose, the US-ASCII character set is applied to the output by default. If you want the CGI output to be mapped to an alternate character set such as ISO-8859-I (Western European), then you can specify the character encoding set to use with the variable as well.

Include this variable in your runtime configuration file as follows:

```
CGI_CONTENT_TYPE contenttype; charset=encoding_set
```

Where *contenttype* is the MIME content type of the generated output, and *encoding_set* is the preferred character encoding set to use.

For example, the WML content type for WAP mobile phones is "text/vnd.wap.wml". To associate your CGI output with WML, include the following in your configuration file:

```
CGI_CONTENT_TYPE text/vnd.wap.wml
```

If you want your WML output to be mapped to the Western European character set, include the following:

```
CGI_CONTENT_TYPE text/vnd.wap.wml; charset=iso-8859-I
```

The content type for eXtensible Markup Language (XML) documents is "text/xml". If your program generates XML data, include the following:

```
CGI_CONTENT_TYPE text/xml
```

Caution: To avoid overriding other Content-Type associations, we suggest that you create a different configuration file for each of the MIME Content-Type associations that you make in your Web server setup.

Please note that if you use this variable, the external forms indicated in your program's DISPLAY syntax *must* contain the appropriate content. In other words, if you associate your program with the "text/xml" content type, the forms must be ".xml" documents with XML syntax. If you associate it with "text/vnd.wap.wml", the forms must be ".wml" documents with WML syntax. Your program can DISPLAY virtually any type of data, as long as the Content-Type ID corresponds to the external form file that you provide.

Be aware that if you do not use the proper file extension for your external form documents, the Web server will interpret the data as HTML and display the wrong data. WML and XML are also more sensitive to syntax errors than HTML.

In addition, note that the capabilities of the configuration entry **CGI_NO_CACHE** may be affected by the content type that you choose.

For information about writing a CGI program in COBOL, refer to Chapter 4 in *A Programmer's Guide to the Internet*.

CGI_NO_CACHE

This variable allows you to choose whether the HTML output of your Common Gateway Interface (CGI) program will be cached by the requesting client.

The default value is "1" (on, true, yes), which means there is no caching. By default, the runtime generates "Pragma: no-cache" in the HTML response header that gets sent to the standard output stream. If you set CGI_NO_CACHE to "0" (off, false, no), the runtime suppresses this line of the response header, and the requesting client caches the output.

For information about writing a CGI program in COBOL, refer to Chapter 4 in *A Programmer's Guide to the Internet*.

CGI_STRIP_CR

When this variable is set to "1" (on, true, yes), the runtime automatically removes carriage return characters from data entered in HTML TEXTAREAS (multiple line entry-fields). Stripping the carriage returns from this kind of input prevents double-spacing problems, as well as conflicts that may arise if the data is used in a context that does not expect a carriage return character to precede each line feed character. Some browsers send a carriage return line feed sequence to the CGI program, and when this sequence is written to a file on operating systems that terminate text lines with line feed characters only, the file may appear to be double spaced. The default value for this variable is "0" (off, false, no).

For example, if you enter the following three lines in a TEXTAREA for a field called "thetext":

```
Sometext line 1
Sometext line 2
Sometext line 3
```

The browser sends the following to the CGI program:

```
thetext=Sometext+line+1%0D%0ASometext+line+2%0D%0ASometext+line+3%0D%0A
```

If the CGI_STRIP_CR is set to “1” (on, true, yes), the runtime strips the carriage return characters so that the input line is the following:

```
thetext=Sometext+line+1%0ASometext+line+2%0ASometext+line+3%0A
```

For information about writing a CGI program in COBOL, refer to Chapter 4 in *A Programmer’s Guide to the Internet*.

CHAIN_MENUS

When this variable is set to “1” (on, true, yes), the runtime system automatically destroys any menu displayed by a program performing a CHAIN or CALL PROGRAM. This destruction is accomplished with the WMENU-DESTROY-DELAYED operation of the **W$MENU** library routine. The effect is that the menu is not actually destroyed until the chained-to program displays a new menu. Setting this variable to “0” (off, false, no) inhibits the destruction of the menu. The default value is “off”.

CHECK_USING

When this value is “1” (on, true, yes), the runtime system tests each use of a LINKAGE data item to make sure that the item passed by the calling program is at least as large as the item declared by the called program. This ensures that unallocated memory is not accidentally referenced.

Setting this value to “0” (off, false, no) inhibits the parameters size matching test. It also inhibits the runtime test that verifies that all parameters of a subprogram are passed by the caller.

The default value is “1”. If you set this value to “0”, you should test your programs carefully to avoid corrupting memory.

Note: It is common for programs in some OLTP environments to specify a data item length as a negative value. By default, this produces a runtime error. Set CHECK_USING to “0” to override the default behavior.

CISAM_COMPRESS_KEYS

This variable allows you to turn off key compression in C-ISAM files. By default, the ACUCOBOL-GT interface to C-ISAM uses the key compression feature of C-ISAM. But some C-ISAM emulators do not understand the compressed keys and cannot read the files created. This variable allows you to turn off the compression.

When the variable is set to "0" (off, false, no), key compression is not used. When it's set to the default of "1" (on, true, yes), key compression is used. Note that this value is examined each time a file is created, so its setting can be changed for each file. The setting is meaningful only when the file is created. After that, the file retains its compression mode.

CLOSE_ON_EXIT

When set to "1" (on, true, yes), this variable enables the automatic closing of all files except print files when a program performs an EXIT PROGRAM statement. When set to "2" it enables the automatic closing of all files when a program exits. When set to "0" (off, false, no), no files will be automatically closed. For more information, see the *ACUCOBOL-GT User's Guide*, **section 2.8.5, "File Handling Options."** The default value is "0".

COBLPFORM

This configuration variable is used to define and print to printer channels C01-C12. Specify the line numbers for each channel with the COBLPFORM configuration variable. Null entries are ignored. Those channels that have line number zero, function-names S01-S052, CSP, or are undefined, are set to line 1.

Example 1

```
COBLPFORM 1:3:5:7:9:11:13:15:17:19:21:23
```

In this example C01 equals 1, C02 equals 3, and so on.

Example 2

```
COBLPFORM :3::5: :9
```

In this example, C01 equals 3, C02 equals 5, C03 equals 1, and C04 equals 9. You can specify only a single line number for each channel.

In example 2 above, channels C05 - C12 are undefined. If a print statement specifies channel C05 - C12, the line is printed at line 1. In addition, in the example shown, C03 equals 1 because its value is a space and therefore undefined.

Any WRITE BEFORE/AFTER PAGE statements cause positioning to be at line 1. Each line advance increases the line number by one. A request to skip to a line number less than or equal to the current line causes a new page to begin. The appropriate number of line feeds are then generated.

CODE_CASE

This configuration variable allows you to adjust the case of an object file name that is specified in a CALL statement. It has five possible values:

NONE	or	“0”	(the default) object file names are not translated
LOWER	or	“1”	object file names are translated to lower case, including directory (path) elements
UPPER	or	“2”	object file names are translated to upper case, including directory (path) elements
LOWER_BASE	or	“3”	object file names are translated to lower case, excluding directory (path) elements
UPPER_BASE	or	“4”	object file names are translated to upper case, excluding directory (path) elements

Translation occurs before the **CODE_SUFFIX** and **CODE_PREFIX** configuration options are applied. You should make sure that those variables specify the correct case. For a complete description of the runtime CALL handling procedure, see **Section 2.10.1** in Book 1, *ACUCOBOL-GT User’s Guide*.

CODE_MAPPING

This configuration variable allows you to modify CALL, CHAIN, and CANCEL names at runtime. This can be particularly useful if you are using AcuServer or AcuConnect. When this variable is set to "1" (on, true, yes), every CALL, CHAIN, and CANCEL statement checks the current configuration for a name that matches the CALL name. This is handled in the same way that file name processing is done (the environment is checked for an uppercase version of the name, with any hyphens treated as underscores). If a matching name is found, its value is substituted. This is done recursively until no more matching names are found.

After this substitution occurs, the CALL name handling proceeds normally (and includes any effects of CODE_PATH, CODE_SUFFIX, and CODE_CASE).

For example, with CODE_MAPPING set to "1", if your configuration file had the following entry:

```
MYPROG  @sun:/app/myprog
```

Then CALL "MYPROG" would act the same as CALL "@sun:/app/myprog".

Thin client applications may find the CODE_MAPPING mechanism useful for automatically adding the "@[DISPLAY]:" prefix to the name of the DLL to run on the display host. For example, if your configuration file includes the entry:

```
mylib.dll  @[DISPLAY]:mylib.dll
```

Then the statement

```
CALL  "mylib.dll"
```

is interpreted as

```
CALL "@[DISPLAY]:mylib.dll"
```

causing "mylib.dll" to run on the display host.

Those wanting to specify the DLL calling conventions will also find CODE_MAPPING useful. For example, if you use the following configuration entries:

```
funcA=funcA@__stdcall
funcB=funcB@__cdecl
```

then the statement

```
CALL "funcA"
```

calls funcA using the stdcall calling convention and

```
CALL "funcB"
```

calls funcB using the cdecl convention.

For more information about calling DLLs from thin client applications, see section 7.2.6 of the *AcuConnect User's Guide*. For information on calling DLLs in general, refer to Chapter 3 of *A Guide to Interoperating with ACUCOBOL-GT.*

The default value for this variable is "0" (off, false, no).

CODE_PREFIX

This variable defines a set of directories that the runtime searches to locate a program object file. The default value is "." (current working directory). Code and data file search paths are described in more detail in **Section 2.8.2** of the *ACUCOBOL-GT User's Guide*.

Directories can be a mix of relative and absolute paths. Entries are separated by spaces. A space is a valid separator on all systems. Alternatively, on UNIX systems you can also separate entries with a colon. On Windows systems a semicolon can be used. On VMS systems a comma can be used.

Include a "^" (carat) to specify the directory containing the calling program. For example:

```
CODE_PREFIX . /cobbin ^
```

causes the runtime to search the current working directory, followed by the "cobbin" root directory, followed by the directory containing the calling program.

You can specify a directory path that contains embedded spaces if you surround the path with quotation marks. For example:

```
CODE_PREFIX C:\“program files” C:\Customers
```

Remote name notation is allowed if your runtime is *client-enabled*. See *User's Guide* **Section 5.2.1** and **Section 5.2.2** for more information about client-enabled runtimes and remote name notation.

Up to 4096 characters can be specified for the value of this variable.

CODE_SUFFIX

The value of this variable is automatically appended to the end of program filenames when those names do not contain explicit suffixes. A suffix is the portion of a filename that follows a period. For example, if CODE__SUFFIX is set to “COB”, then CALL “PGMFILE” causes the runtime to look for the file “PGMFILE.COB”. The default value is empty.

CODE_SYSTEM

The runtime configuration variable CODE_SYSTEM tells the runtime if double-byte character data is being accepted or displayed, and which code system (that is, which standard for encoding Japanese and other Asian character sets, for example) is being used. Each code system has a range of values that it allows within each byte of a two-byte character, so identifying the code system allows the runtime to recognize character boundaries when it is processing double-byte data for ACCEPT and DISPLAY statements.

Setting CODE_SYSTEM to the proper value allows your COBOL applications to handle input and display of double-byte character data without source program changes. The syntax is:

```
CODE_SYSTEM  setting
```

The table below shows the possible settings of the CODE_SYSTEM variable, the code system to which each setting refers, and some examples of operating systems to which the particular code system applies:

Setting	**Code System**	**Op. System Examples**
BIG5	Big Five (Taiwan)	Chinese DOS, Windows
DBC	Acucorp Generic Double-byte Coding Scheme	other double-byte machines
EUC	Extended UNIX	Most UNIX machines
GB	Code of Chinese Graphic Character Set (People's Republic of China)	Chinese DOS, Windows
KSC	Korean Character Standard	Korean DOS
SJC	Shift JIS Code (Japanese Industrial Standard)	DOS/V, Windows, some UNIX machines

The default "0" means ASCII or EBCDIC single-byte characters.

The following table shows the decimal values that the respective code systems allow for each byte of the two-byte character:

Code System Setting	**1st byte**	**2nd byte**
BIG5	161 - 254	64 - 126
(second format)	161 - 254	161 - 254
DBC	128 - 255	128 - 255
EUC	142	161 - 223
(second format)	161 - 254	161 - 254
GB and KSC	161- 254	161 - 254
SJC	129 - 159	64 - 252 (not 127)
(second format)	224 - 239	64 - 252 (not 127)

Note: The first and second byte values are co-dependent; that is, both values must fall within the respective ranges shown in the table. If either value is not within its allowable range, then each byte will be treated as a single character.

COLOR_MAP

This variable can be used to assign colors to programs that do not contain explicit color settings. This is described in **Section 4.4.1** of *ACUCOBOL-GT User's Guide*. The default value is empty.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

COLOR_MODEL

This variable is typically used when a character-based application is moved to a graphical environment. Use the COLOR_MODEL setting to perform uniform changes to your program's color scheme. These changes are represented by rules that act on your colors. An example of a rule is "exchange the foreground and background colors". Use COLOR_MODEL to change your color scheme in a global way.

The default color model is model "0". It causes no changes to occur to your color scheme. The remaining 10 models are "1" through "10".

- The odd-numbered models transform only those parts of your program that are entirely black and white. Any character position that contains any color is left unchanged.
- The even-numbered models apply the changes regardless of color. When selecting a COLOR_MODEL, you can ignore the even-numbered models if you are satisfied with the color portions of your program.

Each color model is actually a composite; it's the equivalent of two or more configuration file variable settings:

COLOR_MODEL	Equivalent Configuration File Variable Settings
"1"	COLOR_TRANS "5" INTENSITY_FLAGS "34" BACKGROUND_INTENSITY "1"
"2"	COLOR_TRANS "4" INTENSITY_FLAGS "34" BACKGROUND_INTENSITY "1"
"3"	COLOR_TRANS "3" INTENSITY_FLAGS "34"
"4"	COLOR_TRANS "1" INTENSITY_FLAGS "34"
"5"	COLOR_TRANS "1" INTENSITY_FLAGS "129"
"6"	COLOR_TRANS "1" INTENSITY_FLAGS "129" BACKGROUND_INTENSITY "2"
"7"	COLOR_TRANS "3" INTENSITY_FLAGS "161"
"8"	COLOR_TRANS "1" INTENSITY_FLAGS "161"
"9"	COLOR_TRANS "3" INTENSITY_FLAGS "193"
"10"	COLOR_TRANS "1" INTENSITY_FLAGS "193"

For more information, see **Chapter 9** in Book 2, *ACUCOBOL-GT User Interface Programming*.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

COLOR_TABLE

This variable is typically used when a character-based application is moved to a graphical environment. Use the COLOR_TABLE variable to cause transformations of individual color combinations. For example, a COLOR_TABLE entry might cause a red foreground on a black background to be translated to a white foreground on a blue background.

Follow the word COLOR_TABLE with the original foreground and background numbers, separated by a comma. Follow these by an equals sign and then the new foreground and background numbers, separated by a comma.

For example, to transform the color combination of foreground 5 on background 2, to foreground 13 on background 2, you would use:

```
COLOR_TABLE   5, 2 = 13, 2
```

These are the possible values for foreground and background settings:

Color	Color value
low-intensity Black	1
low-intensity Blue	2
low-intensity Green	3
low-intensity Cyan	4
low-intensity Red	5
low-intensity Magenta	6
low-intensity Brown	7
low-intensity White	8
high-intensity Black	9
high-intensity Blue	10

Color	Color value
high-intensity Green	11
high-intensity Cyan	12
high-intensity Red	13
high-intensity Magenta	14
high-intensity Brown	15
high-intensity White	16

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

COLOR_TRANS

This variable is typically used when a character-based application is moved to a graphical environment. It determines how the initial colors in an application are transformed. By default, it is set to "0", which causes no transformation. It may be set to any of these values:

1 This mode causes the foreground and background colors to be exchanged for each other. This is equivalent to running the entire program in reverse-video.

2 This causes white to be exchanged for black and black to be exchanged for white. The foreground and background colors are transformed independently. For example, a green foreground on a black background would turn into a green foreground on a white background. This setting usually has the effect of transforming a black background into white while maintaining the general color scheme of the application.

3 The foreground and background colors are exchanged for each other, but only if they are both black or white. If either the foreground or background contains a color other than black or white, then nothing happens. This is equivalent to running the monochrome parts of your program in reverse-video while maintaining the color portions unchanged.

4 The foreground and background colors are exchanged for each other, but only if the background is black. This mode ensures that you never have a black background.

5 If the colors are foreground white and background black, they are exchanged for each other. Otherwise, nothing happens.

Generally speaking, you could use the COLOR_TRANS variable as a starting point in converting an application to appear more natural under Windows. (It's easier to start with **COLOR_MODEL** instead.) Note that if your application is entirely black-and-white, then the first three COLOR_TRANS options are essentially identical. See **Chapter 9** in Book 2, *ACUCOBOL-GT User Interface Programming* for color mapping suggestions.

COLUMN_SEPARATION

This configuration variable sets the default separation distance between columns in a list box. The value is expressed in 10ths of characters. For example, to place a 1/2 character space between list box columns, you would assign a value of "5". See the description of the list box SEPARATION property for more information. The default value of COLUMN_SEPARATION is "5".

COMPRESS_FACTOR

This variable is used to define the compression factor that is applied to indexed files (if the indexed file system supports compression; Vision does). COMPRESS_FACTOR is applied when a file is created with the WITH COMPRESSION phrase in the ASSIGN clause of the file's SELECT and the COMPRESSION CONTROL VALUE phrase is either omitted or specifies a value of "1". If the COMPRESS CONTROL VALUE phrase specifies a value other than one, that value is used and the value of COMPRESS_FACTOR is ignored.

COMPRESS_FACTOR can be set to any value within the range zero to 100. Zero specifies no compression. Values from 2-100 are treated as a percentage that specifies how much of the space saved by file compression is removed from the compressed records. A value of 1, the default, is a special

case that causes the standard default compression factor of 70 to be applied. Note that a file's compression factor is set when the file is created and cannot later be changed except by recreating the file or rebuilding the file with **vutil**. For more information about Vision record compression, see Book 1, *ACUCOBOL-GT User's Guide*, **section 6.1.6.1, "Compression."**

COMPRESS_FILES

Setting this configuration variable to "1" (on, true, yes) causes ACUCOBOL-GT to treat all indexed files as if they had the WITH COMPRESSION phrase specified for them. This affects the status of newly created files only. When the configuration variable is set to the default value of "0" (off, false, no), only those files with the WITH COMPRESSION phrase specified will be compressed. You can specify the amount of compression with the **COMPRESS_FACTOR** configuration variable.

CONTROL_CREATION_EVENTS

This variable applies to those using ActiveX controls in their ACUCOBOL-GT programs. Use it if you want to allow events during the creation of an ActiveX control. By default, the runtime ignores events from all controls while it is creating an ActiveX control. If it did not, subsequent operations on the ActiveX control could fail.

If you are using a control that delivers significant information using events and you don't want to miss those events while you are creating a new control, set the CONTROL_CREATION_EVENTS variable to "1" (On, True, Yes). Alternatively, you could avoid creating an ActiveX control when you are expecting an event.

By default, this variable is set to "0" (Off, False, No).

CURRENCY

This configuration variable can be used to set the desired currency character at runtime. It is followed with the desired character. The default is to use the character specified in the source program's CURRENCY phrase (or "$" if the CURRENCY phrase is absent).

CURSOR_MODE

This configuration variable determines when the cursor should be visible. It has three values:

1 always visible

2 always invisible

3 invisible except during ACCEPT statements, then visible

The default value is "3". Note that a change to the value does not take effect until the next ACCEPT or DISPLAY statement. The sample program MENUBAR.CBL contains examples of how to modify the cursor from within a program. The cursor is always set to Normal, Visible when the runtime exits or when the SYSTEM library routine is called.

CURSOR_TYPE

This configuration variable determines the way the cursor looks on character-based systems. It can be set to one of the following values ("3" is the default):

1 normal cursor (usually underscore)

2 bright cursor (usually block)

3 normal cursor except when in insert mode, then bright

4 vertical bar (when available)

DEBUG_NEWCOPY

This variable determines whether a new copy of a COBOL program being debugged is loaded from disk whenever the debugger is active. By default, DEBUG_NEWCOPY is "True" so that you can continue to use the logical cancel and code caching feature while the debugger is active.

Set DEBUG_NEWCOPY to "False" if you want to keep caching enabled and have the debugger use the copy of the program in the cache instead of reading a new copy from disk. You must then do one of two things:

- Start the debugger before the first execution of the program in the current process
- In a transaction processing system, use the CICS command, CEMT SET PROGRAM(*program_name*) NEWCOPY, to load a new copy of the program to be debugged.

Note: The ACUCOBOL-GT debugger periodically reads source code from the object file on disk. When the program is cached (as the result of a logical cancel), the object file is closed and could be replaced or deleted. For the debugger to function correctly, it must keep the object file open and ensure that the object code in the disk file is identical to the code in memory. Therefore, if the program has been cached (using LOGICAL_CANCELS and DYNAMIC_MEMORY_LIMIT), the debugger unloads the program from the cache, reopens the object file, and reloads the object code from memory. For more information, see**section 6.3, "Memory Management,"** in Book 1, *ACUCOBOL-GT User's Guide*.

DECIMAL_POINT

This configuration variable sets the character to be used as the program's decimal point. Follow it with the desired character. If you use this variable to set the decimal point to a comma, then the place and function of the

decimal point and comma are reversed (just like the phrase DECIMAL_POINT IS COMMA). The default is to use the decimal point specified by the program's source.

Note: You do not have to change the value of DECIMAL_POINT to match the decimal point used by floating point values received from external components. The runtime automatically makes the correct adjustment.

DEFAULT_FILESYSTEM

This variable determines the file system to be used if no *filename*_FILESYSTEM variable is set for a file and none of the other file system variables are set for the file type. The other variables you can use to specify a different file system for indexed, relative, or sequential files, are:

DEFAULT_IDX_FILESYSTEM
DEFAULT_REL_FILESYSTEM
DEFAULT_SEQ_FILESYSTEM

For example, setting:

```
DEFAULT_IDX_FILESYSTEM EXTFH
```

causes all indexed files to go through the EXTH interface. Unless another file system is specified, ACUCOBOL-GT uses its native file handler for relative and sequential files.

Note: The DEFAULT_IDX_FILESYSTEM variable is a synonym for the existing configuration variable, **DEFAULT_HOST**.

By default, all file access is handled by the ACUCOBOL-GT native file handler. For those file types you want to access using an EXTFH library, you need to set one or more of these configuration variables to "EXTFH".

For example, to use the DB2 library to access indexed files, you would set the following two configuration variables:

```
A_EXTFH_LIB=/usr/lpp/cics/lib/libxfhdb2sa.a(libxfhdb2_shr.o)
DEFAULT_IDX_FILESYSTEM=EXTFH
```

For information on specifying EXTFH library and function names to use with the EXTFH interface, see section 11.6, “Working With an EXTFH Interface,” in *A Guide to Interoperating with ACUCOBOL-GT.*

DEFAULT_FONT

This variable defines which font to use for the DEFAULT_FONT (for a description of this font, see Format 3, **ACCEPT Statement** in Book 3, *ACUCOBOL-GT Reference Manual*). When DEFAULT_FONT is set to “0” (the normal setting), the font used depends on the host system as follows:

System	Font Used
Graphical system	MEDIUM-FONT
Non-graphical system	FIXED-FONT

You can set DEFAULT_FONT to one of the following values to use a different font. The following words are valid settings:

Setting	Font Used
TRADITIONAL	TRADITIONAL-FONT
FIXED	FIXED-FONT
LARGE	LARGE-FONT
MEDIUM	MEDIUM-FONT
SMALL	SMALL-FONT

Due to the way the runtime initializes the windowing subsystem, the DEFAULT_FONT setting is effective only when it is placed in the configuration file or the host system’s environment. Setting DEFAULT_FONT from inside a COBOL program has no effect.

DEFAULT_HOST

When the application program is opening an existing file or creating a new file, you need to tell the runtime which file system to use. You accomplish this with one of two configuration variables: DEFAULT_HOST or *filename*_HOST.

```
DEFAULT_HOST filesystem
```

designates the file system to be used for files that are not individually assigned. If this variable is not given a value, and if you have not individually assigned a file system (with *filename*_HOST), the Vision file system is used.

Note: The DEFAULT_IDX_FILESYSTEM variable is a synonym for DEFAULT_HOST.

DEFAULT_MAP_FILE

Use this variable to point to the character map file used for translating international character sets between machines that use differing character codes. The map file is a simple text file that you create with an editor of your choice. Each line in the map file must contain two values in either decimal or hexadecimal: the character code of the character on the client machine, and the character code of the same character on the remote machine. Use a # sign to indicate a comment.

The runtime first searches for the configuration variable **server_MAP_FILE** and, if it is found, uses that setting to locate the map file. If that variable is not set, the runtime searches for DEFAULT_MAP_FILE. If this variable is not set, then no character translation is done.

Example:

```
DEFAULT_MAP_FILE = c:\etc\pc_iso.txt
```

DEFAULT_PROGRAM

Use this variable to specify the name of the program to be run by default if no program name is specified on the command line. The name you give here is treated exactly as it would be if you had typed it on the command line. The default is “cbl.out”.

Remote name notation is allowed for this variable if your runtime is client-enabled. See *ACUCOBOL-GT User’s Guide* **Section 5.2.1** and **Section 5.2.2** for more information about client-enabled runtimes and remote name notation.

DEFAULT_TIMEOUT

This variable is used by the runtime and Web Runtime to define the length of time, in seconds, that they will wait for a response from **acuserve** before timing out. The default value for this variable is 25 seconds. Some networks have long connect times and the default value may not be long enough to allow the application to connect. For example, to change the timeout default of 25 seconds to one minute, you would set the following:

```
DEFAULT_TIMEOUT = 60
```

If the runtime or Web Runtime receives an error before the specified time, they will time out immediately. This variable only works with AcuServer client runtimes and AcuServer client Web Runtimes.

DISABLED_CONTROL_COLOR

This variable allows character-based hosts to use color and video attributes to distinguish disabled screen controls from enabled controls. It can be set to a variety of numeric values that express combinations of attributes. When it is set to “0” (off, false, no), disabled controls appear the same as enabled ones. See **COLOR Phrase** in Book 3, *ACUCOBOL-GT Reference Manual*, for a description of other numeric values that can be used.

DISPLAY_SWITCH_PERIOD

This variable helps to determine how frequently the program's threads will switch control. After a thread performs the value of DISPLAY_SWITCH_PERIOD display operations, the runtime switches control to another thread (if one exists). Note that because a single DISPLAY statement can compile into multiple "display operations," and because thread switching is also affected by other program operations (such as file I/O), it is impossible to predict or control when a thread will change control based on the presence of DISPLAY statements in the source.

By setting DISPLAY_SWITCH_PERIOD to lower values, you cause windows that are updated by multiple threads to update more uniformly, but more time will be spent in the thread switching code. Setting DISPLAY_SWITCH_PERIOD to higher values will decrease the switching overhead, but will also cause the windows to update in blocks. In most cases, applications that use threads will run well with the default setting of "10".

DLL_CONVENTION

This variable allows you to specify the calling convention used to call DLLs. When this variable is set to "0", the cdecl (standard C) interface is used. When this variable is set to "1", the stdcall (Pascal/WINAPI) interface is used. The default for this variable is "0".

Note that there are a few ways to override the DLL_CONVENTION setting:

- You can specify a list of DLL names and calling conventions in the **SHARED_LIBRARY_LIST**configuration variable. This variable can be set in the environment, in the runtime configuration file, or programmatically with the SET ENVIRONMENT statement.
- You can specify the calling convention for individual library functions in the COBOL CALL statement.
- You can set the **CODE_MAPPING** variable to "1", then use configuration entries to specify the calling convention for individual functions.

- You can specify a list of DLL names and calling conventions using the "-y" runtime option. (see the "-y" listing in **Section 2.3.1**, ACUCOBOL-GT User's Guide.)

In all of these cases, the runtime uses the specified calling convention and ignores the value of the DLL_CONVENTION configuration variable. See Chapter 3 in *A Guide to Interoperating with ACUCOBOL-GT* for more details about calling DLLs.

DLL_SUB_INTERFACE

This variable identifies the routine to be used as the "sub" interface routine within a DLL. It applies only to Windows systems. Set DLL_SUB_INTERFACE to the name of the routine you want to use. This name may be "sub" or any name you choose. The runtime checks DLL_SUB_INTERFACE when a DLL is loaded. You may change its value afterwards without any effect on DLLs that have already been loaded.

If DLL_SUB_INTERFACE is empty (default), the runtime does not look for a "sub" interface routine in a called DLL.

DLL_USE_SYSTEM_DIR

When a program calls an unloaded DLL, the value of this variable determines whether the runtime attempts to find the DLL in the Windows and System folders. When set to the default value "1" (on, true, yes), the runtime looks in the Windows and System folders. When set to "0" (off, false, no) the runtime does not look in the Windows and System folders. See Chapter 3 in *A Guide to Interoperating with ACUCOBOL-GT* for more details about calling DLLs.

DOS_BOX_CHARS

This variable allows you to redefine the line drawing characters used with the Windows console (DOS-box) runtime. The value of DOS_BOX_CHARS is a list of characters that draw the line segments. It should be a list of 13

space-delimited characters that correspond, in order, to the line segments as listed below. To redefine the DOS line drawing characters, specify the characters you want in the following order:

1. horizontal line
2. vertical line
3. upper left corner
4. upper right corner
5. lower left corner
6. lower right corner

Four three-way intersections -

7. missing bottom line
8. missing left line
9. missing top line
10. missing right line

and -

11. the four-way intersection
12. upper-half block
13. lower-half block

These line drawing characters may also be specified by decimal value. Characters that are not available on a particular machine should be specified with the decimal value “0”.

The default value for DOS_BOX_CHARS depends on the **CODE_SYSTEM** configuration variable. If CODE_SYSTEM is not set, or is set to “0” (or ASCII or EBCDIC), the default is:

```
DOS_BOX_CHARS 196 179 218 191 192 217 193 195 194 180 197 223 220
```

If CODE_SYSTEM is set to a non-zero value, which is the case in the ACUCOBOL-GT JPN version, the default is:

```
DOS_BOX_CHARS 6 5 1 2 3 4 21 25 22 23 16 0 0
```

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

DOS_SYS_EMULATE

When set to "1" (on, true, yes), this variable causes a program running in the Windows console runtime to run as if it's in a DOS environment. One of its effects is that it prevents such a program from attempting to display GUI screens. It is set to "0" (off, false, no) by default. This variable has meaning only in Windows environments.

DOUBLE_CLICK_TIME

This variable has meaning only on systems that support a mouse. It controls the "double-click" rate on systems that do not control it themselves.

Specify the maximum time (in hundredths of a second) allowed between two clicks that are to be interpreted as a double-click. For example, if DOUBLE_CLICK_TIME were set to "75" (three-quarters of a second), then any two clicks that occur at least that close together would be considered a double-click rather than two single clicks.

The default value is "50" (one-half of a second).

DUPLICATES_LOG

This variable is used during bulk addition of Vision files. It causes Vision to write files rejected for having illegal duplicate keys to a log file. Set DUPLICATES_LOG to the name of a file in which to store the records. If

this log file already exists, it is overwritten. You should use a separate log file for each file opened with bulk addition. You can do this by changing the setting of DUPLICATES_LOG between OPEN statements, as follows:

```
SET ENVIRONMENT "DUPLICATES_LOG" TO "file1.rej"
OPEN OUTPUT FILE-1 FOR BULK-ADDITION

SET ENVIRONMENT "DUPLICATES_LOG" TO "file2.rej"
OPEN EXTEND FILE-2 FOR BULK-ADDITION
```

If no duplicate records are found, the log file is removed when the Vision file is closed. If DUPLICATES_LOG has not been set, or is set to spaces, no log file is created.

Note: The duplicate-key log file may not be placed on a remote machine using AcuServer. The log file must be directly accessible by the machine that is running the program.

See **Section 6.1.6.3, "Bulk addition mode for Vision,"** in Book 1, *ACUCOBOL-GT User's Guide,* for instructions on how to read the log file.

DYNAMIC_FUNCTION_CALLS

This variable allows you to specify a list of functions or function name prefixes that the runtime treats as dynamic functions and therefore searches first, before searching the disk for COBOL programs. This speeds the resolution of calls to functions in the current process or in a shared library.

The runtime checks call names for matches in the list specified in the variable. If a match is found, the runtime attempts to call the routine directly in the current process and in each of the loaded shared libraries. If these attempts fail, the runtime attempts to load a COBOL program with the specified name.

Set DYNAMIC_FUNCTION_CALLS to a space- or comma-delimited list of names of frequently called functions that are linked into the current process or in one of the loaded shared libraries.

The asterisk "*" character can be appended to the end of a name as a wild card. In this case, the characters before the asterisk are treated as a prefix and match any call name that begins with that prefix. A value of asterisk ("*") alone matches all function names. Use this to cause the runtime to treat all names as dynamic functions first before searching the disk or memory for a COBOL program with a matching name.

The value of DYNAMIC_FUNCTION_CALLS is case insensitive. The default value is empty.

DYNAMIC_FUNCTION_CALLS can be set in the environment, configuration file, or programmatically with the SET verb. Set it to spaces to clear the list.

When DYNAMIC_FUNCTION_CALLS is set in the configuration file, there is no limit on the number of function names or overall size of the value of the configuration variable. To specify a configuration file value on multiple lines, you must prepend each line after the first with "-". For example:

```
DYNAMIC_FUNCTION_CALLS =
-                       func1,
-                       func2,
-                       func3
```

The line continuation processing removes all leading and trailing spaces so in this case you must separate the values with a comma (that is, append a comma to each line).

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

DYNAMIC_MEMORY_LIMIT

The value of this variable indicates the maximum number of bytes of dynamic memory that the ACUCOBOL-GT runtime will use to cache canceled programs when the logical cancel mechanism is enabled. When the total amount of memory exceeds the value of DYNAMIC_MEMORY_LIMIT, the runtime releases all memory held by programs that have been logically canceled.

Valid values are:

-1	(the default) no memory limit. In transaction processing systems, memory used by programs that have been logically canceled is released only by the CICS transaction, CEMT SET PROGRAM(*program_name*) NEWCOPY
0	all cancels are physical; program memory is not cached
1 to 2147483647	the maximum number of bytes of dynamic memory

A discussion of memory management and physical and logical cancels is located in **Section 6.3, "Memory Management,"** in Book 1. DYNAMIC_MEMORY_LIMIT is used in conjunction with the **LOGICAL_CANCELS** configuration variable.

ECN-3699

This variable relates to read-only entry fields. As of Version 8.1, read-only entry fields were changed to conform to standard Windows behavior in that the background color is always gray (regardless of the COBOL program's Color setting). If you need the ability to change the color of read-only entry fields, set the runtime configuration variable "ECN_3699" to "0".

The default behavior will not allow color changes and matches Windows behavior: read-only entry fields have gray backgrounds.

EDIT_MODE

This is an obsolete entry that has been replaced by the **KEYSTROKE** configuration variable. Its setting is ignored.

EF_UPPER_WIDE

This variable determines which font measure is used to compute the width of an entry field with the UPPER style. If the value is "1" (on, true, yes), the entry field is sized with the wide font measure. See section 5.9 in Book 2, *ACUCOBOL-GT User Interface Programming* for a description of how entry fields are measured. The default value of EF_UPPER_WIDE is "1".

EF_WIDE_SIZE

This variable sets the boundary size that determines whether an entry field is sized with the standard or wide font measure. An entry field that has a specified width greater than the value of EF_WIDE_SIZE is always sized with the standard font measure. Entry fields that are both non-numeric and not larger than EF_WIDE_SIZE are sized with the wide font measure. See **Section 5.9** in Book 2, *ACUCOBOL-GT User Interface Programming* for a description of how entry fields are sized. The default value of EF_WIDE_SIZE is "5". Setting this variable to "0" causes all entry fields to be sized with the standard font measure (exception: see EF_UPPER_WIDE above). Note that setting the value of this variable to a number larger than your largest entry field causes all entry fields to use the wide font measure.

EOF_ABORTS

This configuration variable can be used to handle two unexpected loop conditions:

1. a loop that results when the runtime has been started with "-i" and an input file terminates prematurely.

2. a loop that results when a terminal emulator disconnects unexpectedly.

If the runtime is started with the "-i" option and a loop occurs when an input file terminates prematurely, you can set EOF_ABORTS to "1" (on, true, yes) to cause the runtime to shut down when an ACCEPT statement detects an end-of-file condition.

On UNIX/Linux systems, if the runtime enters a loop due to an unexpected disconnect from a terminal emulator, you can set EOF_ABORTS to a value of "2" to cause the runtime to generate a hangup signal (SIGHUP) when it detects an EOF on standard input (stdin).

The default value is "0" (off, false, no).

EOL_CHAR

This configuration variable determines the character that is used to mark the end of each line when a pre-existing line sequential file is read. This should be set to the ASCII value of the desired character. The default value is "10" (line-feed). This option may be useful if you must process a line-oriented file that has an unusual line terminator. This configuration variable has no effect under VMS (RMS does not support it).

ERRORS_OK

Normally, if a file error occurs and there is no AT END, INVALID KEY, or Declarative statement to handle it, the runtime system prints an error message and halts. You can cause the runtime to ignore file errors and continue processing by setting ERRORS_OK to either "1" (on, true, yes) or "2" (FILESTATUS).

By default, ERRORS_OK is set to "0" (off, false, no).

When ERRORS_OK is set to "1", if a file error occurs the runtime continues as if no error occurred.

When ERRORS_OK is set to "2", if a file error occurs and there are no Declaratives but a file status variable is defined, the runtime ignores the error and continues processing. However, if a file error occurs and there are no Declaratives and a file status variable is *not* defined, the runtime halts.

Note: In general, it is not recommended that you configure the runtime to ignore file errors.

EXIT_CURSOR

When a STOP RUN is executed, the ACUCOBOL-GT runtime system normally places the cursor on the last line of the screen and then scrolls the screen one line. This allows the operating system prompt to appear on a new blank line at the bottom of the screen. To inhibit this behavior, set EXIT_CURSOR to "0" (off, false, no). This causes the runtime system to leave the cursor in its current location when the program exits. The default value "1" (on, true, yes) causes the standard ACUCOBOL-GT cursor positioning.

This variable has no effect on Windows systems.

EXPAND_ENV_VARS

Setting this variable to "1" (on, true, yes) causes the runtime to expand environment variables in filename specifications. This is the last step of file name interpretation process (see **section 2.9, "File Name Interpretation,"** in Book 1, *ACUCOBOL-GT User's Guide*). A file specification that includes a "$" character will have all the characters from "$" to the end of the name or to the next "/" or "\" replaced with the value of the matching environment variable. For example, if the program attempts to open "$mydir/myfile", the environment and configuration file are searched for the variable "mydir". If found, its value is substituted. If not found, the replacement is null. Referring to the preceding example, if "mydir" is not defined, the runtime attempts to open "/myfile".

The default value is "0" (off, false, no).

Note: The "$" character is a valid filename character in many file systems, including NTFS and most UNIX file systems. If you want to use dollar signs in your file names, you should not enable this option. In particular, if a user chooses the name of the file, you should keep this option disabled.

If you also use **FILE_ALIAS_PREFIX**, note that when EXPAND_ENV_VARS is set to "1", FILE_ALIAS_PREFIX treats "$FILE1" and "FILE1" the same.

EXTEND_CREATES

Setting this configuration variable to “1” (on, true, yes), causes OPEN EXTEND statements to create a new file when the file being opened is not present. The default value is “0” (off, false, no).

EXTFH_KEEP_TRAILING_SPACES

An EXTFH_KEEP_TRAILING_SPACES configuration variable allows you to preserve trailing spaces in line sequential file records when using our EXTFH module with EXTSM. Set this variable to “1” (on, true, yes) to retain the trailing spaces, which is the runtime’s default behavior. With a default value of “0” (off, false, no), trailing spaces are removed.

Note that a related configuration variable is the **STRIP_TRAILING_SPACES** variable.

EXTERNAL_SIZE

ACUCOBOL-GT manages external data items by allocating them in *pools*. The minimum size of each pool is set by the EXTERNAL_SIZE variable. When a new external data item is needed, it is allocated from an existing pool. If it doesn’t fit in any of the allocated pools, a new pool is allocated. The size of this pool is the same as the size of the data item, but never smaller than the value specified by the EXTERNAL_SIZE configuration variable. Using this larger pool reduces memory fragmentation. Because external data items remain allocated after programs are canceled, it’s best to allocate the external data items together so they don’t break up the memory space. The default value for EXTERNAL_SIZE is “8192”. The maximum value is “32767”.

EXTRA_KEYS_OK

This configuration variable allows you to open an indexed file without specifying all of that file’s alternate keys. When it is set to “1” (on, true, yes), you may open an indexed file that contains more keys than are described by

your program, and no file error will occur. However, you will still receive a file error if you open a file that does not contain all of the keys described in your program. EXTRA_KEYS_OK is useful when you are adding new alternate keys to an existing file because you do not need to rework your existing programs. This configuration variable is ignored if you use a Version 1.4 or earlier ACUCOBOL-85 object file.

The default value is “0” (off, false, no).

F10_IS_MENU

By convention, the F10 key is used by Windows and Windows NT to activate program menus. This action is controlled automatically by the program. The F10_IS_MENU configuration variable allows you to set the runtime to handle the F10 key as a user defined-key. The default setting is “1” (on, true, yes). When you change the setting to “0” (off, false, no) you inhibit the menu activation capability. For example, action of Shift-Ctl-F10 may only be defined by the user if F10_IS_MENU is set to “0”, otherwise this key combination activates context menus. This variable does not affect the behavior of the mouse. However, the mouse continues to work with the menu.

FAST_ESCAPE

This configuration variable determines how long the runtime will wait after receiving an escape key before deciding that the key is actually intended as an escape key, and *not* as the start of a function key sequence. (Increasing the number causes the runtime to wait longer.) The default setting varies with the machine. It is generally between 20 and 100.

This variable has no effect on Windows systems.

FAST_SIGN_DECODE

Version 7.3 introduced a new algorithm to decode sign bytes in USAGE DISPLAY (sign incorporated) data items. The new algorithm is 3-4 times faster than the previous one, but produces unexpected results for undefined or non-initialized numeric data items.

This configuration variable turns on this faster algorithm introduced in Version 7.3.

To use the Version 7.3 algorithm, set the FAST_SIGN_DECODE configuration variable to "TRUE". If using this algorithm, numeric data items must be defined/initialized, otherwise incorrect results may occur.

The default value is "FALSE" which means the pre-7.3 version of the algorithm is used. The pre-7.3 version does not require numeric data items to be defined/initialized.

FIELDS_UNBOXED

On most GUI systems, including Microsoft Windows, entry fields are boxed by default. This can cause problems when you are converting applications that have fairly full screen displays, because the box adds roughly 50% to the height of the field. This can make it difficult to fit all the existing fields onto the user's screen.

FIELDS_UNBOXED provides a global method of removing boxes on entry fields. If this field is set to "1" (on, true, yes), the system does not display a box around entry fields. Technically this has three effects:

1. If it is set when the entry field is initially created, the NO-BOX property is automatically implied.
2. If it is set when a floating window is initially created, the window's LABEL-OFFSET property is given a default value of "0".
3. When an entry field is measured by the CELL phrase of the DISPLAY FLOATING WINDOW statement, its height is measured without the box.

On character-based systems, setting this variable to “1” (on, true, yes) eliminates the display of the left and right delimiting symbols used in the textual emulation of entry fields. (See **GUI_CHARS** for more information about these delimiting symbols). Eliminating these symbols affects the location at which the entry fields are displayed on character-based systems.

The default value for this option is “0” (off, false, no).

This variable can be overridden for individual entry fields in the program with the BOXED style in the entry field definition.

FILE_ALIAS_PREFIX

This variable allows you to specify a list of strings to prefix to a file name before searching for that name in the configuration file or environment. Data and code file search paths are described in more detail in **Section 2.8.2** of the *ACUCOBOL-GT User’s Guide*.

When searching for a file alias:

1. The runtime constructs the file alias name by prepending the first string listed in FILE_ALIAS_PREFIX to the file name and searches for that name in the environment or configuration file.
2. If the name is not found, the runtime constructs a new name by prepending the second string in FILE_ALIAS_PREFIX to the file name and searches for that alias.

This process is repeated with each string in FILE_ALIAS_PREFIX until a file alias name is found or the end of the list is reached.

For example, with:

```
SELECT file1-name ASSIGN TO "FILE1".
```

by default, the runtime looks for a configuration or environment variable named “FILE1” and, if found, substitutes its value for the file name. If you specify:

```
FILE_ALIAS_PREFIX "":DD_
```

the runtime first looks for “FILE1” and, if not found, looks for “DD_FILE1”.

The default value of FILE_ALIAS_PREFIX is an empty string (“”). Specifying an empty string as an entry in FILE_ALIAS_PREFIX causes the runtime to search for the file name itself as an alias name. Up to 4096 characters can be specified for the value of this variable.

Note: Separate strings by one or more spaces. A space is a valid separator on all systems. On UNIX systems, you can also separate entries with a colon. On Windows systems, a semicolon can be used and on VMS systems, a comma can be used. Strings can be enclosed in quotation marks. You can specify an empty string using two consecutive quotation marks.

Note on using with EXPAND_ENV_VARS:

If you use the **EXPAND_ENV_VARS** configuration variable and the file name includes a dollar sign ($), the FILE_ALIAS_PREFIX logic is applied to the environment variable name. For example, if EXPAND_ENV_VARS is set to”1” (on, true, yes), “$FILE1” and “FILE1” are treated the same.

For example, with:

```
EXPAND_ENV_VARS=1
FILE_ALIAS_PREFIX=DD_
```

the following statement,

```
SELECT file1-name ASSIGN TO "DIR1/$DIR2/FILE1".
```

causes the runtime to search for an environment or configuration variable named “DD_DIR2” (instead of “DIR2”) and, if found, substitute its value for “$DIR2”.

FILE_CASE

This configuration variable allows you to adjust the case of data file names. Possible values include:

NONE or “0”	(the default) data file names are not translated

LOWER or “1”	data file names are translated to lower case, including directory (path) elements
UPPER or “2”	data file names are translated to upper case, including directory (path) elements
LOWER_BASE or “3”	data file names are translated to lower case, excluding directory (path) elements
UPPER_BASE or “4”	data file names are translated to upper case, excluding directory (path) elements

Translation occurs before the **FILE_PREFIX** and **FILE_SUFFIX** configuration options are applied. You should make sure that those variables specify the correct case.

File name translation does not occur if the file name starts with -F, -D, or -P. (See *ACUCOBOL-GT User’s Guide*, **Section 2.9, “File Name Interpretation”**).

FILE_CONDITION

This configuration variable can be used to alter the File Status value of an individual file status condition. We recommend that you use one of the four pre-defined file status code sets instead. If you need to change an individual status code, contact Technical Support for assistance.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

FILE_IO_PEEKS_MESSAGES

This configuration variable tells the Windows runtime to automatically call the Windows PeekMessage() API function between file operations. When the FILE_IO_PEEKS_MESSAGES configuration variable is set to “1” (on, true, yes), the runtime calls PeekMessage() with flags that tell it to simply

check for messages without removing them from the message queue. This operation tells Windows that the application is alive and responding. The default value of the variable is "0" (off, false, no).

FILE_IO_PROCESSES_MESSAGES

This configuration variable can be used to control whether the runtime processes system messages while performing file I/O operations. When it is set to "1" (on, true, yes), the runtime will process system messages while doing file I/O operations. This was the default behavior *prior* to Version 3.2. Note that the processing of system messages during file I/O should only be enabled under special conditions, as described below.

To understand when it is appropriate to set this configuration variable, it is important to be familiar with *system messages* and how the ACUCOBOL-GT runtime and your program respond to them. For the purposes of this discussion, system messages are the mechanism used by graphical systems, such as Windows, to communicate with your program. They are what the operating system uses to facilitate the communication of user and system activity to the program. They are similar to ACUCOBOL-GT's *events*. Prior to Version 3.2, the runtime automatically processed system events during file operations. This allows the user to manipulate an application window (for example, minimizing it) while file I/O operations are performed. If the application suspends the processing of system messages, the system appears to the user to be frozen.

Starting with Version 3.2, this feature is turned off by default. This is because the processing of messages outside of an ACCEPT statement can cause flaws in a program that uses multithreading or modeless windows. It also creates a state where event procedures can be called at unexpected times. In addition, the controls of the application are not actually functional, though they appear to be working to the user.

Generally speaking, setting this variable is useful only when the application does not use multithreading, modeless windows, or event procedures.

Note: The proper way to process system messages while performing other operations is to start a second thread that performs an ACCEPT statement while the main thread continues with the work. This allows the system to process messages under control of an ACCEPT, which provides a well-defined point in your program from which event procedures can be called.

FILE_PREFIX

This variable defines a set of directories that the runtime searches to locate a data file. The default value is "." (current working directory). Data and code file search paths are described in more detail in **Section 2.8.2** of the *ACUCOBOL-GT User's Guide*.

Directories can be a mix of relative and absolute paths. Entries are separated by one or more spaces. A space is a valid separator on all systems. Alternatively, on UNIX systems you can also separate entries with a colon. On Windows systems a semicolon can be used. On VMS systems a comma can be used.

You can specify a directory path that contains embedded spaces if you surround the path with quotation marks. For example:

```
FILE_PREFIX  C:\"Sales Data"  C:\"Customers"
```

Remote name notation is allowed for the FILE_PREFIX variable if your runtime is client-enabled (for indexed files, remote name notation requires the Vision file system). See *ACUCOBOL-GT User's Guide* **Section 5.2.1** and **Section 5.2.2** for more information about client-enabled runtimes and remote name notation.

Up to 4096 characters can be specified for the value of this variable.

FILE_STATUS_CODES

This variable determines which set of file status codes to use. For details, see the *ACUCOBOL-GT User's Guide*, **section 2.8.3, "File Status Codes."** The default value is "85".

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

FILE_SUFFIX

The value of this variable is automatically appended to data file names that do not contain an explicit suffix. A suffix is the portion of a file name that follows a period. For example, if FILE_SUFFIX is set to "DAT", then opening a file called "EMPFILE" would actually open the file called "EMPFILE.DAT". The default value is empty.

FILE_TRACE

This variable allows you to start file tracing without opening the debugger. Set this variable to a non-zero value to save information about all file OPENs, READs, and WRITES in the error file. This is equivalent to specifying "tf *n*" from the debugger (where *n* is an integer). The default is "0." See **section 3.1.4, "File Tracing,"** of the *ACUCOBOL-GT User's Guide* for more information about the file trace feature.

FILE_TRACE_FLUSH

Set this variable to "1" (on, true, yes) to flush the error file after every WRITE statement. This is equivalent to using "t flush" from the debugger. The default is "0" (off, false, no). See **section 3.1.4, "File Tracing,"** of the *ACUCOBOL-GT User's Guide* for more information about the file trace feature.

FILE_TRACE_TIMESTAMP

Set this variable to “1” (on, true, yes) to cause file trace timestamp information to be recorded in the error file. When this variable is enabled, a timestamp is placed at the beginning of every line in the trace file. The format of the timestamp is: HH:MM:SS.mmmmmm, where “mmmmmm” is the finest resolution that the runtime can obtain from the system. The default setting of this variable is “0” (off, false, no).

Timestamp information is included only when file trace information is directed to a file. Timestamp output can add significant I/O overhead and may have a noticeable impact on performance.

filename

This configuration variable allows you to map Vision 4 and 5 files to a different directory. Vision examines the name of each physical file it attempts to open to determine if the file should be mapped to a different directory. The configuration variable used is constructed from the file’s base name and extension, with all letters converted to upper case and all non-alphanumeric characters converted to underscores.

For example, assume you open “/usr/data/custfile.dat”, and a configuration variable “CUSTFILE_DAT” has the value “/usr2/data/custfile.dat”. Vision treats this value as the actual file name to open, and “custfile.dat” ends up in the “/usr2/data” directory rather than “/usr/data”.

Because the extension is included in the configuration variable name, you can place different parts of a multi-segment file in different directories. If no name is found for a particular segment, then the segment name is used unchanged. Note that you can move parts of a file around by simply moving the segment and adding/modifying its corresponding configuration name. Name mapping is done directly by Vision (as opposed to, for example, FILE_PREFIX, which is handled by the runtime). As a result, all programs that use Vision (such as **vutil** and **vio**) use this variable when present. For programs other than the runtime, the variable must be set in the environment rather than the configuration file.

Two configuration variables can affect the value of this variable. They are: **V_BASENAME_TRANSLATION** and **V_STRIP_DOT_EXTENSION**.

This variable is similar to the **filename_VERSION** configuration variable.

Note: The filename translation performed by this configuration variable is performed by Vision itself. The runtime can also perform filename translation. See Book 1, *ACUCOBOL-GT User's Guide*,**Section 2.8.1** for more information.

---

*filename*_DATA_FMT

This configuration variable specifies a format for naming the data segments of Vision 4 and 5 files. (See **filename_INDEX_FMT** for details about naming the index segments, as both variables should be set to corresponding patterns). The configuration variable used is constructed from the file's base name and extension, with all letters converted to upper case and all non-alphanumeric characters converted to underscores, followed by the "_DATA_FMT" string. Note that by design, this variable does not modify the first specified data segment. The first data segment retains the originally specified name. The filenames of the additional segments of a Vision file are generated from the name of the initial data segment. The *filename*_DATA_FMT variable allows you to change the way the names of the following data segments are formed, but the names still originate from the name of the initial data segment. As long as the names are as expected (and you have set filename_DATA_FMT and *filename*_INDEX_FMT accordingly) the segments will be found properly.

Suppose that the regular name of your COBOL file is "/usr1/gl.dat". The variable you would use to set the data segment naming format for this file is GL_DAT_DATA_FMT.

The variable must be set equal to a pattern that shows how to create the segment names. The pattern shows how to form the base name and extension for each segment. Part of this pattern is a special escape sequence (such as %d) that specifies how the segment number should be represented. Choices include %d (decimal segment numbers), %x (lowercase hexadecimal numbers), %X (uppercase hexadecimal numbers), and %o (octal numbers).

For example, setting the variable GL_DAT_DATA_FMT=gl%d.dat would result in data segments named /usr1/gl.dat (remember that the first data segment is not affected), /usr1/gl1.dat, /usr1/gl2.dat, and so forth.

Escape sequence definitions:

The %d in the value of the *filename*_DATA_FMT above is a printf-style escape sequence. Most reference books on the C language contain an in-depth explanation of these escape sequences, and UNIX systems typically have a man page ("man printf") that explains them in detail. Here are the basics:

- "%d" expands into the decimal representation of the segment number.
- "%x" expands into the hexadecimal representation (with lower case a-f) of the segment number.
- "%X" expands into the hexadecimal representation (with upper case A-F) of the segment number.
- "%o" expands into the octal representation of the segment number.
- You can add leading zeros to the number (to keep all the file names the same length) by placing a zero and a length digit between the percent sign and the following character. "%02d" would result in "00", "01", "02", and so forth, when expanded.
- To embed a literal "%" in the file name, use "%%".

The escape sequence can be positioned anywhere in the file name, including the extension.

Note: While the runtime checks for this segment naming variable in the runtime configuration file as well as in the environment, utilities such as **vutil** and **vio** check only the environment. Therefore, if you are using this variable with the runtime and **vio** or **vutil**, you must set the variable in the environment and not in the configuration file.

Two configuration variables affect the value of this variable: **V_BASENAME_TRANSLATION** and **V_STRIP_DOT_EXTENSION**.

Note: The filename translation performed by this configuration variable is performed by Vision itself. The runtime can also perform filename translation. See Book 1, *ACUCOBOL-GT User's Guide*, **Section 2.8.1** for more information.

*filename*_FILESYSTEM

*filename*_FILESYSTEM is a synonym for *filename*_HOST. See the entry for **filename_HOST**.

*filename*_HOST

Note: *filename*_FILESYSTEM is a synonym for *filename*_HOST.

If the program opens an existing file or creates a new one, you can tell the runtime the file system to use with that file. The Vision file interface is used by default. You specify the file system with one of two configuration variables: DEFAULT_FILESYSTEM, or *filename*_HOST (syn. *filename*_FILESYSTEM). DEFAULT_FILESYSTEM specifies the default file system for all files (see the entry for **DEFAULT_FILESYSTEM**). *filename*_HOST specifies the file system for an individual file. For example,

```
filename_HOST filesystem
```

assigns the specified data file to the named file system. Any file so assigned uses the designated file system and not the one specified by DEFAULT_FILESYSTEM. *Filename* must be the base name of the file and cannot include the path or the file extension (any part of the name that follows the first dot (“.”). For example, to specify that data file “IDX1.DAT” be handled by the EXTFH interface, you would specify:

```
IDX1_HOST EXTFH
```

or

```
IDX1_FILESYSTEM EXTFH
```

You must specify only the base name of the file in *filename*.

To specify that the data file “DXML1.DAT” be handled by the XML interface, you could specify:

```
DXML1_HOST XML
```

Note that XML can be specified only with sequential files.

*filename*_INDEX_FMT

This configuration variable specifies a format for naming the index segments of Vision 4 and 5 files. (See **filename_DATA_FMT** for details about naming the data segments, as both variables should be set to corresponding patterns). The configuration variable used is constructed from the file’s base name and extension, with all letters converted to upper case and all non-alphanumeric characters converted to underscores, followed by the “_INDEX_FMT” string. Note that by design, this variable does not modify the first specified index segment. The first index segment retains the originally specified name. The filenames of the additional segments of a Vision file are generated from the name of the initial index segment. The *filename*_INDEX_FMT variable allows you to change the way the names of the following data segments are formed, but the names still originate from the name of the initial index segment. As long as the names are as expected (and you have set **filename_DATA_FMT** and **filename_INDEX_FMT** accordingly) the segments will be found properly.

Suppose that the regular name of your COBOL file is “/usr1/gl.dat”. The variable you would use to set the format for naming the file’s index segments is GL_DAT_INDEX_FMT.

The variable must be set equal to a pattern that shows how to create the segment names. The pattern shows how to form the base name and how to form the extension for each segment. Part of this pattern is a special character (such as %d) that specifies how the segment number should be represented. Choices include %d (decimal segment numbers), %x (lowercase hexadecimal numbers), %X (uppercase hexadecimal numbers), and %o (octal numbers).

For example, setting the variable GL_DAT_INDEX_FMT=gl%d.idx would result in index segments named /usr1/gl0.idx, /usr1/gl1.idx, /usr1/gl2.idx, and so forth.

Escape sequence definitions:

The %d in the value of the *filename*_INDEX_FMT above is a printf-style escape sequence. Most reference books on the C language contain an in-depth explanation of these escape sequences, and UNIX systems typically have a man page ("man printf") that explains them in detail. Here are the basics:

- "%d" expands into the decimal representation of the segment number.
- "%x" expands into the hexadecimal representation (with lower case a-f) of the segment number.
- "%X" expands into the hexadecimal representation (with upper case A-F) of the segment number.
- "%o" expands into the octal representation of the segment number.
- You can add leading zeros to the number (to keep all the file names the same length) by placing a zero and a length digit between the percent sign and the following character. "%02d" would result in "00", "01", "02", and so forth when expanded.
- To embed a literal "%" in the file name, use "%%".

The escape sequence can be positioned anywhere in the file name, including the extension.

Note: While the runtime checks for this segment naming variable in the runtime configuration file as well as in the environment, utilities such as **vutil** and **vio** check only the environment. Therefore, if you are using this variable with the runtime and **vio** or **vutil**, you must set the variable in the environment and not in the configuration file.

Two configuration variables affect the value of this variable: **V_BASENAME_TRANSLATION** and **V_STRIP_DOT_EXTENSION**.

Note: The filename translation performed by this configuration variable is performed by Vision itself. The runtime can also perform filename translation. See Book 1, *ACUCOBOL-GT User's Guide*, **Section 2.8.1** for more information.

*filename*_LOG

This configuration variable specifies individual log files to be used by the transaction logging system. The format of the variable is:

```
filename_LOG logfilename
```

where *filename* is the base name of the data file, and *logfilename* is the name of the log file. *filename* should not include any directory names nor a file extension. *logfilename* can include the absolute or relative directory path ending with the name of the log file. If the log file is not found, a new file is created with the specified name. Note that *logfilename* can have remote name notation.

FILENAME_SPACES

When this configuration variable is set to "1" (on, true, yes), filenames may contain embedded spaces and the runtime considers the last non-space character as the end of the name. The default is "1". When this configuration variable is set to "0" (off, false, no), then filenames may not contain embedded spaces and the name terminates at the first space character. For example:

```
C:\temp dir\my file name
```

is read as:

```
C:\temp
```

This affects the behavior of the library routines that take a filename as an argument:

C$CHDIR
C$COPY

C$DELETE
C$FILEINFO
C$FULLNAME
C$MAKEDIR
C$RESOURCE
CBL_COPY_FILE
CBL_CREATE_DIR
CBL_DELETE_DIR
CBL_DELETE_FILE
I$IO
RENAME
W$BITMAP
W$KEYBUF
$WINHELP
WIN$PLAYSOUND

*filename*_VERSION

This configuration variable sets the Vision file format on a file-by-file basis. The *filename* is replaced by the base name of the file (the filename minus directory and extension). The meaning of the variable is the same as for "**V_VERSION**". This variable is useful if you want to have all your Vision files in one format, with a few exceptions. For example, you might want to maintain most of your files in Vision Version 3 format to conserve file handles, but have a few files in Version 5 format to take advantage of the larger file size. Note that this variable only affects the file format when it is created. You can always rebuild the file in another format later.

This variable (and the "V_VERSION" variable) is most helpful when you are using transaction management. The transaction system does not record the format of the created file if an OPEN OUTPUT is done during a transaction, because the transaction system is not tied to any particular file system. If you need to recover a transaction, the system will recreate the OPEN OUTPUT files using the settings of the "VERSION" variables.

The behavior of this variable is affected by the settings of the configuration variables **V_STRIP_DOT_EXTENSION** and **V_BASENAME_TRANSLATION**.

- If V_STRIP_DOT_EXTENSION is set to “0” (off, false, no), Vision does not remove any dot extension when replacing the base name of the file. This can be useful if you have two files that share a common name before their dot extension.
- If V_BASENAME_TRANSLATION is set to “0” (off, false, no), Vision includes the entire path of the file in the base name. This can be useful if you have files with the same names stored in different directories.

*filesystem*_DETACH

This configuration variable detaches any file system from the runtime. The syntax is:

```
filesystem_DETACH n
```

where *filesystem* corresponds to the first five letters of the file system name and *n* is a non-zero value. Examples of file systems that may be detached using this feature are:

Btrieve	BTRIE_DETACH
C-ISAM	C_ISA_DETACH
Informix	INFOR_DETACH
SQL Server	MSSQL_DETACH
ODBC	ODBC_DETACH
Oracle	ORACL_DETACH
RMS	RMS_DETACH
Sybase	SYBAS_DETACH
Vision	VISIO_DETACH

The file systems may be detached only when the runtime is started, not during execution. If you detach *all* file systems, the runtime will terminate with an error message. For example, if you detach Vision with VISIO_DETACH on a standard runtime, the runtime will terminate with this message to std err: No file system available.

This feature automatically supports new file systems added to the runtime.

FLUSH_ALL

This configuration variable can be used to control the flushing of file buffers to disk. It is one of several variables that control buffer flushing. See the other entries in this appendix that begin with “FLUSH”.

This variable can take a combination of the following values:

1 (on, true, yes, all)
0 (off, false, no)
MASS_UPDATE
REMOTE

When this variable is set to “1”, files opened for MASS-UPDATE are flushed along with other files. This means that the local cache used to hold the MASS-UPDATE buffers is flushed whenever the operating system cache is flushed.

When this variable is set to the default value of “0”, files opened for MASS-UPDATE are not flushed.

Setting this variable to MASS_UPDATE causes the runtime to flush local files, including files opened with MASS-UPDATE.

Setting this variable to REMOTE causes the runtime to flush local files not opened with MASS-UPDATE, as well as remote files.

You can also set this variable to a combination of values. For example,

```
FLUSH_ALL   MASS_UPDATE   REMOTE
```

causes the runtime to flush all local files, including those opened with MASS-UPDATE, as well as remote files.

Note on bitmask integer values:

Internally, the value of this configuration variable is converted to a bitmask, and its bitmask integer value is determined by the keywords used to set it. Keywords translate into integer values as follows:

FALSE, NO and OFF are equivalent to “0”

MASS_UPDATE is equivalent to “1”

REMOTE is equivalent to “2”

ALL, TRUE, YES and ON are equivalent to “–1”

When the runtime is started with the “-l” and “-e *errfile*” arguments, only the integer value of FLUSH_ALL is recorded in the error file.

FLUSH_COUNT

This configuration variable allows you to flush the disk buffers after a certain number of file updates has occurred. For example, if you set this configuration variable to “10”, then the buffers will be flushed after every ten updates to disk files. Only indexed files are counted. When the buffers are flushed, the exact action depends on the operating system:

Windows	Buffers are written to disk and the file’s directory information is updated. This is roughly equivalent to the action that occurs when a file is closed.
UNIX	The “sync” system routine is called. This causes all of UNIX’s cache to be written to disk. This operation is only scheduled--it occurs when the system finds time to do it. Because the system does this every 30 seconds anyway, probably the only reason to request a call to “sync” is if you have unreliable power.
VMS	VMS does not have a system cache, so this configuration variable has no effect.

Note: Setting this variable to a low non-zero value will improve the chances of recovering a file after a power failure, but will decrease performance. If FLUSH_COUNT is set to “0”, then the system buffers are flushed only when a file is closed. The default setting is “0”.

FLUSH_ON_ACCEPT

This configuration variable causes the system's buffers to be flushed whenever a Format 1 or Format 2 ACCEPT statement is executed. You turn on this configuration variable by setting its value to "1" (on, true, yes). The default setting is "0" (off, false, no). Note that this variable is not recommended for multi-user systems because of the performance penalty.

FLUSH_ON_CLOSE

This configuration variable applies only to Windows systems. When this variable is set to "1" (on, true, yes), the cache buffers will be flushed to disk when the file is closed. Versions prior to 4.3.1 flushed the cache buffers for safety reasons. This, however, reduced system performance, significantly in some programs. This feature is turned "off" by default.

FLUSH_ON_COMMIT

When this configuration variable is set to "0" (off, false, no), the COMMIT verb will not request the host operating system to flush all buffers to disk.

If flushing is prevented, COMMIT and UNLOCK ALL have the same effect. The default setting is "0" (off, false, no).

FLUSH_ON_OPEN

This variable causes the system's buffers to be flushed on the first WRITE, REWRITE, or DELETE that occurs after an indexed file is opened for I/O. The purpose of this is to update the "user count" field in the file's header to keep it accurate. When this configuration variable is set to "1" (on, true, yes), this feature is turned on; when set to "0" (off, false, no), it's turned off. The default setting is "0".

FONT

This variable has meaning only on graphical systems such as Windows. You determine the font used for accepting and displaying data on screen by setting the configuration variable FONT to one of these values:

1	Use the graphical font (default). The character set for this font is referred to as the "ANSI" set.
2	Use the "OEM" character set (MS-DOS font).

This setting must be made in the configuration file before you execute the program. Altering the value of FONT from inside your program has no effect.

By default, data is stored on disk using the same character set that was used when the data was entered. Thus, if you use the graphical font to accept data, that data is stored in the ANSI character set. If you use the MS-DOS font, then data is stored in the OEM character set.

Data moved into a graphical environment from MS-DOS applications was originally stored in the OEM character set. What happens if you now choose the graphical font? As long as your application uses only standard ASCII characters, the underlying representation is the same, and so the data is completely interchangeable. See the variable **TRANSLATE_TO_ANSI** if you are using non-ASCII characters.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

FONT_AUTO_ADJUST

This variable allows you to disable an automatic font adjustment that is applied on Windows machines. The runtime attempts to adjust automatically for differences in the relative proportions between "small fonts" and "large fonts." You can inhibit the adjustment by setting this variable to "0" (off, false, no).

The adjustment is provided because the internal scaling of fonts under Windows changes between “small” and “large” fonts. Under small fonts, digits are slightly wider than the average character. Under large fonts, digits have the same width as the average character. ACUCOBOL-GT uses the size of the font’s width for many calculations. Thus, the change in relative proportion within a single font can cause problems for screens designed for the small fonts. For example, a frame may not be big enough to hold its contents. To prevent this problem, the runtime ensures that the “standard font measure” is always a bit larger than the average character width in a font. To disable this adjustment, set this variable to “0”.

Note: This variable computes the width of a printer font in the same way that the width of a screen font is computed. You can suppress this behavior by compiling with the “-C43” option.

FONT_SIZE_ADJUST

This variable allows you to adjust the size of the standard font measurement that is computed for graphical controls (applies to variable-pitch fonts only). The value of FONT_SIZE_ADJUST is added directly to the computed standard font size. For example, a setting of “1” adds one pixel to the computed width of the font. Because the standard font measure is used to compute the width of nearly all controls, any adjustment made by this variable will have a significant impact on the layout of your screens.

The adjustment to the standard font measure is made after the wide font measure is computed (this is important to note because the wide font measure depends on the standard font measure; to change the wide font measure, use the **FONT_WIDE_SIZE_ADJUST** configuration variable).

After applying the adjustment, the runtime checks and ensures that the computed font measure is not less than one (1) or greater than the widest character in the font. If you find that the default size of most controls is slightly smaller than you prefer, you might try setting FONT_SIZE_ADJUST to a small value (typically 1).

Generally, it is recommended that FONT_SIZE_ADJUST (and FONT_WIDE_SIZE_ADJUST) be left at its default value of “0”. You can also use negative values, but there is rarely a need to do so.

To optimize performance, the runtime computes the font sizes only occasionally. Although you can change the variable dynamically at runtime, the exact time when the new setting will take effect is difficult to predict. For this reason, we recommend that you either set it in your program prior to constructing your initial screen, or directly in the configuration file.

Note: This variable computes the width of a printer font in the same way that the width of a screen font is computed. You can suppress this behavior by compiling with the "-C43" option.

FONT_WIDE_SIZE_ADJUST

This variable allows you to adjust the size of the wide font measurement (applies to variable-pitch fonts only). The wide font measure is normally used when the runtime is measuring small or upper-case entry fields. The value of FONT_WIDE_SIZE_ADJUST is added directly to the computed wide font size.

After applying the adjustment, the runtime checks and ensures that the computed wide font measure is not smaller than the (adjusted) standard font measure or larger than the widest character in the font. If your upper-case fields are not quite as wide as you prefer, try setting this variable to a small value (typically 1 or 2).

Generally, it is recommended that FONT_WIDE_SIZE_ADJUST (and **FONT_SIZE_ADJUST**) be left at its default value of "0". You can also use negative values, but there is rarely a need to do so.

Note: In order to improve performance, the runtime computes the font sizes only occasionally. Although you can change the variable dynamically at runtime, the exact time when the new setting will take effect is difficult to predict. For this reason, we recommend that you either set it in your program prior to constructing your initial screen, or directly in the configuration file.

FOREGROUND_INTENSITY

Use this variable to set the default foreground intensity.

0	The runtime uses the default intensity for the output device. For Windows the default is low-intensity.
1	The runtime uses low-intensity.
2	The runtime uses high-intensity.

If your program specifies a default intensity, then the runtime will never assign high-intensity if the foreground is black. As with the background, we do this to prevent a washed-out appearance. There's one exception to this rule. The runtime will assign high-intensity to a black foreground if the output device does not support independent background intensities. In this case, the device will typically show the background in high-intensity and keep the foreground black. Note that if your program explicitly sets high-intensity, then that will be used regardless of the foreground color. The default value for this variable is "0".

FREEZE_AX_EVENTS

This configuration variable applies only in a thin client environment. During the processing of an ActiveX event, the Windows and thin client runtimes attempt to suspend subsequent ActiveX events until the first event has completed. By default, the thin client runtime also attempts to suspend ActiveX events whenever the application is not processing an ACCEPT statement. To suspend and resume events, the runtime calls the ActiveX function IOleControl::FreezeEvents().

You might want to disable calls to "FreezeEvents" for ActiveX controls that discard events while in a "FreezeEvents" state. For example, if a user double-clicks in an ActiveX control, the control might generate three events: mouse-down, mouse-up, and double-click. If the COBOL program terminates an ACCEPT statement in response to the mouse-down event, the runtime calls FreezeEvents(), and the ActiveX control might discard the mouse-up and double-click events.

You can disable the FreezeEvents() logic by setting the FREEZE_AX_EVENTS runtime configuration variable to “0” (off, false, no) in the configuration file or programmatically with the SET verb. The default value of FREEZE_AX_EVENTS is “1” (on, true, yes).

Note: The FreezeEvents() logic protects against unexpected nesting of ActiveX events and against event procedures running unexpectedly during a CREATE, DISPLAY, MODIFY, INQUIRE, or other operation that waits for results from the thin client. Turning this feature off can cause unexpected behavior.

For more information about ActiveX controls in a thin client environment, refer to the *AcuConnect User’s Guide*.

FULL_BOXES

This variable applies only in text-mode environments. When FULL_BOXES is set to “1” (on, true, yes) a full, four-sided box is drawn around *boxed* entry fields. By default, to save screen space on character-based systems, only the left and right edges of a box are drawn around boxed entry fields. The default value of FULL_BOXES is “0” (off, false, no).

Note: This variable requires that the boxed entry field be defined as MULTILINE.

GRID_BUTTONS_CAUSE_GOTO

This variable applies to graphical programs that include one or more paged grid controls. When GRID_BUTTONS_CAUSE_GOTO is set to “1” (on, true, yes) and a user clicks a scroll button on the side of a paged grid, the runtime checks to see if the grid has focus. If the grid does not have focus, the runtime gives the grid the focus, generating any events that result normally from that focus change. This usually means that a CMD-GOTO event is sent to the COBOL program. The default value for this variable is “0” (off, false, no).

GRID_NO_CELL_DRAG

This variable applies to graphical programs that include grid controls and sets the NO-CELL-DRAG style as the default behavior for all grid controls, as opposed to specifying NO-CELL-DRAG style individually for each grid control. To configure the NO-CELL-DRAG style as the default setting, set the GRID_NO_CELL_DRAG configuration variable to "1" (on, true, yes). The default value is "0" (off, false, no) and will enable the user to drag a cell in a GRID control unless you specify a NO-CELL-DRAG style on that particular grid control.

GUI_CHARS

This variable configures a character-based host runtime to substitute some specific characters when it is performing textual emulation of graphical controls. If the GF-GUI-MAP terminal database entry doesn't exist or has the character "0" (zero) in a particular character's position, the runtime examines the GUI_CHARS variable to determine what character to display. GUI_CHARS is used only by character-based host runtimes.

The value of GUI_CHARS is a list of 14 space delimited characters strictly ordered to correspond with the following graphical elements (defaults are in parentheses):

1.	System menu button	(*)
2.	Title left corner	(+)
3.	Title right corner	(+)
4.	Title fill character	(=)
5.	Minimizer	(.)
6.	Maximizer	(^)
7.	Scroll bar up button	(^)
8.	Scroll bar down button	(v)
9.	Scroll bar left button	(<)
10.	Scroll bar right button	(>)
11.	Scroll bar page area	( )

12.	Scroll bar slider	(#)
13.	Left entry field box	([)
14.	Right entry field box	(])

The characters may be specified in ASCII or decimal form. To specify an ASCII value, precede the value with a space. For example, to use "-" in place of "=" for the title fill character, make the following entry in the runtime configuration file:

```
GUI_CHARS  0 0 0 - 0 0 0 0 0 0 0 0 0 0
```

or

```
GUI_CHARS  0 0 0 -
```

The presence of a "0" (zero) following a space, causes the default character to be used.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

HELP_PROGRAM

If the program uses help automation (see Book 2, *ACUCOBOL-GT User Interface Programming*, **Chapter 10: Help Automation**), this variable should be assigned the name of the help processor program. The help processor's entry point is always a COBOL program. The program can be the help processor itself, or a shell to some other help processor, such as Windows Help. The default value of HELP_PROGRAM is "AcuHelp".

HINTS_OFF

Controls how long the pop-up hint is displayed before being erased. See **Section 3.7.4** in Book 2, *ACUCOBOL-GT User Interface Programming* for a description of pop-up hints. Set this variable to the number of hundredths

of seconds to display the hint. The default value is “400” (four seconds). If you set this value to “0”, the hint will remain displayed until some other event (such as using the button or typing) causes it to disappear.

HINTS_ON

Controls how long the mouse must remain stationary over a bitmap button before displaying its pop-up hint. For a description of pop-up hints, see **Section 3.7.4** in Book 2, *ACUCOBOL-GT User Interface Programming*. Set this variable to the number of hundredths of a second that the mouse must be stationary. The default value is “75” (three-quarters of a second). If you set this variable to “0”, pop-up hints will not be displayed. Setting this variable to “1” or “2” is not recommended because it may result in the pop-up hint being displayed while the mouse is moving across the button face (because the mouse motion events may occur less than 0.02 seconds apart).

HOT_KEY

ACUCOBOL-GT offers two methods for assigning hot keys--the HOT_KEY variable, described here, and the KEYSTROKE hot-key format described in the *ACUCOBOL-GT User’s Guide*, **Section 4.3.2.2**.

Using the HOT_KEY variable described below, you can easily assign a whole range of keys to a single hot-key program and determine which key activated the program. This lets you write a single program that handles an entire menu. Each menu item can act as a “hot key” to call this program.

This HOT_KEY format differs from the KEYSTROKE hot-key described in the *User’s Guide* in three ways:

- You assign a hot key by referencing its *exception value* instead of referencing its *key code*. Thus, if you assign the same exception value to several individual keys, you can associate these keys with the same hot-key program by making one COBOL configuration file entry.

  Similarly, menu items and individual keys can be assigned the same exception value, and then associated with the same hot-key program in a single configuration file entry.

- You may assign a range of exception values to activate the same program. You could use this to write a menu handler by assigning all of your menu items to a unique range and then assigning that range to a single hot-key program.

- A hot-key program activated using the HOT_KEY format is passed an additional parameter. This third parameter contains the value of the exception key that activated the program. This is passed as a COMP-1 data item.

Use this variable to associate an exception value, or range of values, with a program. HOT_KEY has the following format:

```
HOT_KEY  program = value1 [, value2]
```

where program is the name of the program to run, value1 is the lower (or only) exception value that activates the program, and value2 is the upper value of the activation range. Value2 may be omitted; if it's used it must include the separating comma. You must place program in single or double quotes if you require a lower-case program name.

For example, to assign a program called "mymenu" to exception values 100 through 200, use the following entry:

```
HOT_KEY  "mymenu" = 100, 200
```

A special exception value named TIMEOUT may be specified as the first exception value. When this value is used as the first exception value for a HOT_KEY program, the runtime will execute the named program whenever an ACCEPT BEFORE TIME times out. When that occurs, the second exception value is ignored.

Remote name notation is allowed for the HOT_KEY variable if your runtime is client-enabled. See *ACUCOBOL-GT User's Guide* **Section 5.2.1** and **Section 5.2.2** for more information about client-enabled runtimes and remote name notation.

Multiple HOT_KEY entries may reference the same program. This allows you to specify noncontiguous activation ranges. (Be aware that no more than 16 hot-key entries can be included in the COBOL configuration file. Using a contiguous range of exception values assigns many keys while counting as only one entry towards the limit.)

If you specify a *value1* value of "0", then all hot-key references to *program* are removed. Within a given run unit, this is the only way to remove the assignment of an exception value to a hot-key program after it has been assigned. You will probably use SET ENVIRONMENT in your source code to do this.

If you assign multiple hot-key programs to the same exception value, the results are undefined.

You may assign different hot keys using both the HOT_KEY variable, described here, and the KEYSTROKE hot-key format described in the *ACUCOBOL-GT User's Guide*, **Section 4.3.2.2**. The results are undefined if you assign the same key using both formats. The total number of hot-key entries defined by both methods cannot exceed 16.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

HP_TERMINAL_ATTRIBUTE_HANDLING

When set to "1" (on, true, yes), if the previous character was written to a line other than the current one this variable causes the runtime to set the attribute for the character to be written even if the attribute has not changed. When set to "0" (off, false, no), this variable causes the runtime to behave as it always has. For example, if the terminal attribute has not changed then it will not be set again. The default setting is "0" (off, false, no).

HTML_TEMPLATE_PREFIX

This variable is used to specify a series of directories for locating HTML template files. This variable is similar to FILE_PREFIX and CODE_PREFIX. It specifies a series of one or more directories to be searched for the desired HTML template file. The directories are specified as a sequence of space-delimited prefixes to be applied to the file name. All directories in the sequence must be valid names. The current directory can be indicated by a period (regardless of the host operating system). This is the default.

Note: Remote name notation is *not* allowed for the HTML_TEMPLATE_PREFIX variable, even if your runtime is client-enabled.

ICOBOL_FILE_SEMANTICS

This variable affects the behavior of indexed and relative files when reversing direction after reading past the beginning or end of a file. Normally, if you perform a series of READ NEXTs that reach to the end of the file (returning file status "10"), a subsequent READ PREVIOUS will return the last record in the file. The file pointer's position after each READ NEXT is just past the end of the last record. Similarly, reading past the beginning of the file and then doing a READ NEXT will return the first record in the file.

Under ICOBOL, these conditions produce different results. The record returned by the READ PREVIOUS is the second-to-last record in the file, and the record returned by the READ NEXT is the second record in the file. Essentially, when a series of READs passes either end of the file, the record pointer remains on the first or last record.

Setting ICOBOL_FILE_SEMANTICS to "1" (on, true, yes) will cause the runtime to emulate ICOBOL's handling. This is useful when porting ICOBOL programs to ACUCOBOL-GT. This option is effective only in programs that have been compiled for ACUCOBOL-85 2.0, or later. The default value is "0" (off, false, no).

ICON

This variable has meaning only on graphical systems such as Windows. Use this variable to designate a program's *minimized* icon. (By default, it uses the non-debugger icon provided with ACUCOBOL-GT.) Set ICON to the name of the file that contains the icon. This file should be an ".ICO" file that contains a 16-color icon. (Although the file may contain icons in other

formats, the runtime accesses only the 16-color icon.) The icon should be created using an icon editor like the one in the Windows Software Development Kit.

For example, if your custom icon were contained in the file "PAYROLL.ICO" in the directory \ACCT\ICONS, you would add the following line to your COBOL configuration file:

```
ICON   \ACCT\ICONS\PAYROLL.ICO
```

The maximum icon size that can be used with the ICON variable is 32 x 32 bits. The runtime uses the first icon it finds in the icon file that fits its maximum. If that icon is larger than the maximum, a memory access violation may occur.

Note: The ICON configuration variable determines the icon used only when your application is minimized. It does not determine the icon displayed by the Program Manager.

IMPORT_USES_CELL_SIZE

This variable is used when importing graphical screens into the AcuBench Screen Designer using the screen import utility. IMPORT_USES_CELL_SIZE allows you to choose whether fields are measured using the actual cell size of the imported screen or measured in 10-pixel by 10-pixel cells. The runtime checks this variable only if you are importing screens. When IMPORT_USES_CELL_SIZE is set to the default value of "1" (on, true, yes), the screen import utility captures the actual cell size used to create windows. If this variable is set to "0" (off, false, no), the screen import utility outputs information based on 10-pixel by 10-pixel cells. Note that there is no need to set this variable when importing character screens, which should always import with a cell size of 10-pixel by 10-pixel cells. See the AcuBench documentation for more information on importing screens.

INACTIVE_BORDER_COLOR

This variable is used on character-based hosts to specify the color and video attributes of the characters that form the border (box) around an inactive floating window. INACTIVE_BORDER_COLOR can be set to a variety of numeric values that express combinations of color and video attributes. See the documentation for the COLOR phrase in the "Common Screen Options" section of the *ACUCOBOL-GT Reference Manual* (**Section 6.4.9**).

If INACTIVE_BORDER_COLOR is set to "0", the inactive window's border is drawn with the colors and video attributes specified in the COBOL program when the window is first created. The default value is "0".

INCLUDE_PGM_INFO

This variable causes additional program information to be added to the string passed to COBOL error procedures.

When INCLUDE_PGM_INFO is set to "1" (on, true, yes) the string passed to COBOL error procedures is prepended with the current program name and the address of the program failure. The address may not be exactly the same as that in the COBOL listing, but it will be very close. (The given address is the actual current program counter, which is typically slightly advanced from the line on which the fault occurred.) When INCLUDE_PGM_INFO is set to the default, "0" (off, false, no), the string contains only the text of the intermediate error message.

For more about COBOL error procedures, see the entry for the **CBL_ERROR_PROC** in Appendix I.

INPUT_STATUS_DEFAULT

The value of this variable is returned when an ACCEPT FROM INPUT STATUS statement is executed on a machine that cannot determine the input status of the terminal. This can be used to make a running program behave correctly on a new machine. The value must be a single digit. The default value is "0".

If the input is redirected (not attached to a terminal), the **SCRIPT_STATUS** configuration variable determines whether ACCEPT FROM INPUT STATUS returns the value of INPUT_STATUS_DEFAULT or returns the actual status of the input file.

INSERT_MODE

This variable determines whether or not keystrokes are inserted in front of any existing text when the user types an entry. Set this variable to "1" (on, true, yes) to enable insertion. The default value is "0" (off, false, no), which causes typing to replace existing text.

The user can change the state of INSERT_MODE with various key actions. For example, pressing <insert> can enable insertion. This variable has no effect on Windows controls.

INTENSITY_FLAGS

This variable takes effect only if you use **COLOR_TRANS** and only if it changes your color scheme. After any color transformation is completed, the runtime system then transforms the foreground and background intensities according to the setting of INTENSITY_FLAGS. The value for this variable is actually the sum of the values you choose from the list below. The default value is "0".

Set INTENSITY_FLAGS to a combination of the following options by adding their values together:

1	Exchanges the foreground and background intensities for each other. This is useful if you are swapping a black background into the foreground and want to assign the foreground's intensity to the background.
2	Causes the foreground intensity to be inverted. That is, if the foreground is high-intensity, it becomes low-intensity. Otherwise, it becomes high-intensity. This is useful if you are transforming the background to white and the foreground to black. Setting this will cause your low-intensity foreground to be shown as gray while your high-intensity item will show as black.
4	Forces the foreground to high-intensity. This will not be applied to a black foreground.
8	Forces the foreground to low-intensity. This may not be used if "4" is used.
16	Causes the "4" or "8" setting to be used even if the COLOR_TRANS setting had no effect. This is an override switch that you can use to cause all foreground intensities to be set to high or low.
32	Forces the background to high-intensity. This will not be applied to a black background.
64	Forces the background to low-intensity. This may not be used if "32" is used.
128	Forces the background to high-intensity, but only if it is black. This may be used in conjunction with setting "32" or "64" for special effects.
256	Causes the "32", "64", or "128" setting to be used even if the COLOR_TRANS setting had no effect.

These transformations are performed in the order listed above. After this variable is applied, the **COLOR_TABLE** setting is applied to the program.

IO_CREATES

Setting this configuration variable to a “1” (on, true, yes) causes the runtime system to create a relative or indexed file when the program attempts to open a nonexistent file for I-O. This is provided for compatibility with some other COBOL systems. The default value is “0” (off, false, no).

IO_FLUSH_COUNT

Use this variable to specify how often the runtime should flush pending screen output during file operations. When set to a positive value, the variable indicates the number of file operations to perform between each screen flush. By default, IO_FLUSH_COUNT is set to 20.

For optimal performance, set IO_FLUSH_COUNT to zero (“0”). When IO_FLUSH_COUNT is set to zero, COBOL file verbs will not automatically flush pending screen output.

To reduce unexpected screen behavior, however, leave this variable at its default setting. The overhead at the default setting is small.

IO_READ_LOCK_TEST

When this variable is set to “1” (on, true, yes), the runtime will cause a read with no lock to fail if the file is opened for I-O and the record is locked. This setting will work with Vision files only. The setting is provided for compatibility with some other COBOL systems. The default value is “0” (off, false, no). The default behavior is to allow the read to succeed.

IO_SWITCH_PERIOD

The value of this variable affects the frequency with which the program’s threads change control based on file IO activity. After a thread performs the value of IO_SWITCH_PERIOD operations, the runtime switches control to

another thread (if one exists). Note that because thread switching is also affected by other program operations (such as display I/O), it is impossible to predict or absolutely control when a thread will change control.

The default value of IO_SWITCH_PERIOD is “10”. This value will provide good results with most applications. To produce behavior that more closely imitates that of Versions 6.1 and earlier, set IO_SWITCH_PERIOD to “1”. Zero and negative values are invalid and will result in undefined behavior.

ISOLATE_FILE_CREATES

It is possible to experience unexpected file errors when trying to create a file if another process is simultaneously creating or removing the same file name. Setting ISOLATE_FILE_CREATES to “1” (on, true, yes) causes files to be created with temporary names and then renamed when they are fully formed. This prevents another process from interfering with the creation. This option is effective only with Vision, and has undefined effects when used with other file systems. We recommend that you use this option only if you are experiencing unexpected errors when trying to create a file.

JAVA_LIBRARY_NAME

This variable is designed for those calling a Java program from COBOL via the C$JAVA library routine. In this variable, specify the name of the DLL that exports the Java Native Interface (JNI) API for loading the JVM. In the case of JRE 1.4.2_04, the file is called “jvm.dll” or “libjvm.so”. If the path to the DLL is in the Path environment variable, the filename is sufficient here; otherwise, this name should be fully qualified with the path.

For details see thee **C$JAVA** routine in Appendix I. For information on calling Java from COBOL using C$JAVA, see section 2.3.1 in *A Guide to Interoperating with ACUCOBOL-GT.*

JAVA_OPTIONS

This variable is designed for those calling a Java program from COBOL via the **C$JAVA** library routine. In this variable, specify the command-line options that you want passed to the JVM when it is started.

Note that both CLASSPATH (java.class.path system property) and the java.library.path must be configured in order for C$JAVA to locate the Java class to run. The CLASSPATH is the location of ".jar" or ".class" files. The java.library.path is the location DLLs or shared objects that are required either by the runtime or by the Java Virtual Machine (JVM).

If these properties are not set in the environment, use JAVA_OPTIONS to set CLASSPATH and java.library.path. For example:

```
JAVA_OPTIONS="-Djava.library.path="c:\usr\lib" -Xms128m
-Xmx128m -classpath /java/MyClasses/myclasses.jar"
```

JUSTIFY_NUM_FIELDS

When this variable is set to "1" (on, true, yes), all entry fields that have a numeric or numeric-edited VALUE data item associated with them are right justified (the same as if the RIGHT style were specified). You can inhibit this for a given field by specifying the LEFT style for the entry field. Note that this variable is examined only when each entry field is created and has no further effect. The default value is "0" (off, false, no).

KBD

These variables can be used in conjunction with the KEYBOARD variable to set global terminal attributes. For details, see the *ACUCOBOL-GT User's Guide*, **section 4.3.2.1, "The KEYBOARD variable."**

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

KEY_MAP

This is an obsolete variable that has been replaced by the **KEYSTROKE** configuration variable. Its setting is ignored.

KEYBOARD

This variable sets global terminal attributes. For details, see the *ACUCOBOL-GT User's Guide*, **section 4.3.2.1, "The KEYBOARD variable."**

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

KEYSTROKE

This variable redefines the action of a particular keystroke. It can also control mouse handling. For details on redefinition of keystrokes, see **section 4.3.2, "Redefining the Keyboard,"** in Book 1, *ACUCOBOL-GT User's Guide*. Information on mouse handling is provided in **Chapter 7** in Book 2, *ACUCOBOL-GT User Interface Programming*.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

LC_ALL

This variable supports the transfer of double-byte character variables and string literals and has meaning only on 32-bit Windows systems that support double-byte characters (e.g., Asian Windows machines). It should not be set in other environments and is needed only if you are passing data to a COBOL program from another language, such as Visual Basic, and are using **C$GETVARIANT** or **C$SETVARIANT**. By using the LC_ALL configuration variable, you cause the runtime to set the locale to a particular

value. You do not need to set this variable on Japanese machines. The runtime automatically detects Japanese versions of Windows and automatically sets the locale, LC_ALL, to “Japanese_Japan.932”.

The default value for this variable is “C”. The C locale assumes that all characters are 1 byte and that their value is always less than 256. The value of LC_ALL is in the format:

```
language[_country[.code_page]]
```

or

```
.code_page
```

where “language” is one of the supported language strings, “country” is one of the supported country or region strings, and “code_page” is the Windows code page setting for the language and country. “country” and “code_page” are optional. For example, the following are all equivalent:

```
LC_ALL Japanese
LC_ALL Japanese_Japan
LC_ALL Japanese_Japan.932
LC_ALL .932
```

For Korean double-byte character support under Windows use:

```
LC_ALL Korean
```

For Chinese use:

```
LC_ALL Chinese
```

or

```
LC_ALL Chinese-simplified
```

or

```
LC_ALL Chinese-traditional
```

The following are the supported language strings:

Chinese	Chinese	“chinese”
Chinese	Chinese (simplified)	“chinese-simplified” or “chs”
Chinese	Chinese (traditional)	“chinese-traditional” or “cht”

Czech	Czech	“csy” or “czech”
Danish	Danish	“dan” or “danish”
Dutch	Dutch (Belgian)	“belgian”, “dutch-belgian”, or “nlb”
Dutch	Dutch (default)	“dutch” or “nld”
English	English (Australian)	“australian”, “ena”, or “english-aus”
English	English (Canadian)	“canadian”, “enc”, or “english-can”
English	English (default)	“english”
English	English (New Zealand)	“english-nz” or “enz”
English	English (UK)	“eng”, “english-uk”, or “uk”
English	English (USA)	“american”, “american english”, “american-english”, “english-american”, “english-us”, “english-usa”, “enu”, “us”, or “usa”
Finnish	Finnish	“fin” or “finnish”
French	French (Belgian)	“frb” or “french-belgian”
French	French (Canadian)	“frc” or “french-canadian”
French	French (default)	“fra” or “french”
French	French (Swiss)	“french-swiss” or “frs”
German	German (Austrian)	“dea” or “german-austrian”
German	German (default)	“deu” or “german”
German	German (Swiss)	“des”, “german-swiss”, or “swiss”
Greek	Greek	“ell” or “greek”
Hungarian	Hungarian	“hun” or “hungarian”
Icelandic	Icelandic	“icelandic” or “isl”
Italian	Italian (default)	“ita” or “italian”
Italian	Italian (Swiss)	“italian-swiss” or “its”
Japanese	Japanese	“japanese” or “jpn”
Korean	Korean	“kor” or “korean”
Norwegian	Norwegian (Bokmal)	“nor” or “norwegian-bokmal”

Norwegian	Norwegian (default)	“norwegian”
Norwegian	Norwegian (Nynorsk)	“non” or “norwegian-nynorsk”
Polish	Polish	“plk” or “polish”
Portuguese	Portuguese (Brazilian)	“portuguese-brazilian” or “ptb”
Portuguese	Portuguese (default)	“portuguese” or “ptg”
Russian	Russian (default)	“rus” or “russian”
Slovak	Slovak	“sky” or “slovak”
Spanish	Spanish (default)	“esp” or “spanish”
Spanish	Spanish (Mexican)	“esm” or “spanish-mexican”
Spanish	Spanish (Modern)	“esn” or “spanish-modern”
Swedish	Swedish	“sve” or “swedish”
Turkish	Turkish	“trk” or “turkish”

The following are the supported country/region strings:

Australia	“aus” or “australia”
Austria	“austria” or “aut”
Belgium	“bel” or “belgium”
Brazil	“bra” or “brazil”
Canada	“can” or “canada”
Czech Republic	“cze” or “czech”
Denmark	“denmark” or “dnk”
Finland	“fin” or “finland”
France	“fra” or “france”
Germany	“deu” or “germany”
Greece	“grc” or “greece”
Hong Kong	“hkg”, “hong kong”, or “hong-kong”
Hungary	“hun” or “hungary”
Iceland	“iceland” or “isl”
Ireland	“ireland” or “irl”

Italy	“ita” or “italy”
Japan	“japan” or “jpn”
Mexico	“mex” or “mexico”
Netherlands	“nld”, “holland”, or “netherlands”
New Zealand	“new zealand”, “new-zealand”, “nz”, or “nzl”
Norway	“nor” or “norway”
Peoples Republic of China	“china”, “chn”, “pr china”, or “pr-china”
Poland	“pol” or “poland”
Portugal	“prt” or “portugal”
Russia	“rus” or “russia”
Singapore	“sgp” or “singapore”
Slovak Republic	“svk” or “slovak”
South Korea	“kor”, “korea”, “south korea”, or “south-korea”
Spain	“esp” or “spain”
Sweden	“swe” or “sweden”
Switzerland	“che” or “switzerland”
Taiwan	“taiwan” or “twn”
Turkey	“tur” or “turkey”
United Kingdom	“britain”, “england”, “gbr”, “great britain”,”uk”, “united kingdom”, or “united-kingdom”
United States of America	“america”, “united states”, “united-states”, “us”, or “usa”

LICENSE_ERROR_MESSAGE_BOX

This configuration variable prevents **acushare** licensing errors from appearing in a message box, which requires a response from the user. The error messages will instead go to the error output (stderr, or an error file if one

is specified). Set LICENSE_ERROR_MESSAGE_BOX to 0 to prevent these messages from appearing in a message box. The default value is 1, which allows these messages to appear in a message box.

LISTS_UNBOXED

Meaningful only in character-based environments, this variable indicates whether list boxes should be boxed (set to a value of "0", off, false or no) or unboxed (set to a value of "1", on, true or yes). The default setting is "0".

LITERAL_ENTRY

This variable can be used to loosen the literal match restrictions of ENTRY point name matching logic. By default, the matching logic is case sensitive and distinguishes between hyphens and underscores. Setting LITERAL_ENTRY to "0" (off, false, no) causes the runtime to handle ENTRY point name matching with case insensitivity (upper and lower case equivalent) and to treat hyphens and underscores ("-", "_") as equivalent. The default value is "1" (on, true, yes).

LOCK_DIR

When set, this controls automatic device locking on UNIX systems. This is described in **Section 6.1.5** of the *ACUCOBOL-GT User's Guide*. The default value is empty.

LOCK_OUTPUT

When set to a "1" (on, true, yes), this configuration variable causes all files open for OUTPUT to be locked for exclusive use. Setting this can dramatically improve performance on VMS machines. Some other COBOL systems lock files that are open for OUTPUT. The default value is "0" (off, false, no).

LOCK_SORT

When this configuration variable is set to a "1" (on, true, yes), input files to the SORT verb are opened for INPUT ALLOWING READERS. This can improve the performance of the SORT verb slightly and also ensure that the data being sorted is not modified. The default value is "0" (off, false, no).

LOCKING_RETRIES

This configuration variable is designed for Windows 98 systems. It gives you some control over situations where a user must wait for access to a shared file. The runtime will try repeatedly to acquire the file lock, up to 400 times by default. Set this variable to the number of attempts you would like the runtime to make to acquire the file lock.

LOCKS_PER_FILE

This value determines the maximum number of record locks that can be held on a file by a single process. This value affects only the files that are maintaining multiple record locks. The default setting is "10". The maximum value is 32767 for Vision files. Setting this variable to its maximum value can waste resources and is not recommended.

Note: If you increase the value of LOCKS_PER_FILE, you may wish to increase the value of **MAX_LOCKS** as well.

LOG_BUFFER_SIZE

This sets the maximum buffer size, in bytes, for the transaction log file. Acceptable values are from "0" to "32767". LOG_BUFFER_SIZE is examined before each write to the log file. Its default value is "512". If LOG_BUFFER_SIZE is set to "0", then writes to the log file are synchronous (unbuffered). This value can also be set inside a COBOL program with the SET ENVIRONMENT verb.

LOG_DEVICE

Setting this value to “1” (on, true, yes) causes the transaction management system to assume that the log file is actually a device, rather than a file. This means that a special device locking method will be used on the log file (see the *ACUCOBOL-GT User’s Guide* **Section 6.1.5**, “Device Locking Under UNIX”). It also guarantees that the log file will be opened “append” and that no seeks will be performed on it. This allows for the use of a tape device for the log file on many systems. The default setting is “0” (off, false, no).

LOG_DIR

This variable allows you to specify a directory to be used for holding the temporary files generated by the transaction management system. The value of LOG_DIR is treated as a prefix, much like FILE_PREFIX. If no directory is specified, temporary files are placed in the current directory.

Note: In general, you should not use remote name notation in the LOG_DIR variable. Although remote name notation is allowed for the LOG_DIR variable, it is not advisable to place temporary files on a remote server.

LOG_ENCRYPTION

If this value is set to “1” (on, true, yes), record images are encrypted before they are written to the transaction log file. The default setting is “0” (off, false, no).

LOG_FILE

This identifies the name of the default log file for transaction management. The default log file is opened at the beginning of the first transaction unless **NO_LOG_FILE_OK** is set to “1.” If it does not exist, it is created. This variable can be set programmatically with the SET ENVIRONMENT verb.

The default setting is empty. Unless you have set NO_LOG_FILE_OK to "1", you must set the LOG_FILE variable if you want to use transaction management.

Remote name notation is allowed for the LOG_FILE variable if your runtime is client-enabled. See *ACUCOBOL-GT User's Guide* sections 5.2.1 and **Section 5.2.2** for more information about client-enabled runtimes and remote name notation.

LOGGING

Setting this variable to "0" (off, false, no) disables the logging of file updates to the log file. This means that the data file recovery process (using the library routine **C$RECOVER**) is impossible. However, because rollback information is maintained internally by the runtime, the program can still use the transaction management system to START, COMMIT, and ROLLBACK transactions. This variable can be set programmatically with the SET ENVIRONMENT verb. The default setting is "1" (on, true, yes).

LOGICAL_CANCELS

This variable is used to enable *logical cancels*. Logical cancels reduce CALL overhead and can, as a result, improve performance. Cancels, both logical and physical (the default), are initiated by the CANCEL verb or through a function of the C interface. A discussion of memory management and physical and logical cancels is located in section **Section 6.3, "Memory Management,"** in Book 1. A description of the **CANCEL Statement** is located in section 6.6 of Book 3.

LOGICAL_CANCELS can be set to the following values:

-1 (default) all cancels are physical cancels except for programs called from CICS that have the "Resident" attribute set to TRUE.

0 all cancels are physical cancels. Cancels of programs called with the C interface are treated as physical cancels even if the “cache” field is set to “1”.

1 all cancels are logical cancels. Cancels of programs called with the C interface are treated as logical cancels even if the “cache” field is set to “0”.

LOGICAL_CANCELS is used in conjunction with the **DYNAMIC_MEMORY_LIMIT** configuration variable, which specifies the size of the dynamic memory pool for programs. See its entry in this appendix.

MAKE_ZERO

When set to a “1” (on, true, yes), this configuration variable causes a numeric data item that contains non-numeric data to be treated as zero when that item is used in a numeric statement. Setting the value to “0” (off, false, no) causes the item to be treated “as is” with whatever effects that will have. The default value is “1” (on, true, yes).

MASS_UPDATE

When set to “1” (on, true, yes), this variable causes all OPEN...WITH LOCK statements to be treated as if they were written OPEN...WITH MASS_UPDATE. This does not apply to OPEN INPUT, however. Setting this configuration variable will improve file performance for applications that lock files, but may lead to file corruption if the program is killed before it completes. For more information on this topic, see the *ACUCOBOL-GT User’s Guide*, **section 6.1.6, “Indexed File Considerations.”** The default value is “0” (off, false, no).

MAX_ERROR_AND_EXIT_PROCS

This variable sets the maximum number of error and exit procedures that a program can install or call. The default is 64. If the limit is exceeded, program execution is aborted. For more information about error and exit procedures, see **Error and Exit Procedures**, **CBL_ERROR_PROC**, **CBL_EXIT_PROC**, and **CBL_GET_EXIT_INFO** in Appendix I.

MAX_ERROR_LINES

This variable sets the maximum number of lines that can be included in the error file. It's especially useful when you are using the file trace function of the debugger. When the size of the error file reaches the number of lines specified in this variable, the error file is *rewound* to its beginning, and subsequent lines of output overwrite existing lines in the file. For example, if you set the maximum to 300, then lines 301 and 302 would overwrite lines 1 and 2 in the file. Note that there is an upper limit of 32767, setting a larger number causes odd behavior. The default value is "0", which means do not limit the size of the file.

When using the "-l" runtime option (which causes the configuration settings to be logged in the trace file) the trace file wraps to the line below the MAX_ERROR_LINES entry. For this reason, MAX_ERROR_LINES should be the last entry in the runtime configuration file

To help you locate the end of the file, the message "*** End of log ***" is output whenever the runtime shuts down. Line numbers are included in each line of the file, in columns one through seven.

This variable is not available on VMS systems.

MAX_FILES

This variable sets the maximum number of files that can be opened by the runtime system. The default value is "32". Keeping this value small conserves memory. Many operating systems limit the number of files that can be opened by a single process, so you may have to make some adjustments there too. The maximum value of MAX_FILES is 32767.

MAX_LOCKS

This value sets the maximum number of record locks that can be held by the runtime system for all of the files together. The default value matches the setting of **MAX_FILES**. Many operating systems also have limits on the number of record locks that can be held, so you may have to make adjustments there too. The maximum value is 32767 for Vision files. Setting this variable to its maximum value can waste resources and is not recommended.

Note: If you change the value of MAX_LOCKS, you should consider changing the value of **LOCKS_PER_FILE** as well.

MENU_ITEM

This variable affects the behavior of pull-down menus.

The default action of a menu item is to return an exception value equal to the item's ID. You can change the default action of a particular item by using MENU_ITEM.

Use MENU_ITEM in the same fashion as the KEYSTROKE variable, except that the last entry on the line is the menu's ID, not the key code. For example, to cause a menu item whose ID is "200" to act the same as the Delete key, use the following:

```
MENU_ITEM  Edit=Delete  200
```

Alternately, you could cause menu item "200" to call the "notepad" sample program by using:

```
MENU_ITEM  Hot_Key="notepad" 200
```

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

MESSAGE_BOX_COLOR

This variable is used on character-based hosts to specify the color and video attributes of characters displayed in a message box window. MESSAGE_BOX_COLOR can be set to a variety of numeric values that express combinations of color and video attributes. See the documentation for the COLOR phrase in the "Common Screen Options" section of the *ACUCOBOL-GT Reference Manual* (**Section 6.4.9**).

If MESSAGE_BOX_COLOR is set to "0" (the default value), the message box is displayed with the default window colors and attributes.

MESSAGE_QUEUE_SIZE

This variable sets the initial size of the message queue, in bytes. The message queue is dynamically resized, as needed, to hold large messages. However, it is not resized to hold multiple messages (instead, the sending threads wait until the queue empties). Setting the value larger than the default will allow more messages to be queued. Setting it to a smaller value will allow fewer messages to be queued and conserves memory. The default size is 32767.

MIN_REC_SIZE

This configuration variable sets the minimum record size for print records, and for line sequential files for which trailing space removal has been specified. The default value is "1". If set to "0", then a record will be reduced to zero size (except for the line delimiter) if the line is blank. You may also set MIN_REC_SIZE to higher values to establish a minimum record size other than one.

MONOCHROME

When set to "1" (on, true, yes), this configuration variable disables color output for machines with graphics video cards. The default value is "0" (off, false, no).

ACUCOBOL-GT assumes that machines with graphics video cards are color machines. If you have a monochrome monitor attached to such a machine, some program screens may be difficult to see. You tell ACUCOBOL-GT to disable color output for these machines through the MONOCHROME configuration variable. When this variable is set to a "1", ACUCOBOL-GT will use only black and white.

MOUSE

This variable has meaning only on systems with a mouse. When the user selects a field in the Screen Section, the exact behavior depends on the field's underlying type. The runtime distinguishes between three classes of fields: numeric, numeric-edited, and all others. These are referred to respectively as NUMERIC, EDITED, and ALPHA.

You can control the behavior of the mouse with regard to each of these field types with the MOUSE configuration variable. This variable takes as its arguments one of the field-type names and two keywords. The first keyword defines how the field is selected when the user presses the left button. The second keyword indicates the shape that the mouse pointer should take while in the field. The first keyword can be one of the following:

None	Indicates that this type of field may not be selected with the mouse. When this keyword is used, then the second keyword (which defines the mouse's shape) is ignored. The mouse adopts the shape used for areas of the screen that are not part of any field.
Field	Indicates that pressing the left button anywhere in the field will cause the cursor to be positioned at the beginning of the field.
Character	Indicates that pressing the left button in the field will position the cursor at the character pointed to by the mouse. If this is past the last non-prompt character in the field, the cursor will be placed just after the last non-prompt character.

The second keyword indicates the shape that the mouse pointer should take while in the field. It can be one of the following:

Arrow	The mouse pointer appears in the default arrow shape.
Bar	The mouse appears as a vertical bar. This is the "I-Bar" shape typically used to indicate that the mouse can be positioned at a particular character.
Cross	The mouse appears as cross-hairs.

You may also define the shape that the mouse will take when it is used in the *current field*. Because the action of the mouse is the same for all field types once they become the current field, the mouse shape is the same for all three types. You set the desired shape using the Current keyword in the MOUSE configuration variable. The default shape is the Bar shape.

Configuring the MOUSE variable

Depending on where you are setting the MOUSE variable, there are three methods of setting its configuration:

1. If you want to implement this variable in a configuration file, the variable can be set without using the equals sign. For example:

   ```
   MOUSE_NUMERIC_SHAPE      Bar
   ```

2. If you are setting the variable in the Windows environment, the variable would look this:

   ```
   SET MOUSE_NUMERIC_SHAPE=Bar
   ```

3. If you are setting the variable in your program using COBOL syntax, the variable would look like this:

   ```
   SET ENVIRONMENT "MOUSE_NUMERIC_SHAPE" TO "Bar"
   ```

The default configuration is as follows:

MOUSE_ALPHA_CHARACTER	Bar
MOUSE_NUMERIC_FIELD	Arrow
MOUSE_EDITED_FIELD	Arrow
MOUSE_CURRENT	Bar

You may place multiple entries on the MOUSE configuration line, but you are not required to do so.

The following configuration variables can also be used to set the behavior of the mouse:

To set field selection:

MOUSE_ALPHA_SELECT

MOUSE_EDITED_SELECT

MOUSE_NUMERIC_SELECT

To set cursor shape:

MOUSE_ALPHA_SHAPE

MOUSE_EDITED_SHAPE

MOUSE_NUMERIC_SHAPE

MOUSE_CURRENT_SHAPE

With these variables, you need to set the first and second keywords separately. For example, to change the defaults shown above for a numeric field, you would enter:

```
MOUSE_NUMERIC_SELECT  character
MOUSE_NUMERIC_SHAPE  bar
```

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

MOUSE_FLAGS

This variable has meaning only on systems with a mouse. Indicate which mouse actions will return an exception value to your program by setting the value of the configuration variable MOUSE_FLAGS. Mouse actions that you don't want to deal with will be ignored. The value you set is actually one or more values added together. The possible values are:

1	Causes ACUCOBOL-GT to use its automatic mouse handling facility. (default)
2	Enables the "left button pushed" action.
4	Enables the "left button released" action.
8	Enables the "left button double-clicked" action.
16	Enables the "middle button pushed" action.
32	Enables the "middle button released" action.
64	Enables the "middle button double-clicked" action.
128	Enables the "right button pushed" action.
256	Enables the "right button released" action.
512	Enables the "right button double-clicked" action.
1024	Enables the "mouse moved" action.
2048	Forces the mouse pointer always to be the default arrow shape when you are using automatic mouse handling. If this is not set, then the shape of the mouse pointer varies depending on various other configuration options. See MOUSE above.
16384	This causes all enabled mouse actions that occur within your application's window to return an exception value. If this is not set, then only mouse actions that occur within the current ACUCOBOL-GT window return a value. (The current ACUCOBOL-GT window is a window created by your program with the DISPLAY WINDOW verb.)

For example, if you wanted your program to receive an exception value whenever the user pressed either the left or right button, you would set:

```
MOUSE_FLAGS  130
```

NO_CONSOLE

This variable has meaning only on graphical systems that create an application window, such as Windows. Set this variable to "1" (on, true, yes) to indicate that you've built your own user interface entirely in C or that you are using an interface created by a code-generating tool. This is equivalent to executing the runtime system with the "-b" command-line option. When this variable is set to "1", the runtime won't create its own application window. Instead, your C code must build its own window. When you provide your own user interface, you may not use ACCEPT or DISPLAY verbs in your COBOL program (except for those that don't interact with the screen or keyboard).

The default value is "0" (off, false, no).

NO_LOG_FILE_OK

Setting this variable to "1" (on, true, yes) eliminates the need to specify a default transaction log file with the **LOG_FILE** variable. When this variable is set, the runtime will write transaction recovery information only to the log files specified via the **filename_LOG** variables.

The default value is "0" (off, false, no).

NO_TRANSACTIONS

This variable allows you to disable ACUCOBOL-GT's built-in transaction management system. When NO_TRANSACTIONS is set to "1" (on, true, yes), all calls to ACUCOBOL-GT's built-in transaction management system return without doing any work. This affects all file systems in use. The value of NO_TRANSACTIONS is checked once, the first time a BEGIN, COMMIT, or ROLLBACK is attempted and it is not checked again. Therefore, although the variable can be set in the program, the effective setting cannot be changed after the first transaction management action has been attempted. The default value is "0" (off, false, no), meaning that the built-in transaction management facility is enabled.

NT_OPP_LOCK_STATUS

This configuration variable controls how files on a shared drive are opened if you are working in the Windows opportunistic locking mode. NT_OPP_LOCK_STATUS can take one of four values: CREATEFILE, SAFE, GETFILETYPE, or FAST. The default value is "SAFE", which is a synonym for "CREATEFILE".

If you set this variable to "GETFILETYPE" or "FAST" (synonyms for each other), the runtime uses the fast method of opening files.

Note: If your Windows installations are not completely up to date with all available patches from Microsoft, particularly those related to opportunistic locking, the GETFILETYPE or FAST setting may cause file corruption (error 98).

NESTED_AX_EVENTS

When an application dialog contains an ActiveX control that is assigned an event procedure, the event handler sometimes triggers additional ActiveX events. This variable determines whether or not the event procedure will be nested.

Set this variable to "1" (on, true, yes) if you want the event procedure to be nested. (This is the default). When NESTED_AX_EVENTS is set to "1", the runtime allows events to trigger while it is processing other events. It is your responsibility to know when the event procedure is busy and reject events when this is the case, or to look for specific events and properly handle them. For example, consider this code:

```
AX-EVENT.
MOVE 1 TO MY-LOOP.
PERFORM UNTIL MY-LOOP = 10
* Do some stuff
ADD 1 TO MY-LOOP
END-PERFORM
```

When NESTED-AX-EVENTS is set to “1”, it is possible that when your code is inside the event, possibly executing the loop for the fifth time, a new event triggers, setting MY-LOOP back to “1”. The perform loop could execute simultaneously in two threads on the same data, and the runtime could crash. When you do not have reentrant events, MY-LOOP can only become “1” one time. This is the case when NESTED-AX-EVENTS are set to “0”.

Set NESTED_AX_EVENTs to “0” (off, false, no) if you do not want the event procedure to be nested. Be aware, however, that this option may cause you to lose certain events (typically events triggered by modifications made in the event procedure).

When NESTED_AX_EVENTS is set to “0”, once a program has entered an ActiveX control’s event procedure, new events are ignored. This prevents the runtime from executing the same code, at the same time. However, events that are imperative for the code execution may be refused.

Note: NESTED_AX_EVENTS applies only to the local runtime and has no effect in thin client scenarios.

NO_BARE_KEY_LETTERS

This variable is related to the terminal manager KEYSTROKE EDIT=ALT (see **Section 4.3.2.2, “The KEYSTROKE variable”**) method of requiring users to press the Alt key along with the key letter to move to a new control. In character mode and when accepting a push-button, check box, or radial button, the runtime’s default behavior is to terminate the accept if the key letter of a control is pressed, and move to that control.

This behavior can be changed by setting NO-BARE-KEY-LETTERS to “TRUE”. When set to “TRUE”, in order to move to these types of controls, users will be required to press the key set by the KEYSTROKE setting EDIT=ALT before pressing a key letter of that control. The default setting is “FALSE”, which uses the behavior described in the previous paragraph.

NUMERIC_VALIDATION

If this configuration variable is set to “1” (on, true, yes), the runtime checks for proper format when data is converted to a numeric type (via a MOVE, for example). When NUMERIC_VALIDATION is set to the default “0” (off, false, no), numeric conversion checking does not occur.

OLD_ARIAL_DIMENSIONS

The Arial font shipped with Windows 98 Version 2 has a character width of 35 pixels, while the Arial font shipped with earlier versions of Windows and Windows NT has a width of 23 pixels. This might cause field overlap or screen distortions in programs that rely on the smaller version of the Arial font. Setting this variable to “1” (on, true, yes) causes the runtime to use the 23-pixel measurement for fields, regardless of which version of Arial (35 or 23-pixel) is being used.

The default value of “0” (off, false, no) will cause fields to be sized according to the version of Arial used.

Note: Because the 35-pixel version of Arial is wider, uppercase characters may be truncated when their size is computed with the 23-pixel measurement. Use of this variable may not compensate for all possible character width sizing issues. Some reprogramming of your screens may be required.

OPEN_FILES_ONCE

This variable allows different logical files that access the same physical file to open the physical file only once. The default for this variable is “1” (on, true, yes). This variable is valid only for UNIX runtimes.

OPTIMIZE_CONTROL_RESIZE

This configuration variable determines how the runtime optimizes control resize requests. Prior to Version 5.2, the runtime would optimize away requests to resize a screen control if the new size and position matched the control's current size and position on the screen. In Version 5.2, or later, the runtime optimizes the control resize request using the SIZE and LINES indicated (or implied) by the program. Setting OPTIMIZE_CONTROL_RESIZE to "0" (off, false, no) prevents any optimization of control resizing operations. The default of "1" (on, true, yes) enables the new behavior. See **Appendix C, "Appendix C: Changes Affecting Previous Versions,"** for more details.

OPTIMIZE_INDIVIDUAL_LINKAGE

This variable enables the runtime to perform address optimizations on each Linkage item individually. In versions prior to 8.1 this optimization was done either for all Linkage items or for none of them, which could result in scenarios where optimizations could have occurred on some items, but did not.

The default and recommended setting is "1" (on) because the main effect is improved CALL performance in a greater number of scenarios. Usually, the only reason to turn this variable off is if a flaw is suspected in the optimization.

PAGE_EJECT_ON_CLOSE

When set to "1" (on, true, yes), this variable will cause print files to print a page advance record when the file is closed, unless the close contains the NO REWIND phrase. This is provided for compatibility with RM/COBOL version 2. The default value is "0" (off, false, no).

PAGED_LIST_SCROLL_BAR

This variable applies only in text-mode environments. PAGED_LIST_SCROLL_BAR can be set to "-1", "0", or "1". The default value is "-1". When set to "-1", the vertical scroll bar is displayed to the right of a paged list box if the user interface configuration supports a mouse. Otherwise, the right border appears just like the left border. The appearance depends on whether the NO-BOX style is set and the values of the **FULL_BOXES** and **LISTS_UNBOXED** configuration variable settings.

The runtime internally calls the W$MOUSE library routine with the TEST-MOUSE-PRESENCE op-code to determine whether the user interface supports a mouse. Note that mouse support is available for X terminals only if the a_termcap entry includes the "km" function. (See the *AcuCOBOL-GT User's Guide*, section 4.6.8, "Mouse Support for X Terminals.")

When PAGED_LIST_SCROLL_BAR is set to "1", the vertical scroll bar is always displayed to the right of a paged list box. When set to "0", the vertical scroll bar is never displayed to the right of a paged list box.

PARAGRAPH_TRACE

This variable is used for debugging purposes and turns on paragraph tracing. Set this variable to "1" (on, true, yes) to turn on paragraph tracing from within the configuration file or the COBOL program. This is equivalent to the debugger "tp" command. The COBOL program must be compiled with symbols ("-Gs", or anything that implies that option) to get any error output.

PERFORM_STACK

This variable sets the depth to which PERFORM statements can be nested at runtime when the "-Zr" compile-time option is used. The default value is "128". The maximum value is "10916". Setting this variable to its maximum value can waste resources and is not recommended.

PRELOAD_JAVA_LIBRARY

This variable is designed for those calling a Java program from COBOL via the C$JAVA library routine. By default, the Java Virtual Machine (JVM) is loaded by the runtime the first time it executes a CALL C$JAVA statement. If you want to load the JVM when the runtime is started, set this configuration variable to "1". If you do not want to preload the JVM, set the variable to "0".

See Appendix I for details on the **C$JAVA** routine. For information on calling Java from COBOL using C$JAVA, see section 2.3.1 in *A Guide to Interoperating with ACUCOBOL-GT.*

PROFILE_TYPE

This configuration variable provides an optional method of profiling ACUCOBOL-GT on Windows called "COUNTER". The counter method uses the debugger to perform counting and appears to provide the most accurate results in Windows environments.

Set the PROFILE_TYPE configuration variable to either "ASYNCH" or "COUNTER". When set to the default value of "ASYNCH", the runtime performs profiling the way it historically has. When set to the value "COUNTER", the runtime uses this method of profiling. Note that your COBOL programs must be compiled with "-Gd" as well as "-Gs" options to use the counter method.

The counter method is also available on UNIX and can be used if profiling your COBOL results in a message similar to "profile timer expired". This method doesn't completely solve that problem, but does substantially mitigate it.

PROMPTING

This variable is used on character-based hosts to turn ENTRY-FIELD prompting off or on. When PROMPTING is set to "0" (off, false, no) prompting is not performed. The default value of PROMPTING is "1" (on, true, yes).

QUEUE_READERS

This configuration variable evens out user access by modifying the rules Vision uses when several users are accessing a file. This variable applies to UNIX machines. Because of restrictions, it is recommended only for sites that are experiencing performance problems with updaters.

By default, the runtime allows multiple readers to access a file simultaneously, while updaters require exclusive access to the file. When a file has many readers, an updater can get blocked out of the file for a period of time while the runtime waits for a moment when there are no active readers. While this allows processes that read the file to have nearly immediate access, updaters may need to wait for a noticeable amount of time.

The QUEUE_READERS configuration variable lets you request that the runtime service each user in turn. This means a reader will have the same priority for accessing a file as an updater does. Each user is processed in turn so that access to files is evenly balanced among all the users.

By default, QUEUE_READERS is set to “0” (off, false, no). Set it to “1” (on, true, yes) to force the readers to take turns instead of having immediate access.

Because of technical limitations in the UNIX file system, if you use this configuration variable you must provide read-write access to all indexed and relative files that the runtime uses. This is true even for files that are open only for input--UNIX requires that the runtime have write access to the files in order to place the kind of lock that causes each user to take turns.

QUIT_MODE

This variable has meaning only on graphical systems such as Windows. It gives you control over the Close action that appears on the System menu in a graphical environment. You may use the QUIT_MODE variable with only the main application window. All other windows return the CMD-CLOSE event when they are closed. QUIT_MODE has no effect on windows created with the NO-CLOSE phrase (see formats 11 and 12 of the **DISPLAY Statement**, in Book 3, *ACUCOBOL-GT Reference Manual*, section 6.6).

Many COBOL programs should not be shut down in an uncontrolled manner. This is especially true of any application that updates several files in a row. If the program is halted after updating the first file, but before updating the last, the files are left in an inconsistent state. For this reason, ACUCOBOL-GT allows you to control the Close action.

To do this, you set QUIT_MODE to a non-zero value. The value that you specify affects the Close action as follows:

-2	Disable Close: disables the Close action entirely. The Close menu item will appear gray on the System menu, and the user will not be able to select it.
-1	Close only on input: the runtime disables the Close action except when it is waiting for user input. This prevents the user from stopping the runtime in the middle of a series of file operations, but still allows the user to quit the application any time that the application is waiting for input.
0	Always Close: the runtime halts the program whenever Close is selected from the system menu.
>0	Program controlled Close: when a positive value is used, the Close item becomes a standard menu item with an ID equal to the value of QUIT_MODE. You may then handle the Close item just like any other menu item.

For example, if you set QUIT_MODE to “100”, then your program will receive exception value 100 when the user selects the Close item. If you wanted to call a special shutdown program when the user selected Close, you could assign the Close action to a hot-key program:

```
MENU_ITEM  Hot_Key =“shutdown”  100
```

In this example, the “shutdown” program might pop up a small window to confirm that the user wanted to exit and, if so, do a STOP RUN.

If you start your program in “safe” mode with the “-s” runtime option, then QUIT_MODE will be initialized to “-2” instead of “0”. This prevents the user from using the Close menu item. A QUIT_MODE entry in the configuration file takes precedence over the default handling of “-s”.

If a user attempts to end the Windows session when it is not allowed, a pop-up message box asks the user to terminate the application first. You can customize the message that appears in the box by setting the TEXT configuration variable, message number 18.

Note: The QUIT_MODE setting affects only the main application window. All other windows always return the event CMD-CLOSE when the window is closed.

QUIT_ON_FATAL_ERROR

This configuration variable applies only when running in HP COBOL compatibility mode (with the "-Cp" compiler option). The QUIT_ON_FATAL_ERROR configuration variable causes the runtime to call the MPE QUIT intrinsic when an error occurs. The MPE job control word (JCW) is then set, and the MPE environment can determine if the program terminated with a fatal error. When set to "1" (on, true, yes), QUIT_ON_FATAL_ERROR calls the MPE QUIT intrinsic. The default setting is "0" (off, false, no).

QUIT_TO_EXIT

When this variable is set to "1" (on, true, yes), the user must press the close button on the title bar (or an alternate close mechanism provided by the window) after the program executes a STOP RUN. The default value is "0".

RECURSION

ACUCOBOL-GT allows a program to call itself, directly or indirectly. A CALL statement that attempts to call an active program is termed a *recursive call*.

To use recursive calls, you must set the configuration variable RECURSION to "1" (on, true, yes). The default setting for RECURSION is "0" (off, false, no), which disallows recursive calls.

When you allow recursive calls, an active program may be called again. This causes a new copy of the program to be loaded into memory and executed, as if it were the first call of that program. Files and data described in that program are local to each copy of the program.

More specifically, the runtime assigns a *recursion level* to each recursively called program. The first time a program is called, it is assigned a recursion level of "0". If that program is still active and it is called again, it receives a recursion level of "1". The recursion level is incremented by 1 for each active copy of the same program.

When you call a program at a specific recursion level for the first time, it is freshly loaded from disk and its Working-Storage data items are given their initial values as defined by their VALUE clauses. Subsequent calls to a program at the same recursion level will find the files and data left in the same state that the program had when it last exited.

Files and data items are distinct between different recursion levels.

When you CANCEL a recursively called program, all of its inactive copies are removed from memory. Active copies are left alone. Subsequent calls to any of the canceled recursion levels will reload the program from disk and reinitialize the files and data items.

If you need to share data between different active copies of the same program, you can pass this data through the Linkage Section. Alternatively, you can share both files and data items by declaring them as EXTERNAL items. Yet another option is the **RECURSION_DATA_GLOBAL** configuration option.

The runtime system shares the program code for recursively called programs. Thus, while each recursion level has its own set of data, there is only one copy of the Procedure Division code in memory, regardless of how many active copies of the program there are. The runtime system does not, however, share overlays. Each copy of the program in memory has its own overlay area.

RECURSION_DATA_GLOBAL

This configuration variable allows you to configure the runtime so that each instance of a recursively called program shares the same data as the original instance of the program. The primary reason for configuring the runtime in this manner is if you are migrating code that relies on this behavior from another COBOL system, such as HP COBOL.

When RECURSION_DATA_GLOBAL is set to "1" (on, true, yes), files and data items in a recursive call of a program refer to the identical items in the original call of that same program. This is true regardless of the entry point into the program. Changes to data or file state made in any recursive instance of the program are seen by all other instances of that program in the same run unit.

Note that to use this feature, you must not only set the configuration variable RECURSION_DATA_GLOBAL to "1", you must also set the **RECURSION** configuration variable to "1" to allow recursion (which is not allowed under standard ANSI-85 COBOL rules).

The default is "0" (off, false, no).

REL_DELETED_VALUE

This configuration variable is helpful when you use relative files and need to have a valid record that contains binary zeros. However, because binary zeros are used as the deleted record marker, you have to be able to change this marker. REL_DELETED_VALUE can hold the ASCII character value for the new deleted record marker.

REL_LOCK_READ_THROUGH

This variable allows you to "read through" relative file record locks in Windows environments. This means that READs that do not assert a lock are allowed to READ a relative file record even if it is locked by another process. To turn on the "read through" capability, all processes accessing the relative file must set the configuration variable REL_LOCK_READ_THROUGH to "1" (on, true, yes). When this variable is turned on, the Vision library uses an

alternate location for relative file record locks that does not block other processes from reading the records. This is necessary because Windows has "enforced" file locks that preclude all other access to the locked region. Failure to set the configuration variable to the same value on all processes accessing the relative file results in undefined behavior. This variable has no effect on UNIX platforms. For more information about relative files and record locking, refer to **section 6.1, "Handling Files,"** of the *ACUCOBOL-GT User's Guide*.

RENEW_TIMEOUT

When set to "1" (on, true, yes), this variable restarts the timeout used by ACCEPT BEFORE TIME after each keystroke that the user makes. The default setting is "0" (off, false, no), which means that the timeout is canceled as soon as the user starts typing.

RESIZE_FRAMES

This variable is used to turn off the automatic resizing of frames that is performed on character-based hosts. By default, a character-based host runtime automatically resizes frames to visually surround all controls whose home coordinates are bounded by the frame. This makes it easier to maintain applications that run on both character and graphical systems. To turn automatic resizing off, set RESIZE_FRAMES to "0" (off, false, no). The default value is "1" (on, true, yes).

RESIZE_FREELY

Normally, under a graphical host (such as Windows), the runtime will not allow you to resize an AUTO-RESIZE window larger than its logical size (as determined by the number of rows and columns the program requested when it created the window). When the RESIZE_FREELY variable is set to "1" (on, true, yes), the user can resize the window to any size. Maximizing the window will cause the window to occupy the entire available screen. The region of the window that lies outside of the logical area will not be accessible by the program and will be shown as the background color of the logical window (defined as the last background color to be applied to the

entire window). Setting this option is essentially a cosmetic change to the way your application looks when maximized. The default value is "0" (off, false, no).

RESTRICTED_VIDEO_MODE

This value determines which rules will be followed when the program is positioning attribute characters for "magic cookie" terminals (terminals whose attributes occupy screen positions). For more details, see the *ACUCOBOL-GT User's Guide*, **section 4.5, "Restricted Attribute Handling."** The default is "0".

RMS_NATIVE_KEYS

This variable is for use only on VAX/VMS systems. When it is set to "1" (on, true, yes), it causes the runtime to specify a key type for numeric keys. In order to make use of this variable, you must also create XFD files at compile-time ("-Fx"), and you must set the configuration variable **XFD_DIRECTORY** to point to the directory containing those XFD files.

When these steps have been taken, the runtime will create RMS files with keys having either the packed decimal or integer attribute, under certain conditions dictated by the RMS file system. In particular, a key will have the packed decimal or integer attribute only if it is a single-segment key and only if there is a single field in the key. In this case, a USAGE COMP-3 data item in the key will receive the packed decimal attribute, and a USAGE COMP-5 data item (if it is 2, 4, or 8 bytes long) will receive the integer attribute.

One effect of having this attribute set on key fields is that the order of the data is changed. Without this attribute, a file that has records with keys -3, -2, -1, 0, 1, 2, 3 would have those records ordered in this way: 0, 1, -1, 2, -2, 3, -3. With this attribute, those records would be ordered in this way: -3, -2, -1, 0, 1, 2, 3. The default setting for this variable is "0" (off, false, no).

SCREEN

This configuration variable controls a variety of screen configuration options. For details, see the *ACUCOBOL-GT User's Guide*, **section 4.4.2, "The SCREEN Option."**

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

SCREEN_COL_PLUS_BASE

This variable allows you to configure the COLUMN PLUS phrase in the Screen Section. This capability is provided to improve compatibility with other COBOLs. SCREEN_COL_PLUS_BASE allows you to choose the behavior that works best for your program. It takes the following values:

-1 (default) This value causes the runtime to determine the behavior of the COLUMN PLUS phrase based on whether ICOBOL compatibility has been specified with the "-Ci" compile option. If so, "COLUMN + 1" produces a space between items, and "COLUMN + 0" creates adjacent items. If ICOBOL compatibility has not been specified, "COLUMN + 1" produces adjacent items. This matches the prior behavior of ACUCOBOL-GT.

0 This value causes column adjustments to start counting at zero. In this case, "COLUMN + 0" produces adjacent items, and "COLUMN + 1" puts a space between items.

1 This value causes column adjustments to start counting at one. In this case, "COLUMN + 1" produces adjacent items, and "COLUMN + 2" puts a space between items.

SCREEN_TRACE

This variable is used for debugging and turns on screen tracing. Set this variable to "1" (on, true, yes) to turn on screen tracing from within the configuration file or the COBOL program. This is equivalent to the debugger "ts" command.

SCRIPT_STATUS

This variable controls the behavior of ACCEPT FROM INPUT STATUS when the input is not attached to a terminal. If SCRIPT_STATUS has its default setting of "0" (off, false, no), an ACCEPT FROM INPUT STATUS statement will return a fixed value when the program has redirected input. The value returned is the value of the **INPUT_STATUS_DEFAULT** configuration variable.

When SCRIPT_STATUS is not "0", and input is redirected, then ACCEPT FROM INPUT STATUS will return the actual status of the script file (i.e., it will return "1" (on, true, yes) unless the script file has been exhausted).

SCRN

This variable can be used in conjunction with the **SCREEN** variable to control many attributes of the video sub-system. For details, see the *ACUCOBOL-GT User's Guide*, **section 4.4.2, "The SCREEN Option."**

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

SCROLL

This variable affects when screen scrolling will occur. When it is set to "1" (on, true, yes), scrolling and cursor positioning occur normally. When it is set to "0" (off, false, no), screen scrolling will occur only as the result of a SCROLL phrase in an ACCEPT or DISPLAY statement, and any DISPLAY statement that references a line past the bottom of the current window will be ignored. ACCEPT statements that reference a line past the bottom of the current window will be placed in the home position of the window. The default setting is "1".

*server*_MAP_FILE

This variable is used to point to the character map file used for translating international character sets between a client machine and a specific server that uses different character codes. The map file is a simple text file that you create with an editor of your choice. Each line in the map file must contain two values in either decimal or hexadecimal: the character code of the character on the client machine, and the character code of the same character on the server. Use a # sign to indicate a comment.

The map file may be stored on either the client machine or the server machine.

The server specified in the configuration variable name must match the server specified in the remote name notation that points to the data files. For example, if you are using AcuServer to access remote files on a machine named sun3, you would use remote name notation to specify the directory that contains the data files. It might look like this:

```
@sun3:/user/acct/inventory
```

Then, create a map file and use this configuration variable to point to the map file:

```
sun3_map_file  @sun3:/user/acct/inventory/map.txt
```

If the map file is local, your value might look like this:

```
sun3_map_file  C:\user\utility\map.txt
```

If the map file is located on a server, you must have the AcuServer product on that server, to enable client access.

The runtime first searches for the configuration variable *server*_MAP_FILE and, if it is found, uses that setting to locate the map file. If that variable is not set, the runtime searches for **DEFAULT_MAP_FILE**. If that variable is also not set, then no character translation is done.

*server*_PASSWORD

Designed to be defined in the environment (rather than in the configuration file), *server*_PASSWORD and its mate **server_port_PASSWORD** make working with AcuServer easier when the compiler and **cblutil** are called repeatedly from the AcuBench integrated development environment. In this scenario, when one of these variable is used, the user never has to enter a password. When these variables are not used, if a password is required the user must provide it repeatedly.

Set *server*_PASSWORD to the value of the password. For example:

```
MERCURY_PASSWORD=we1rneB
```

where *server* is replaced by the name of the server.

The compiler checks the variable *server_port*_PASSWORD first. If it isn't defined, *server*_PASSWORD is checked. If *server*_PASSWORD is not defined, the user is prompted for a password. If either variable is defined, but the value does not match the value in the AcuAccess file, the connection attempt fails.

*server_port*_PASSWORD

Designed to be defined in the environment (rather than in the configuration file), *server_port*_PASSWORD and its mate **server_PASSWORD** make working with AcuServer easier when the compiler and **cblutil** are called repeatedly from the AcuBench integrated development environment. In this scenario, when one of these variable is used, the user never has to enter a password. When these variables are not used, if a password is required the user must provide it repeatedly.

Set *server_port*_PASSWORD when you want to connect to a server on a particular port. For example, to set a password to connect to a server named "MERCURY" that is listening on port 4330, you can set the following:

```
MERCURY_4330_PASSWORD=we1rneB
```

where *server* is replaced by the name of the server, and *port* is replaced by the port number that AcuServer is using.

The compiler checks the variable *server_port*_PASSWORD first. If it isn't defined, *server*_PASSWORD is checked. If *server*_PASSWORD is not defined, the user is prompted for a password. If either variable is defined, but the value does not match the value in the AcuAccess file, the connection attempt fails.

SHARED_CODE

For many UNIX machines, ACUCOBOL-GT supports the ability to have multiple users share the same copy of a COBOL program's object code in memory. This configuration variable indicates which programs you want to share code. Use of shared memory is recommended only if you have a problem with limited memory and excessive swapping. In this case, the advantage of reduced swapping will usually more than make up for the overhead added by sharing memory. To use shared code for all of your programs on UNIX, add the following line:

```
SHARED_CODE    1
```

This will cause all programs to attempt to share code. Every code segment loaded into memory will be placed into shared memory until shared memory is full. Further code segments will then be placed in conventional memory. If the system runs out of shared memory and the shared code requests start failing, each runtime will have its own copy of the program in its own memory space.

Since shared memory is a limited resource under UNIX, you will usually want to restrict the use of shared code to those programs where it will be most beneficial. This will ensure that other programs do not use up all of the available shared memory first. To do this, specify each program you want to share as follows:

```
SHARED_CODE  Program1
SHARED_CODE  Program2
SHARED_CODE  Program3
```

(The program name may also be enclosed in single or double quotes, for example, "Program1" or 'Program2'.) When you use this method, "Program1", "Program2", and so forth are the PROGRAM-IDs from the

programs' Identification Divisions (note that a program's object file name is *not* used). If you use this method, then setting SHARED_CODE to "1" will have no effect.

For additional information, see the *ACUCOBOL-GT User's Guide*, **section 2.12, "acushare Utility Program."**

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

SHARED_LIBRARY_EXTENSION

This variable allows you to define the filename extension for UNIX/Linux shared object libraries. The default value is ".so". This variable has meaning only on systems that support UNIX shared libraries.

SHARED_LIBRARY_LIST

This variable allows you to specify the names of UNIX/Linux shared object libraries or Windows DLLs.

SHARED_LIBRARY_LIST can be set in one of three ways:

1. In the environment
2. In the runtime configuration file
3. Programmatically with the SET ENVIRONMENT statement

The runtime loads the listed objects on program startup or as the result of a SET ENVIRONMENT statement. Names must be delimited by spaces, colons (UNIX/Linux), or semicolons (Windows).

With DLLs, you can specify both the name of the DLL and the calling convention to use. Any calling convention specified this way overrides the DLL_CONVENTION variable setting. For information about specifying

DLLs and calling conventions, see section 3.3.2, “Loading DLLs with Configuration Variables,” in *A Guide to Interoperating with ACUCOBOL-GT.*

You can also list objects without path information and use the **SHARED_LIBRARY_PREFIX** configuration variable to specify a set of directories that the runtime will search when attempting to load a shared library.

Once loaded, functions exported by these libraries can be called directly.

The SET ENVIRONMENT statement can be used to set SHARED_LIBRARY_LIST any number of times during program execution. Each time it is set, the runtime loads the libraries listed. Previously loaded libraries remain loaded. Libraries loaded with SHARED_LIBRARY_LIST remain in memory until the process exits. The CANCEL statement cannot be used to unload the library.

On some systems, such as AIX, if the shared module is a member of an archive, you must specify the name of the member in parentheses after the name of the archive. For example:

```
SHARED_LIBRARY_LIST="/usr/opt/db2_08_01/lib/libdb2.a(shr.o)"
```

SHARED_LIBRARY_LIST is like the runtime “-y” option, except that it does not require setting the SHARED_LIBRARY_EXTENSION variable, and unlike “-y”, you can mix “.a” and “.so” libraries in the list.

Note: The SHARED_LIBRARY_LIST configuration variable does not load client-side DLLs for thin client applications that make calls using the CALL verb “@[DISPLAY]:” syntax. These applications must explicitly load the DLL by calling it with the CALL verb before calling a function within the DLL.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

SHARED_LIBRARY_PREFIX

This variable allows you to specify a set of directories that the runtime will search when attempting to locate a shared library. The format of the value is the same as that allowed for **FILE_PREFIX**. You can set SHARED_LIBRARY_PREFIX in the configuration file, environment, or programmatically with the SET verb.

The default value on Windows systems is empty.

The default value on UNIX and Linux systems is "/opt/acucorp/8xx/lib /opt/acu/lib". This helps the runtime find "libclnt.so" (or "libclnt.sl") when the operating system's shared library environment variable (e.g., LIBPATH, LD_LIBRARY_PATH, SHLIB_PATH, etc.) is not set.

SHUTDOWN_MESSAGE_BOX

This variable allows you to specify whether or not you want the runtime's shutdown message to be displayed in a message box. If this variable is set to "0" (off, false, no), the runtime will display the shutdown message to the screen without a message box. The default value is "1" (on, true, yes).

SORT_DIR

This variable allows you to place temporary files used by the SORT verb in another directory. By default these files are stored in the current directory. You can specify an alternate directory to hold the sort files by setting the configuration variable SORT_DIR to the desired directory. This value is treated as a prefix, much like FILE_PREFIX. You can improve the performance of the SORT verb by placing the temporary files on a fast device. Take care, however, that the device has enough free space to hold twice the size of the data to be sorted.

You may not use the SORT_DIR variable with AcuServer.

Remote name notation is allowed for the SORT_DIR variable if your runtime is client-enabled. See *ACUCOBOL-GT User's Guide* sections 5.2.1 and 5.2.2 for more information about client-enabled runtimes and remote name notation.

SORT_FILES

This configuration variable sets the number of temporary files used by SORT. The acceptable range is from 4 to 64. The default value is "8".

Increasing the number of files used will usually improve SORT performance, particularly for large sorts. Note that you must have enough available file handles to open all of the temporary files concurrently. In general, the number of files available is more important than the amount of memory used. If you are experiencing long sorts, try increasing the number of files before you increase the amount of sort memory.

The SORT verb removes all of its temporary files, except for one, prior to beginning its output phase.

SORT_MEMORY

This variable specifies the number of 64 KB blocks of memory that the SORT verb will try to allocate when it executes. The acceptable range is from 1 to 16384. The default value is "32". Using a value lower than the default can be useful if memory is tight on the host machine. Using a higher value may enhance SORT performance.

Take care, when increasing the SORT_MEMORY setting, to ensure that you do not assign too much memory to the runtime. For most operating systems, the memory used by SORT is not returned to the system. While the runtime may use the memory for other purposes, this memory is not available to other programs until the runtime exits.

The SORT verb will attempt to allocate the amount of memory specified in SORT_MEMORY. If the requested amount is not available, the runtime will return an out of memory error.

SPACES_ZERO

This configuration variable applies only to object files generated with ACUCOBOL-85 Version 1.5 and earlier. For later object files, use the "-Zz" compiler option. When SPACES_ZERO is "1" (on, true, yes), it alters the method in which USAGE DISPLAY data items are used by the runtime system. The main effect is that, in most cases, a data item containing spaces will be treated as if it contained zeros. Note that this may not occur in all instances because the ACUCOBOL-GT compiler may construct code that directly acts on a data item without first converting it to a number. The default value is "0" (off, false, no).

SPOOL_FILE

This configuration variable allows you to hold a *pipe* open when you close the named file with the CLOSE WITH NO REWIND verb. This enables you to gather multiple reports into a single job for the print spooler. For additional details about pipes and file name interpretation, see the *ACUCOBOL-GT User's Guide*, **section 2.9, "File Name Interpretation."**

The value given to the SPOOL_FILE variable must be the ASSIGN name of a sequential file that has been attached to a pipe. The pipe must be attached to the ASSIGN name in the COBOL configuration file via the "-P" option. For example, suppose you have a file defined as follows:

```
SELECT PRINT_FILE
ASSIGN TO PRINT "PRINTER"
```

and that you have the following pipe defined in the COBOL configuration file:

```
PRINTER  -P  lp  -s
```

Then, to specify that you want to keep the pipe open when the file is closed WITH NO REWIND, you would add the following line to the COBOL configuration file:

```
SPOOL_FILE PRINTER
```

The name specified for SPOOL_FILE is processed in the same way as the external file name specified in the file's ASSIGN clause.

When the corresponding file is closed with a NO REWIND option, the pipe is not closed. Instead, if the file is later opened again, the same pipe is used. The pipe is not closed until a CLOSE verb without the NO REWIND option is executed on that file, or until the run unit finishes. Only one pipe can be held open in this manner.

STD_FIXED_FONT

This configuration variable allows you to set the standard font used by the Windows version of the runtime. It can be set in the configuration file to one of the following values:

-1 (default) The web runtime uses ANSI_FIXED_FONT. Other instances of the runtime use SYSTEM_FIXED_FONT.

0 All runtimes use ANSI_FIXED_FONT for the standard font.

1 All runtimes use SYSTEM_FIXED_FONT for the standard font.

2 All runtimes use OEM_FIXED_FONT for the standard font.

If this variable has not been set, or has an invalid value, it will default to “-1”.

STOP_RUN_ROLLBACK

When this variable is set to “1” (on, true, yes), the system performs an implied ROLLBACK rather than a COMMIT after a STOP RUN.

With a “0” (off, false, no) setting for this variable, the system performs an implied COMMIT after a STOP RUN. The default value for this variable is “0”.

STRIP_TRAILING_SPACES

This variable provides an alternate method for determining which LINE SEQUENTIAL files will have trailing spaces removed from records written to them. At the time a LINE SEQUENTIAL file is opened, the value of this variable is examined. If this variable is "1" (on, true, yes), then automatic space suppression is applied to this file.

Otherwise, the file is processed according to the normal rules, as described in the *ACUCOBOL-GT User's Guide*, **section 6.1.1, "Sequential Files."** The default value for this variable is "0" (off, false, no).

Note that a related configuration variable is the **EXTFH_KEEP_TRAILING_SPACES** variable.

SWITCH_PERIOD

This variable helps determine how frequently threads switch control. When a thread executes SWITCH_PERIOD number of selected operations, the threads switch control. The selected operations are generally comparisons. Comparison operations are used to cause compute-bound threads to switch.

Setting the value of SWITCH_PERIOD lower will increase the overhead spent switching threads, but increase the uniformity of thread execution. Setting the value very low can significantly hurt performance. The default value is "100".

SYSINTR_NAME

This variable defines the location of the SYSINTR file that may be used with MPE emulation software. This variable must be specified with HFS syntax and set to the full path of the SYSINTR file. For example:

```
SYSINTR_NAME /opt/mpux/etc/sysintr.txt
```

TC_AUTO_UPDATE_FAILED_MESSAGE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. If the thin client automatic update process fails for any reason, a message box may appear informing the user of the failure. The TC_AUTO_UPDATE_FAILED_MESSAGE configuration variable lets you specify the text in this message box. Its default value is

```
ACUCOBOL-GT Thin Client: Automatic update was
unsuccessful
```

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide.*

TC_AUTO_UPDATE_FAILED_TITLE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. If the thin client automatic update process fails for any reason, a message box may appear informing the user of the failure. The TC_AUTO_UPDATE_FAILED_TITLE configuration variable lets you set the title bar text for this message box. Its default value is

```
ACUCOBOL-GT Thin Client
```

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide.*

TC_AUTO_UPDATE_NOTIFY_FAIL

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. If the thin client automatic update process fails for any reason, a message box may appear informing the user of the failure. If you do not want the thin client to inform the user that the automatic update has failed, set the TC_AUTO_UPDATE_NOTIFY_FAIL configuration variable to "false" (0, off, no). The default value of this variable is "true" (1, on, yes).

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_AUTO_UPDATE_QUERY

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. When an event triggers the update process, the thin client displays a message box informing the user that an upgrade is required. The default setting of “1” (on, true, yes) for the TC_AUTO_UPDATE_QUERY configuration variable enables the display of that message box. Setting this variable to “0” (off, false, no) prevents the message box from appearing.

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_AUTO_UPDATE_QUERY_MESSAGE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. When an event triggers the update process, the thin client displays a message box informing the user that an upgrade is required. The value of the TC_AUTO_UPDATE_QUERY_MESSAGE configuration variable determines the message displayed in that message box. The default value of the variable depends on the circumstances that triggered the automatic update. For example, if the automatic update is initiated by a version or protocol number mismatch, the default message displayed is:

```
Incompatible server version
Server version: <srvvers>, client <clntvers>
Server protocol: <srvproto>, client <clntproto>
Press OK to automatically correct this problem
```

where <*srvvers*>, <*clntvers*>, <*srvproto*>, and <*clntproto*> are replaced by the server version, client version, server protocol number, and client protocol number, respectively.

For detailed information about other default values for this configuration variable and about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide*.

TC_AUTO_UPDATE_QUERY_TITLE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. When an event triggers the update process, the thin client displays a message box informing the user that an upgrade is required. You use the TC_AUTO_UPDATE_QUERY_TITLE configuration variable to specify the title bar text in that message box. The default value of this variable is

```
ACUCOBOL-GT Thin Client
```

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_AX_EVENT_LIST

In thin client deployments, this variable lets you control which events your program receives, giving you more control over the volume of network traffic. It must be set in the configuration file and cannot be changed programmatically with the SET verb. It contains the numeric value of a single .NET or ActiveX event type or a list of .NET or ActiveX event types separated by non-numeric characters like spaces or commas. Whether your program receives these events depends on the value of **TC_EXCLUDE_EVENT_LIST**. If its value is “0”, then your program receives the events listed in TC_AX_EVENT_LIST. If TC_EXCLUDE_EVENT_LIST is set to “1”, then the events listed in TC_AX_EVENT_LIST are not sent to your program.

TC_CHECK_ALIVE_INTERVAL

This variable allows you to set a time interval in seconds (a value between "1" and "32767") during which the server runtime checks for thin client activity. This activity can be either regular thin client user interaction or, if the user interface is inactive, two "ping" messages sent by the thin client during the defined interval. If no thin client activity of any kind occurs during a particular interval, the server runtime process exits. Setting this variable to "0" disables the client activity check feature. The default value is "300" (5 minutes).

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_CHECK_INSTALLER_TIMESTAMP

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. The value of the TC_CHECK_INSTALLER_TIMESTAMP configuration variable determines whether the thin client compares the modification times of the installer files on the client and on the server. If this variable is set to "1" (on, true, yes) and the modification time of the client file is older than the time of the server file, the automatic update process is initiated. If the installer file does not exist on the client, then the comparison is made with the modification time of the thin client executable (**acuthin**) currently running. The default value for this variable is "0" (off, false, no).

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_CONTINUITY_WINDOW

If this configuration variable is set to "1" (on, true, yes), the Thin Client creates an invisible window after the next window is created by the COBOL application. This invisible window remains until the Thin Client shuts down.

This variable must be set in the application's configuration file or in the initialization code of the application so that it is applied to the initial window created by the application.

The invisible window is needed because the GetMessage API routine behaves differently under Windows 2000 and Windows XP than in older versions of Windows. If an application has no open windows and it calls GetMessage, the focus is transferred to another application. Once transferred, the application cannot force the focus to return to the original application, even if it subsequently creates a window to receive the focus. This situation arises under the Thin Client if the COBOL application destroys all of its windows before creating a new one.

Since this variable applies to a very specific situation, the default setting for TC_CONTINUITY_WINDOW is “0” (off, false, no). You must set it explicitly if you want to use this feature.

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_CONTROL_SYNC_LEVEL

This variable determines which VALUE data items in a Screen Section are updated when a BEFORE, AFTER or EXCEPTION procedure executes. (This variable only affects BEFORE, AFTER and EXCEPTION procedures. The values of all variables are made current anytime an ACCEPT terminates.) The possible values for TC_CONTROL_SYNC_LEVEL are:

1	(default) Only the VALUE data item associated with the current field is updated when its AFTER or EXCEPTION procedure executes.
2	Only the VALUE data item associated with the current field is updated when its BEFORE, AFTER or EXCEPTION procedure executes.
3	All VALUE data items are updated when executing any BEFORE, AFTER or EXCEPTION procedure.

For best performance, we recommend leaving this variable at its default setting of “1” unless that causes your program to perform incorrectly. In which case, you can increase the setting of TC_CONTROL_SYNC_LEVEL to “2” or “3” to adjust for problems in the application behavior.

Note: Alternatively, you can directly INQUIRE the value of a control in an embedded procedure. This allows you to tune application performance more precisely than TC_CONTROL_SYNC_LEVEL will allow.

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_DELAY_ACTIVATE

This variable determines precisely when the thin client sends CMD-ACTIVATE events to the server. Under the default setting of “1” (on, true, yes), the client delays sending the event until after the Windows notification routine receiving the event has completed. However, ActiveX events are never delayed. The alternate setting of “0” (off, false, no) sends the event to the server immediately when it is generated on the client.

We recommend leaving this variable at its default setting because the Windows API occasionally alters actions taken by the program when they occur within the scope of an activation notification. (For example, Windows will sometimes override a “set focus” call.) Delaying the COBOL program's response to the activation until after the Windows notification routine is complete avoids these alterations.

If you experience an unexplained difference in window activation when running under the thin client, try setting this variable to “0”. If this produces the desired behavior, the handling of the CMD-ACTIVATE events by the program is unusual and may not be performing as intended. For example, the EVENT procedure that handles the CMD-ACTIVATE event may be destroying an unrelated window instead of transferring control to the window issuing the CMD-ACTIVATE event.

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_DELAY_PRE_EVENT_OPS

This configuration variable applies only to the ACUCOBOL-GT Thin Client. Using this variable, you can direct the thin client to buffer some requests received from the server and process them later. When you set this variable to "1", the thin client buffers the requests received between the time that the client sends an event to the server and the time that the server informs the client that it has started the related event procedure. The events are processed only after the event procedure starts in order to prevent the thin client from processing requests that generate more events before the first event procedure has started. The default value of TC_DELAY_PRE_EVENT_OPS is "0".

Note: The buffering behavior described for this configuration variable was introduced as the default behavior in Version 6.1. Beginning with Version 7.2, the buffering behavior is turned off by default.

TC_DISABLE_AUTO_UPDATE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. You can disable the automatic update process by setting the TC_DISABLE_AUTO_UPDATE configuration variable to "1" (on, true, yes). The default value of this variable is "0" (off, false, no).

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide.*

TC_DISABLE_SERVER_LOG

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. If the thin client automatic update process fails for any reason, a log file may be created on the server. This file contains a log of the update operations and details about the failure. To prevent the creation of this log file, set the TC_DISABLE_SERVER_LOG configuration variable to "true" (1, on, yes). The default value of this variable is "false" (0, off, no).

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_DOWNLOAD_CANCEL_MESSAGE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. During the automatic update installer file download process, a progress dialog appears. You can cancel the download at any time from this dialog box. Use the TC_DOWNLOAD_CANCEL_MESSAGE configuration variable to specify the message that appears when the download is cancelled. The default value for this variable is

```
Please wait while the download is being cancelled . . .
```

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_DOWNLOAD_DESCRIPTION

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. During the automatic update installer file download process, a progress dialog appears. You use the TC_DOWNLOAD_DESCRIPTION configuration variable to specify the text that appears in the middle of the download progress dialog. Its default value is

```
Downloading installation file. . .
```

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_DOWNLOAD_DIALOG

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. During the automatic update installer file download process, a progress dialog appears. The default value of "1" (on, true, yes) for the TC_DOWNLOAD_DIALOG configuration variable allows the appearance of this dialog box. If you set this variable to "0" (off, false, no), the progress dialog does not appear.

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_DOWNLOAD_DIALOG_TITLE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. During the automatic update installer file download process, a progress dialog appears. The TC_DOWNLOAD_DIALOG_TITLE configuration variable is used to specify the title bar text in this dialog box. The default value of this variable is

```
ACUCOBOL-GT Thin Client Automatic Update
```

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_EVENT_LIST

This configuration variable lets you control which events your program receives, giving you more control over the rate of network traffic. It must be set in the configuration file and cannot be changed programmatically with the SET verb. It contains the numeric value of a single event type or a list of event types separated by non-numeric characters like spaces or commas. Whether your program receives these events depends on the value of **TC_EXCLUDE_EVENT_LIST**. If its value is "0", then your program

receives the events listed in TC_EVENT_LIST. If TC_EXCLUDE_EVENT_LIST is set to “1”, the events listed in TC_EVENT_LIST are not sent to your program.

TC_EXCLUDE_EVENT_LIST

The value of this variable determines whether the events listed in **TC_AX_EVENT_LIST** and **TC_EVENT_LIST** are sent to your program. A value of “1” means the specified events are not sent to your program. The default value is “0”.

TC_INSTALLER_ARGS

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. The thin client uses the value of the TC_INSTALLER_ARGS configuration variable as the command-line options passed to the installer executable. For example, if you want “msiexec.exe” to log all of its operations to a file named “msi.log”, then you could set TC_INSTALLER_ARGS to “/log msi.log”. TC_INSTALLER_ARGS has no default value.

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide*.

TC_INSTALLER_CLIENT_FILE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. You use the TC_INSTALLER_CLIENT_FILE configuration variable to specify the path and file name of the installer file that you want to create on the client. The default value of this variable is

```
<APPDATA>\ACUCOBOL-GT\<installer_server_filename>
```

where <APPDATA> is a special directory name for C:\Documents and Settings\<user>\Application Data and <*installer_server_filename*> is the file name specified in the **TC_INSTALLER_SERVER_FILE** configuration variable.

For detailed information about special directory names like <APPDATA> and about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_INSTALLER_RUN_ASYNC

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. You use the TC_INSTALLER_RUN_ASYNC configuration variable when you want to prevent the thin client from restarting after an automatic update or when your installer file handles the automatic update process to completion. When you set this variable to "1" (on, true, yes), the thin client starts the installer process asynchronously and then exits immediately. It does not wait for the automatic update process to complete and does not restart the application. The default value is "0" (off, false, no).

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_INSTALLER_SERVER_FILE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. You set the TC_INSTALLER_SERVER_FILE configuration variable to the path and file name of the server installer file. Its default value is

```
<runtime_path>/acuthin.msi
```

where <*runtime_path*> is the directory that contains the **runcbl** runtime executable.

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_INSTALLER_TARGET_DIR

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. You use the TC_INSTALLER_TARGET_DIR configuration variable to specify the location where you want the updated thin client to be installed. This variable has no default value.

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_INSTALLER_UI_LEVEL

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. The keywords or numeric values in the TC_INSTALLER_UI_LEVEL configuration variable control the Windows installer interface. Set TC_INSTALLER_UI_LEVEL to NONE or “0” if you do not want the Windows installer to display a user interface. Set this variable to UNATTENDED or “1” if you want the Windows installer to display informational and progress messages but to execute unattended. Set the variable to INTERACTIVE, DEFAULT, or “2” if you want the Windows installer to prompt for and accept user input for the installation process. Set the variable to REDUCED or “3” if you want to use a reduced user interface.

For detailed information about the thin client automatic update process, refer to section 7.4, “Thin Client Automatic Update,” in the *AcuConnect User’s Guide.*

TC_MAP_FILE

In thin client deployments, set this variable to point to the character map file that defines the mapping of international characters between client and server systems. A detailed description of international character handling is located in the *AcuConnect User's Guide*, section 4.5, "International Character Handling."

TC_NESTED_AX_EVENTS

This variable determines how thin client handles nested ActiveX events. Because thin client processes Windows messages while waiting for responses from the server, it is possible for new ActiveX events to be sent while still waiting for an earlier event procedure to return, causing event procedures to be nested within event procedures. Because nested event procedures can cause unpredictable results, including memory access violations (MAVs), this variable is set to "0" (off, false, no) by default. If you want to enable it, set it to "1" (on, true, yes).

TC_QUIT_MODE

This variable lets you control how your COBOL application shuts down when no client activity occurs during the interval defined by **TC_CHECK_ALIVE_INTERVAL**. Setting TC_QUIT_MODE to "-1" (the default value) shuts your program down according to the value chosen for the **QUIT_MODE** configuration variable. If you set this variable to "0", the runtime stops the program immediately.

When this variable is set to a value greater than "0" (up to "32767"), your application has a program-controlled exit. When the runtime determines that the thin client is no longer responding (no user interaction and no pings during **TC_CHECK_ALIVE_INTERVAL**), the MSG-MENU-INPUT event is sent to the program's main window and EVENT-DATA-2 contains the value defined by TC_QUIT_MODE. Your program can detect this in the main window's event procedure and you can perform whatever code you desire. At this point there is no connection to the thin client, so user interface operations may not be performed. You must end your shutdown code with "STOP RUN" to terminate the runtime.

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_REQUIRES_BUILD_NUMBER

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. When the thin client executes, it compares its build number with the value of the TC_REQUIRES_BUILD_NUMBER configuration variable. If the value of this variable does not match the client's build number, the automatic update process is initiated. Set this variable to the thin client build number required by the application. The default value of this variable is "0" (off, false, no).

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide*.

TC_RESTRICT_AX_EVENTS

This variable controls whether the application will ignore ActiveX events between ACCEPT statements (the termination of one ACCEPT and the beginning of the next). Setting this variable to "1" (on, true, yes) enables this behavior. The default value is "0" (off, false, no).

Ordinarily, the thin client runtime suspends all ActiveX events when the application is not processing an ACCEPT statement. However, some ActiveX controls do not support the ability to suspend and resume events when an application is not processing an ACCEPT statement. As a result, in a thin client environment, an event procedure may be run unexpectedly during a CREATE, DISPLAY, MODIFY, INQUIRE, or any other operation that waits for results from the thin client. Setting TC_RESTRICT_AX_EVENTS provides some control over these ActiveX events.

To determine if a particular ActiveX control supports suspending and resuming events, check the control's documentation or ask the control vendor. Note that the control must implement the "IOleControl::FreezeEvents()" function.

For more information about ActiveX control handling, see Chapter 4 in *A Guide to Interoperating with ACUCOBOL-GT*, and section 6.3 of the *AcuConnect User's Guide*.

TC_SERVER_LOG_FILE

This configuration variable applies only to the ACUCOBOL-GT Thin Client automatic update feature. If the thin client automatic update process fails for any reason, a log file may be created on the server. This file contains a log of the update operations and details about the failure. The TC_SERVER_LOG_FILE configuration variable can be used to configure the location and name of that log file. The name can optionally include the hostname of the client machine and the process ID of the server runtime that was managing the automatic update at the time of the failure.

By default, this file is named "autoupdate.%c.%p.log", where "%c" is replaced by the client hostname and "%p" is replaced by the process ID of the server runtime. The default location is the working directory specified in the alias on the server. Note that the directory must exist at the time of the failure for the log file to be created.

For detailed information about the thin client automatic update process, refer to section 7.4, "Thin Client Automatic Update," in the *AcuConnect User's Guide.*

TC_SERVER_TIMEOUT

This variable lets you determine how many seconds (from "0" to "32767") the thin client waits for a response from the server. If the thin client receives no response from the server in the specified time period, the following message box appears:

```
The remote host is not responding.
Press OK to close this program.
Press Cancel to wait another %s seconds.
```

where "%s" is the value of TC_SERVER_TIMEOUT. The default value is "20".

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TC_TV_SELCHANGING

This variable is designed for thin client applications. It provides some control over when the runtime generates Msg-Tv-Selchanging events for tree view controls. Because most applications that use tree view controls do not process Msg-Tv-Selchanging events, the thin client suppresses its generation in some cases. This improves both performance and stability. TC_TV_SELCHANGING recognizes the following values:

0 never generate Msg-Tv-Selchanging events

1 (default) generate Msg-Tv-Selchanging events when the selection is about to change due to the user using the mouse or the keyboard to change to current selection

2 always generate Msg-Tv-Selchanging events

The default setting of “1” allows you to detect user-initiated events in your program while filtering out many other causes of the event.

If you know your program doesn’t handle any Msg-Tv-Selchanging events, you can set TC_TV_SELCHANGING to “0” to entirely inhibit generation of the event. This can slightly improve performance.

If TC_TV_SELCHANGING is set to “1” and your program experiences odd behavior with tree view controls under the thin client, you can try setting the variable to “2” to generate all Msg-Tv-Selchanging events. This setting can help you determine whether a Msg-Tv-Selchanging event is the cause of the odd behavior. If this setting eliminates the odd behavior, it indicates that your program relies on Msg-Tv-Selchanging events in cases other than the user initiating a selection change.

For more information about the thin client, refer to the *AcuConnect User's Guide*.

TEMP_DIR

This variable lets you specify where certain temporary files used by the ASSIGN clause will be created on VAX/VMS systems. These temporary files are created when you use the %TMP% option for assigning a file to a simulated pipe with "-P". For more information, see the *ACUCOBOL-GT User's Guide*, **section 2.9, "File Name Interpretation."**

TEMPORARY_CONTROLS

By default, graphical controls are created as permanent controls. By setting this configuration variable to "1" (on, true, yes), you cause controls to be created *temporary* by default. This is useful when you are converting older programs that assume that a screen update will overwrite any existing screen data. You can make individual controls permanent or temporary explicitly by using the PERMANENT and TEMPORARY styles (see **Section 5.2** in Book 2, *ACUCOBOL-GT User Interface Programming*).

TEXT

This configuration variable controls the text of runtime messages. The ACUCOBOL-GT runtime system displays a number of informational and warning messages to the end user. Several of these messages can be customized via entries in the configuration file.

For each message that you want to change, place the word "TEXT" in your configuration file, followed by a message number from the list below, an "=" sign, and then the text you would like to use.

For example, the standard message #1 is "press return". You can change that message to "push enter" by placing this line in your configuration file:

```
TEXT  1=push enter
```

Note: There is no space before or after the equals sign, and the new message is not in quotes.

These are the standard runtime messages and their numbers:

Message #	Text
1	“Press return”
2	“Number required”
3	“Entry required”
4	“Field must be filled with data”
5	“Too many hot keys active”
6	“Program missing or inaccessible”
7	“Not a COBOL program”
8	“Corrupted program”
9	“Inadequate memory available”
10	“Unsupported version of object code”
11	“Program already in use”
12	“Too many external segments”
13	“Large-model program not supported”
18	“Please end this application first”
19	“Japanese objects not supported”
	This message is displayed when a standard runtime attempts to execute an object that contains Japanese COBOL extensions.
20	“Too many lines”
	This message is displayed when the user exceeds the MAX-LINES setting for a multiline entry field.
21	“License manager (acushare) not running”
	This message is displayed when acushare is not running and the runtime is unable to start it (e.g., because it is not in the path).
22	“Data must fit this format:”
	This message is displayed when the user enters illegal data when using the NUMERIC_VALIDATION configuration option.
23	“&Ok”
24	“&Yes”

Message #	Text
25	"&No"
26	"&Cancel" Messages 23, 24, 25, and 26 are used by character-based versions for the message box facility.
28	"Unable to access the file "%s" due to heavy usage by other users. Would you like to continue waiting for it?" (See the configuration variable WAIT_FOR_FILE_ACCESS for more information about this message.)
30	"Connection refused - perhaps AcuConnect is not running"
31	"Please enter a value between %ld and %ld" This message is displayed when the user enters a value outside of the allowed range for an entry-field (see MIN-VAL/MAX-VAL in the entry-field reference). The first "%ld" is replaced by the MIN-VAL setting. The second "%ld" is replaced by the MAX-VAL setting. You may omit these if you desire. Note that the second character in the sequence is the letter "l", and not the number one ("1").
32	"Program contains object code for a different processor"
33	"Incorrect serial number"
34	"Connection refused - user count exceeded on remote server"
35	"License error"
36	"The remote host is not responding.\nPress OK to close this program.\nPress Cancel to wait another %s seconds. This message is displayed when Thin Client does not receive a response from the server in the number of seconds specified in TC_SERVER_TIMEOUT. Use "\n" to separate lines and "%s" to substitute the number of seconds (value of TC_SERVER_TIMEOUT). If you don't want to display the number of seconds, omit the "%s".

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

TRACE_STYLE

This variable allows you to customize the format of error and trace messages. You can set it to the sum of one or more of the following values:

0	The default. No “ACU” prefix, process ID, time, or date is included in the trace output.
1	Adds “ACU” prefix to each line of the trace output.
2	Adds the process ID.
4	Adds the time.
8	Adds the microseconds; has an effect only if “4” is also specified.
16	Adds the date.

You can also set TRACE_STYLE to one of the following keywords, which correspond to the indicated numerical values:

NONE	0
TIMESTAMP	12 -- The TIMESTAMP style is 4+8; it outputs timestamps with microseconds.
APPSERVER	23 -- The APPSERVER style is 1+2+4+16; at the beginning of each line of the error file it outputs “ACU” followed by the date, the process ID, and the time without microseconds.

TRANSLATE_TO_ANSI

This variable has meaning only on graphical systems such as Windows. It is used only if:

- you are using the graphical system’s font to accept data, and
- you store your data using the OEM character set. (For example, Vision files may contain OEM characters if they were created with a DOS runtime.)

Set the variable TRANSLATE_TO_ANSI to “1” (on, true, yes) to turn on a character set translator. Then, if you use the graphical system’s font for accepting data, the runtime will translate from one character set to the other for you. Data that is accepted from the screen will be translated into the OEM character set before it is stored on disk. Data stored in the OEM character set will be translated to the ANSI character set before it is displayed on screen. This also applies to the printer, if you are using Windows spooling and the printer uses an ANSI font.

Setting TRANSLATE_TO_ANSI to the default, “0” (off, false, no), turns off the translation process.

This variable can be set from within a COBOL program with the SET verb. For example:

```
SET ENVIRONMENT “TRANSLATE_TO_ANSI” TO “YES”.
```

Note on ANSI and OEM characters:

The ANSI and OEM representations of the following standard English characters are identical:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Abcdefghijklmnopqrstuvwxyz
0123456789 <space>
! " # $ % & ' ( ) * + , - . / :
; < = > ? @ [ \ ] ^ _ ‘ { | } ~
```

Only the representations of accented vowels and other special or non-English characters are different.

TREE_ROOT_SPACE

This variable controls the number of screen columns between the left edge of the Tree-View control and the root level text. TREE_ROOT_SPACE is used only with the LINES-AT-ROOT property. If LINES-AT-ROOT is not specified, the root level item text will be displayed starting at the leftmost screen column inside the tree-view control.

For example, if TREE_ROOT_SPACE is set to 5, there will be 5 screen columns before the text of each root level item. The screen column where the root level line will be drawn is determined by this formula:

```
root level-line = (TREE_ROOT_SPACE - 1)/2 + 1
```

Taking off from the previous example, if TREE_ROOT_SPACE=5, the root level line will be drawn in screen column 3, counting from the left edge of the Tree-View control.

This has the effect of centering the vertical root level line in the space between the left edge of the Tree-View control and the last root level text.

The "+" or "-" button is displayed in the column to the right of this vertical line if the TREE_ROOT_SPACE is set to a value greater than or equal to 2. If the TREE_ROOT_SPACE is set to 1, the "+" or "-" button appears in the first screen column of the Tree-View control. The default value of TREE_ROOT_SPACE is 2.

TREE_TAB_SIZE

This configuration variable is one of two variables that affect the appearance of character-based Tree-View controls. TREE_TAB_SIZE controls the number of screen columns between each level in the visual representation of the tree. For example, if TREE_TAB_SIZE is set to 10, the horizontal distance between the first character of text in the first level and the first character of text in the succeeding levels of the tree will be 10 screen columns each. The default value of TREE_TAB_SIZE is 3.

See **TREE_ROOT_SPACE** variable.

TRX_HOLDS_LOCKS

This configuration variable allows you to control which locks are released at the end of a transaction. If this variable is set to "1" (on, true, yes), then locks set using the READ statement that are not specifically released or replaced by extended transaction locks (for example, by a REWRITE) are held at the end

of the transaction. Locks are released during a transaction by any operation that would ordinarily release them, unless those locks were replaced by extended transaction locks.

If TRX_HOLDS_LOCKS is set to the default, "0" (off, false, no), then locks are released at the end of a transaction, and the UNLOCK verb has no effect during a transaction.

UPPER_LOWER_MAP

This variable allows you to define which upper-case characters correspond to which lower-case characters, for characters outside of the standard ASCII character set (those whose underlying decimal values are 128 or larger).

You might find this useful if you are experiencing problems with the UPPER or LOWER option of the ACCEPT statement when non-standard characters are entered (such as an "e" with an accent above it). The ACUCOBOL-GT runtime system relies heavily on C library routines to handle conversions between upper-case and lower-case characters. On many machines, these routines do not handle characters outside of the standard ASCII character set correctly.

To specify corresponding characters, use UPPER_LOWER_MAP followed by pairs of characters, where the first character is the upper-case version and the second character is the lower-case version. Separate the characters by a space. Describe the characters either by typing them at the keyboard or by entering the decimal value that represents them.

For example, on a standard IBM PC, the video card represents an upper-case "U" with an umlaut (Ü) as character 154, and the lower-case "u" with an umlaut (ü) as 129. The upper-case "E" with an accent character is 144 (É) and the lower-case "e" with an accent is 130 (é). To express this in the configuration file, you would add the following line:

```
UPPER_LOWER_MAP  154 129 144 130
```

This could be extended to include all of the character pairs available.

By default, Windows systems come with the UPPER_LOWER_MAP defined to be the character pairs available on the standard video cards produced by IBM. Note that using "code pages" can change this, so the default may not work in all cases for these machines. For other machines, the default is empty (which means that C library routines are used for conversion). If you experience difficulties, UPPER_LOWER_MAP allows you to define a mapping that reflects your hardware configuration.

Only characters whose decimal values are 128 or greater may be mapped by this technique.

Note: This variable *cannot* be read with the ACCEPT FROM ENVIRONMENT statement.

USE_CICS

Set this variable to indicate to the runtime that the program makes calls to the CICS interface. When USE_CICS is set to "1" (on, true, yes), the runtime attempts to pass calls to functions that begin with the string "CICS" to the CICS interface. If the named routine does not exist, the runtime uses the normal search sequence to find a matching function. When USE_CICS is set to the default value of "0" (off, false, no), the runtime does not perform any special handling.

USE_EXECUTABLE_MEMORY

When set to "TRUE", this variable enables a COBOL program compiled for Native Code (-n compiler option) to run on a Windows machine that has Data Execution Protection (DEP) enabled for all processes. The default value is "FALSE".

USE_EXTSM

Set this variable to indicate that the runtime should use an external sort module. When USE_EXTSM is set to "1" (on, true, yes), the runtime uses the linked-in EXTSM function to perform the SORT or MERGE operation. When USE_EXTSM is set to the default value of "0" (off, false, no), the runtime does not perform any special handling for SORT and MERGE verbs.

USE_LARGE_FILE_API

On UNIX systems, this variable allows you to turn on or off file system API support for very large files (greater than 2 gigabytes). Support for large files is enabled when USE_LARGE_FILE_API is set to "1" (on, true, yes). Some UNIX systems do not support files greater than 2 gigabytes in size. In those situations, setting this variable to the default of "0" (off, false, no) causes the runtime to use the standard 32-bit file system API. This variable has no effect on Windows platforms.

USE_LOCAL_SERVER

This variable is used by the runtime and Web Runtime to specify whether or not you want to run client applications on the same machine as an AcuServer file server. When USE_LOCAL_SERVER is set to the default of "0" (off, false, no), AcuServer is bypassed when accessing local files that have remote name notation. The remote name is stripped off and the file I/O operation is handled by the runtime or Web Runtime. Set this variable to "1" (on true, yes) to use AcuServer to access local files that have remote name notation. This variable only works with AcuServer client runtimes and AcuServer client Web Runtimes.

USE_MPE_REDIRECTION

This configuration variable applies only when running in HP COBOL compatibility mode (with the "-Cp" compiler option) on machines that support the MPE environment. With the use of the USE_MPE_REDIRECTION configuration variable, input for an ACCEPT

statement is read from the file specified by STDIN=, and output from a DISPLAY statement is written to the file specified by STDLIST= on the RUN command line. To enable this behavior, set USE_MPE_REDIRECTION to “1” (on, true, yes). The default value is “0” (off, false, no). In addition, when this variable is set, no terminal manager escape sequences are written to the redirected output file.

USE_MQSERIES

Use this variable to indicate to the runtime that the program makes calls to WebSphere MQ (formerly MQSeries). When USE_MQSERIES is set to “1” (on, true, yes), the runtime attempts to pass calls to functions that begin with the string “MQ” to the WebSphere MQ interface. If the named routine does not exit, the runtime uses the normal search sequence to find a matching function. When USE_MQSERIES is set to the default value of “0” (off, false, no), the runtime does not perform any special handling.

USE_SYSTEM_QSORT

This variable instructs the runtime SORT routine to use the system qsort() function, rather than the built-in sort function. Set USE_SYSTEM_QSORT to “1” if you want to use the system qsort() function. The default value is “0” and results in the use of the built-in sort function.

Some systems have qsort() functions that perform better than the built-in function. Consider experimenting with this variable’s settings to determine if this option yields better performance on your system. Pay particular attention to the number of comparisons done during the sort, which can be seen in the runtime trace output.

USE_WINSYSFILES

This variable specifies whether the runtime should recognize calls to modules with the extensions “.drv” and “.ocx” as well as those with the extension “dll”. By default, it is set to “1” (on, true, yes).

For backwards compatibility, you can turn this feature off by setting it to “0” (off, false, no). Then, only calls to “.dll” files are supported.

V_BASENAME_TRANSLATION

This variable allows you to tell Vision whether to include full path information in the filename. By default, only the base name is included (the filename with no extension and no path information). Retaining the path information can be helpful in instances where Vision files of the same name are stored in different locations and you want to map one of the segments from one directory to a new location.

When V_BASENAME_TRANSLATION is set to “0” (off, false, no), Vision uses the entire path of the file. When it is set to “1” (on, true, yes), the default setting, Vision uses only the base name.

The setting of V_BASENAME_TRANSLATION affects the behavior of three configuration variables that handle Vision filename translation: **filename**, **filename_DATA_FMT**, and **filename_INDEX_FMT**. The following illustrates how the configuration variables interact.

For the file “/user/data/record1.vix”:

- If V_BASENAME_TRANSLATION is set to “on” (the default), *filename*, *filename*_INDEX_FMT, and *filename*_DATA_FMT use “RECORD1_VIX” as the base name.
- If V_BASENAME_TRANSLATION is set to “off”, *filename*, *filename*_INDEX_FMT, and *filename*_DATA_FMT use “_USER_DATA_RECORD1_VIX” as the base name (underscores replace instances of “/” and “.”).

For a description of *filename*, *filename*_INDEX_FMT, and *filename*_DATA_FMT, see their respective entries in this appendix.

V_BUFFERS

This variable sets the number of indexed block buffers to allocate. These buffers are used to improve the performance of indexed files. Each buffer is 512 bytes plus some overhead. Increasing the number of buffers can improve file performance. Decreasing the number conserves memory. The value of V_BUFFERS has no effect on versions of ACUCOBOL-GT that do not use Vision files. The value of V_BUFFERS can range from zero (no buffering) to 2097152. The default value is 64.

V_BUFFER_DATA

The setting of this variable determines whether or not Vision indexed file data blocks (as opposed to key blocks) will be held in the memory-resident disk buffers. When it is set to "1" (on, true, yes), both data blocks and key blocks will use the buffers. When set to "0" (off, false, no), only key blocks will use the buffers. Setting this value to "1" will usually improve performance unless very few buffers are being used.

Note: Holding data blocks in the buffers slightly increases the chances of losing data if a file opened for MASS_UPDATE is not closed properly (power failure, etc.). The default setting of this variable is "1".

V_BULK_MEMORY

Vision allocates a memory buffer for each file opened for bulk addition. The size of this buffer is controlled by the V_BULK_MEMORY configuration option. The default size of this buffer is 1 MB.

Note: The default size is fairly large because it is assumed that only a few files will be open for bulk addition on a system at any one time. If this buffer cannot be allocated, the OPEN fails with a status indicating inadequate memory.

To change the size of the allocated memory buffer to, for example, 500 KB, you would enter:

```
V_BULK_MEMORY = 500 KB
```

V_FORCE_OPEN

This variable allows you to force the runtime to open broken files that would normally cause an error 98. This means you can write COBOL programs to recover these files in ways that are not available with **vutil**. Set V_FORCE_OPEN to "1" (on, true, yes) to open the files. The default is "0" (off, false, no).

Note: When this variable is set to "1", make sure you do not also have the **V_OPEN_STRICT** variable set to "1" because the settings conflict.

V_INDEX_BLOCK_PERCENT

This configuration variable allows you to specify index pre-allocate and extension factors as a percentage of the factors applied to the data segment. In Vision 4 and 5 files, the index data contained in the index segments is often much smaller than the record data contained in the data segments. As a result, a large pre-allocate or extension factor typically allocates many more index blocks than are needed. This can be undesirable, especially if disk space is tight.

Setting V_INDEX_BLOCK_PERCENT to a number less than 100 causes fewer index blocks than data blocks to be created. Setting the variable to a number greater than 100 causes more index blocks than data blocks to be created. The valid range for V_INDEX_BLOCK_PERCENT is one through 1000. If the value specified is less than one, it will be promoted to one. V_INDEX_BLOCK_PERCENT is set to 100 by default (the default pre-allocate and extension factors for a file).

For example, if a file has an extension factor of 10, setting V_INDEX_BLOCK_PERCENT to 50 causes 10 new data blocks and five new index blocks to be created the next time the file is extended. Setting V_INDEX_BLOCK_PERCENT to 200 causes 10 new data blocks and 20 new index blocks to be created the next time the file is extended.

Note: The number of blocks pre-allocated will never be larger than that which can fit in the initial data and index segments. If the pre-allocation value specified or calculated from V_INDEX_BLOCK_PERCENT is larger than the segment size, the pre-allocation amount is automatically reduced to the segment size.

V_INTERNAL_LOCKS

This configuration variable allows you to control whether the runtime enforces internal record or file locking. When V_INTERNAL_LOCKS is set to "0" (off, false, no), Vision tracks locks but does not enforce internal record or file locking. As a result, the runtime does not return a record or file locked condition for a record or file that was previously locked by the same run unit. When V_INTERNAL_LOCKS is set to the default of "1" (on, true, yes), internal record and file locking are enforced.

Note: The Windows operating system enforces a single lock per process on a region of a file. This means that if your program opens the same physical file as two different logical files and then tries to lock the same record in both "files", the second lock will fail (with an error "99") even if V_INTERNAL_LOCKS is set to "0". So V_INTERNAL_LOCKS 0 practically affects programs running on UNIX operating systems only.

V_LOCK_METHOD

This variable selects which locking method Vision will use to control simultaneous access to indexed files. It affects only the Vision file system, and only files directly accessed by the runtime (it does not apply to files accessed via AcuServer).

The default setting of "0" (zero) causes Vision to lock the first byte of the file for every access to the file (both reads and updates). This ensures that the process is not interfered with by another process. This locking method is always used by Vision Version 2 files.

Setting this variable to “1” causes Vision to lock the first byte of the file for all operations except random READs or READ NEXTs. These two operations proceed without the lock. Instead they perform some additional reads of the file, to ensure that they get consistent results. If they get inconsistent results, they are retried, this time locking the first byte as other operations do. This locking method is available only for Vision Version 3, 4, and 5 files.

Note: This variable must have the same setting for all the runtimes accessing a file, whether they are reading or writing to it. For example, if a runtime set with V_LOCK_METHOD=1 is reading from a file, any runtimes that are writing to that same file must also have V_LOCK_METHOD set to 1.

Lock method “1” can produce better performance on some machines. These machines fall into two categories:

- Machines that take a long time to place a lock.
- Machines that do not queue lock requests, and are very busy. In this case, some users typically get good performance, while others get poor performance.

Setting V_LOCK_METHOD to “1” might help improve performance with Vision Version 3, 4, or 5 files. For example, setting V_LOCK_METHOD to “1” can be helpful on some Windows networks. A peer-to-peer network of Windows 98 machines can exhibit problems reading Vision files when a process performs a tight read loop. The problem usually surfaces as either an error 30,33 or an unexpected error 99. This occurs because the runtime is unable to place a lock on the header of the file after 400 attempts over a 20-second period. For other networks, setting V_LOCK_METHOD to “1” can substantially reduce the number of lock requests made by the runtime and can often resolve these problems.

To get statistics about header locks, select Trace Files level “3” in the debugger (for example, “TF 3”). These statistics print on the runtime’s error output each time a Vision file is closed. They cover the operations in that file since it was last opened. You can also view these statistics (without the full trace) by adding “256” to the lock method chosen (for example, setting V_LOCK_METHOD to “257” selects method “1” and prints statistics).

Setting the V_LOCK_METHOD variable to “2” enables “asynchronous reads” of Vision files. This option is intended to further reduce the number of file locks required to perform random READs and READ NEXTs.

The advantage of the “2” setting is that it is less likely to require retrying a READ with a lock when a file is undergoing heavy modification. With V_LOCK_METHOD=1, the READ is retried with a lock whenever it detects that the file has been updated in any way; with V_LOCK_METHOD=2, the READ is retried only when Vision encounters inconsistent data while traversing the index tree or reading the record data. This leads to less locks and therefore greater performance for machines with slow locking functions.

V_LOCK_METHOD=2 works only for Vision 4 and 5 files. A fundamental requirement for the V_LOCK_METHOD=2 feature to work properly is that the operating system must provide atomic write operations. That is, if one process is writing to a file, another process will always see the contents of the file as it exists either before or after the write operation, never the intermediate contents as the write operation runs. There is evidence that Linux does not provide atomic writer operations and therefore it is not recommended to use this setting in a Linux environment.

If any process reading a particular file is using V_LOCK_METHOD=2, all other processes (runtimes) updating that file must be ACUCOBOL Version 5.0.0 or greater. This is because Version 5.0.0 contains changes that affect the way Vision updates the tree structure of its files. These changes allow for greater consistency of the tree from the viewpoint of an asynchronous reader. This requirement is not enforced by Vision, however, so it is important for the users to pay careful attention to the versions of programs accessing their files to avoid receiving erroneous data. Therefore, before enabling this option, make sure that all runtimes updating files on which asynchronous reads are to be performed (V_LOCK_METHOD=2) are Version 5.0.0 or later.

As with V_LOCK_METHOD=1, adding 256 to the value of the V_LOCK_METHOD setting causes statistics about header locks to be printed to the runtime’s error output each time a Vision file is closed. So, setting V_LOCK_METHOD=258 selects method 2 and turns on the header lock statistics.

V_MARK_READ_CORRUPT

This variable allows you to configure Vision so that it does not mark a file as broken if it encounters a corruption during a read or start operation. The effect is that the user is allowed to retry the program. This may be useful when the error is spurious (for example due to a network caching glitch). If the user retries the program and once again receives a file-corrupt message, then the file should be rebuilt or recovered normally. To enable this option, set the configuration option "V_MARK_READ_CORRUPT" to "0" (off, false, no). The default setting is "1" (on, true, yes).

V_NO_ASYNC_CACHE_DATA

This configuration variable turns on the caching of data blocks for file reads. By default, Vision 4 and 5 do not cache data blocks in its internal cache (all V_BUFFERS are allocated only to index blocks). This is required for the asynchronous reads feature (**V_LOCK_METHOD**=2) to work properly (each data record needs to be read/written in a single system call).

The default setting of this configuration variable is "0" (off).

If you are *not* using the asynchronous reads feature *at all*, you may turn on the caching of data blocks by setting the V_NO_ASYNC_CACHE_DATA configuration variable to "1". This may improve READ performance.

Caution: Be certain that you do not use this configuration variable with V_LOCK_METHOD=2 in any combination, as silent data corruption may result.

V_OPEN_STRICT

By default, Vision allows OPEN INPUT on files that are marked as broken. This behavior is intended to make it easier to recover records from broken files. If you want to receive an error status when opening a file marked as broken for INPUT, set V_OPEN_STRICT to "1" (on, true, yes). The default setting of "0" (off, false, no) allows open input on broken files.

Note: When this variable is set to "1", make sure you do not also have the **V_FORCE_OPEN** variable set to "1" because the settings conflict.

V_READ_AHEAD

Setting this configuration variable to "0" (off, false, no) turns off Vision's read-ahead logic. This may improve performance in cases where highly random file processing is being used. The default value is "1" (on, true, yes).

V_SEG_SIZE

This configuration variable sets the maximum size of a Vision 4 or 5 file segment in bytes. The default value is 2,147,482,112 (i.e., 2GB – 1536), except on older HP/UX machines where it is 1,073,740,288 (i.e., 1GB – 1536) due to an operating system limitation. You may not use larger values, but you can set smaller ones. The default value is the maximum allowed. The value specified will automatically be rounded down to a multiple of the block size of the file being created. For example, if the default V_SEG_SIZE value is used and a file with a block size of 1024 is created, the segment size for that file will be 2,147,481,600 (i.e., 2GB – 2048).

Using a smaller value for the segment size can help if you do not have 2GB free on any disk or for testing purposes. The minimum value allowed is 81,920 bytes. To minimize the number of files created, you should set this value as high as possible.

The segment size of a file is set at file creation time and cannot be modified without recreating the file (i.e., using **vutil** –**rebuild** with a different V_SEG_SIZE setting). **vutil** uses this variable, but since it does not use a configuration file, this variable must be set in the environment.

V_STRIP_DOT_EXTENSION

The V_STRIP_DOT_EXTENSION variable determines whether or not Vision strips a trailing "dot extension" (".dat") from the logical name of a data file when generating file names for index and data segments (other than the first data segment). Setting this variable to "0" prevents the extension from being removed. For example, by default, the first index segment name for the logical file "file.one" is "file.vix" (which would conflict with the index segment of "file.two"). When V_STRIP_DOT_EXTENSION is set to "0" (off, false, no), the index segment name is "file.one.vix". The default value for this variable is "1" (on, true, yes).

Note: The setting of this variable affects the behavior of four configuration variables: **filename**, **filename_DATA_FMT**, **filename_INDEX_FMT**, and **filename_VERSION**. See their respective entries in this appendix for details.

V_VERSION

This variable specifies the version number of new Vision files that are created. The default value is "5", which produces Vision files in the format of the current version (Version 5). The value "4" produces Version 4 files. Version 5 and 4 files are generated in a dual file format, with data records filed in one segment and overhead key information filed in another. The value "3" produces Version 3 files, in which data and keys are stored in a single file. The value "2" produces Version 2 files. Any value other than "2", "3", or "4" produces Version 5 files.

V23_GRAPHICS_CHARACTERS

Programs written for and executed with UNIX versions of the runtime up to and including Version 2.4.0 use hex values 1-8 to display line drawing characters on the screen. Runtimes after Version 2.4.0 use hex values offset by one (1). When older programs are used with runtimes released after Version 2.4.0, line drawing characters do not display as expected. To use the old values for line drawing characters, set this variable to "1" (on, true, yes).

If the variable is set to "0" (off, false, no) or is not set at all, the runtime will use the newer offset values. This variable works only for UNIX systems.

V30_MEASUREMENTS

This configuration variable affects whether the runtime sizes certain controls according to the rules from Version 3.0 or from the current version. If the current measurement code is causing your application to display incorrectly, then setting this variable to "1" (on, true, yes) will use Version 3.0 sizing rules instead. When V30_MEASUREMENTS is set to the default "0" (off, false, no), then the current sizing rules are in effect.

The related configuration variables, V31_MEASUREMENTS and V32_MEASUREMENTS have the same effect of setting the sizing rules to that of their respective versions.

V31_FLOATING_POINT

This configuration variable allows you to disable a correction that was made to the way floating-point numbers are displayed. Because some loss of precision in the display of "USAGE DOUBLE" fields was possible in Version 3.1, an improvement was introduced. Setting this variable to "1" (on, true, yes) means that the Version 3.1 method of displaying floating-point numbers is used. When V31_FLOATING_POINT is set to the default "0" (off, false, no), then the correction is in effect.

V42_FLOATING_POINT

This variable affects how floating-point arithmetic is performed. Starting with Version 4.3, floating-point arithmetic was enhanced to more closely reflect the way that floating-point values are determined on the host system. This enhancement can affect the behavior of existing programs. To revert to the computation method used prior to Version 4.3, set the value of V42_FLOATING_POINT to "1" (on, true, yes). By default, this variable is set to "0" (off, false, no).

V43_PRINTER_CELLS

This variable affects whether the runtime sets the width of a printer cell according to the rules from Version 4.3 or from the current version. Version 4.3 (and prior versions) computed the width of a printer cell based on the average width of a selected printer font. The width of a printer cell is currently computed in the same way that cells are computed for the screen, namely by the width of the "0" character. For fixed-width fonts, such as Courier, these values are the same for all characters. For proportional fonts, such as Times New Roman, some characters might be wider than the "0" character.

If the current computation is causing your application to print incorrectly, then setting this variable to "1" (on, true, yes) will use Version 4.3 rules instead. When V43_PRINTER_CELLS is set to the default "0" (off, false, no), then the current rules are in effect.

V52_BITMAP_BUTTONS

If some event in the system forces the focus away from a bitmap-based push button after a click has been started but not finished, this variable determines whether the click is voided. If you do not want the click to be voided, set this variable to "1" (on, true, yes). The default setting is "0" (off, false, no).

V52_BITMAPS

This variable determines whether your application uses device-dependent or device-independent bitmaps for image processing. The following settings are recognized:

1	Use Version 5.2 and earlier image-processing code (device-independent) for bitmap controls.
0	Use Version 6.0 and later image-processing code (device-dependent) for bitmap controls.
-1	(default) Dynamically apply the image-processing code based on the program's object semantics. For programs compiled for pre-Version 6.0 semantics, use the older imaging code. For programs compiled for Version 6.0 or later semantics, use the newer code.

V52_GRID_GOTO

This configuration variable determines how the runtime behaves when a user clicks in a grid control cell containing the cursor. Prior to Version 5.2, the runtime would not pass a MSG-GOTO-CELL-MOUSE event to the program when the user clicked in a grid cell containing the cursor. For programs compiled with Version 5.2, or later, this event *is* passed to the program. Setting V52_GRID_GOTO to "0" (off, false, no), maintains the pre-5.2 behavior. The default of "1" (on, true, yes) enables the new behavior, even for programs compiled with Versions 5.1 or earlier and run with Versions 5.2 or later. See **Appendix C, "Appendix C: Changes Affecting Previous Versions,"** for more details.

V60_LIST_VALUE

This variable allows you to select the algorithm used by the runtime to match a list box or combo box VALUE with an item in the control's list.

Prior to Version 6.0, setting the VALUE of a combo box or list box caused the first item in the list that started with the value of VALUE to be selected, regardless of case. Beginning with Version 6.0, when a box's VALUE is set, the list is searched for an exact, case sensitive match with the specified value. If the value is found, it is selected. If an exact match is not found, the list is searched for an exact match regardless of case. If a match is still not found, the list is searched again, this time for the first string that contains the passed VALUE as a leading substring, regardless of case. V60_LIST_VALUE allows you to specify which algorithm to use. It accepts the following values:

1 directs the runtime to use the Version 6.0 search algorithm

0 directs the runtime to use the pre-6.0 search algorithm (substring search only)

-1 (default) directs the runtime to use the 6.0 search algorithm on objects compiled for Version 6.0 or later, and to otherwise use the old search algorithm. This means that objects compiled for compatibility with versions prior to 6.0 that are run with a Version 6.0 runtime will not exhibit the new behavior.

V62_MAX_WINDOW

Starting with Version 7.0, when the runtime reduces the size of a window to fit the screen, it includes any fractional lines and columns that fit, provided the COBOL program attempts to create a window with fractional lines and columns. For example, if you create a 70.0 line window, but only a 66.4 line window fits on the display, the runtime detects that no fractional lines were attempted, and truncates the number of lines to 66.0. However, if you attempt to create a 70.1 line window, the runtime recognizes the fractional measurement and displays a 66.4 line window. To preserve the pre-7.0 behavior, set the configuration variable V62_MAX_WINDOW to "1" (on, true, yes) and fractional lines and columns are always removed. The default value is "0" (off, false, no).

V71_ALIGNED_ENTRY_FIELD

Starting with Version 7.2, the wheel mouse can be used for scrolling in a center- or right-aligned entry field. To preserve the pre-7.2 behavior, set the V71_ALIGNED_ENTRY_FIELD configuration variable to “1” (on, true, yes). The default value of this variable is “0” (off, false, no).

V71_FONT_WIDTHS

Windows has a function called GetTextMetrics that returns information about a font. This data is used by the runtime to compute the “maximum character width” and “wide character width” of a font. The “maximum width” amount is used to set a lower bound for how small an entry field can be (to ensure that at least one character is always visible). The “wide width” is used to scale small entry fields and uppercase entry fields. The “wide width” is computed by averaging the maximum and average character widths. Experimentation has shown that the “maximum character width” data returned may be inaccurate, sometimes by very large margins.

With the use of the V71_FONT_WIDTHS configuration variable, the runtime validates the data returned by the Windows function and corrects it when it is too large. The change does not affect programs until they are recompiled with Version 7.2 or later, or the change is specifically enabled through the V71_FONT_WIDTHS configuration option. The variable can have the following values:

-1	(default) The change is enabled for programs using Version 7.2 or later semantics. In other words, the program has been compiled with Version 7.2 or later and the command line does not contain a compiler option for pre-7.2 semantics.
0	The change is enabled.
1	The change is disabled and the Version 7.1 and earlier font measuring code is used.

Please note the following issues regarding the use of this variable:

- The runtime's standard fonts are not affected by this configuration variable setting.

- Entry fields defined by physical units (CELLS or PIXELS) and all screens created using the AcuBench Screen Designer will not change.
- Entry fields will not grow larger due to this configuration variable setting. The majority will stay the same size, and a few might get smaller.
- Fixed width fonts are not affected by this configuration variable setting.

VMS_COBOL

When set to "1" (on, true, yes), this causes the VMS runtime to write to print files the same way that VMS COBOL does. When set to "0" (off, false, no), this variable causes the runtime to behave as it always has. The default setting is "0" (off, false, no).

WAIT_FOR_ALL_PIPES

This configuration variable determines if the runtime calls the wait system call each time a “-P” file is closed. When WAIT_FOR_ALL_PIPES is set to “0” (off, false, no), the runtime does not make this call until it is ready to close the last pipe it knows about. Setting this configuration variable to the default “1” (on, true, yes) means that the runtime calls the wait system call when a “-P” file is closed.

WAIT_FOR_FILE_ACCESS

This configuration variable is designed for Windows 98 systems. It gives you some control over situations where a user must wait for access to a shared file. The runtime will try repeatedly to acquire the file lock, up to 400 times. If it has been unable to obtain a file lock after 400 tries, it will (by default) display a message box, asking if the user would like to continue waiting. If the user clicks the “Yes” button, then the runtime will try again

another 400 times (or the value of LOCKING_RETRIES). If the user clicks the "No" button, then the runtime will return an error to the COBOL program (such as file error 30,33 (system error) or file error 99 (record locked).

The WAIT_FOR_FILE_ACCESS variable lets you choose one of three behaviors: either the user will always see the message box and make a choice, or the program will always return an error code if it cannot acquire the lock, or the runtime will always behave as if the user answered "yes" to the message box.

You can modify the text shown to the user in the message box via the TEXT configuration variable. The message is number 28. To include the filename in your message, insert "%s" at the place where you want the name of the file to appear. You can introduce line breaks by including "\n" in the message.

Possible values for the WAIT_FOR_FILE_ACCESS variable are:

0	"No"	Do not display message box if lock is not acquired. Send error to COBOL program.
1	"Ask"	Show the message box and ask the user. (Default)
2	"Yes"	Do not show the message box. Assume that the user wants to wait for the file. This ensures that the user eventually can access the file, but introduces a small risk of an infinite loop if the system's lock table becomes corrupt.

For programs running in background ("-b" runtime option), or programs with redirected input or output, the "Ask" option is treated the same as the "Yes" option.

WAIT_FOR_LOCKS

This determines how the runtime handles file status error 99 conditions on record reads. This variable is not checked on record write operations. It can have one of the following values:

0	Do not wait for locked records, return error 99.

1 Wait for the locked record if no Declarative is available for the file, otherwise return error 99.

2 Always wait for the locked record, never return error 99.

Any other value (including the default value of "-1") causes the runtime to wait for locked records only if you have compiled for RM/COBOL compatibility and the file does not have a Declarative.

WARNINGS

This configuration variable controls whether a warning message is printed and an error raised for the following conditions:

1. when non-numeric data is used in a context where numeric data is required
2. when there is a reference modification range error

By default, the runtime silently corrects reference modification range errors as follows:

- A start reference less than 1 is treated as 1. For example, var(0:3) is treated as var(1:3).
- A length reference less than 0 is treated as 0. Moving a zero-byte item is equivalent to moving spaces to the destination item. A zero-byte destination is not affected by the move. In a STRING statement, a length of zero for a string source is treated as 1, not 0.
- A start plus length reference that is past the end of the item is treated as meaning to the end of the item. For example, if the var is a PIC X(5) item, var(4:23) is treated as var(4:2).

WARNINGS can take the following values:

0 (off, false, no) No warning is printed.

1 (on, true, yes) A warning is printed. This is the default.

2 A warning is printed or sent to the error file. If you are in the debugger, an automatic breakpoint occurs.

3 For a non-numeric error, a warning is printed, an *intermediate* error is generated that calls the installed error procedures, if any, and the runtime is halted. For more information on error procedures, see **CBL_ERROR_PROC** in Appendix I.

Note: The setting you select for WARNINGS applies to reference modifier range errors when the start plus length reference is past the end of the item. Reference modifiers that are equal to or less than zero are always silently corrected as described above.

WARNING_ON_RECURSIVE_ACCEPTS

An event procedure may CALL another procedure which may contain ACCEPT statements, which, in turn, may contain embedded procedures. Although this is handled in the same fashion as nested PERFORMs and is perfectly legal, doing this poses the danger of going from one ACCEPT to another uncontrollably. When the limit of 10 nested accepts is reached, the program starts overwriting memory. It is possible to warn the user when the limit is reached by giving this configuration variable a zero (“0”) value. This gives users the opportunity to continue at their own risk. Giving WARNING_ON_RECURSIVE_ACCEPTS a non-zero value suppresses the warning.

To avoid overwriting memory, you may choose to re-code affected programs to terminate the ACCEPT and perform the CALL after you exit from the ACCEPT. You may also use CHAIN or CALL PROGRAM instead of the regular CALL, if applicable.

WHITE_FILL

This variable has meaning only on graphical systems such as Windows. Some graphical systems (such as Windows) use a “background brush” when they resize a window. By default, the background brush color for ACUCOBOL-GT is black (“0”, off, false, no). If you have arranged your

default background to be white, you will see a black flash when you resize the window. This does not affect the final appearance of the window, but is briefly noticeable while the window is being redrawn.

Set WHITE_FILL to "1" (on, true, yes) to cause ACUCOBOL-GT's background brush to be set to white instead of black. Doing this will also cause the initial screen that ACUCOBOL-GT paints to be white instead of black.

Note: This variable *must* be set in the configuration file to be effective. Modifying this variable with the SET ENVIRONMENT verb has no effect.

WIN_ERROR_HANDLING

This variable has meaning only on graphical systems such as Windows. Use WIN_ERROR_HANDLING to control how hardware errors are handled.

When this variable is set to the default of "1" (on, true, yes), certain errors are handled directly by the host environment, and do not automatically return a file error code. For these errors, a dialog box is displayed that describes the error and offers "Cancel" and "Retry" buttons. The user may correct the error and press "Retry". If the user presses "Cancel", then your program receives the file error that it would have normally received.

If you set WIN_ERROR_HANDLING to "0" (off, false, no), then the dialog box is not shown, and your program receives the error directly.

WIN_F4_DROPS_COMBOBOX

This configuration variable applies only to programs running under Windows.

If WIN_F4_DROPS_COMBOBOX is set to its default value of "1" (on, true, yes), then combo boxes use the standard Windows handling for the <F4> key. Pressing <F4> while a combo box is active causes it to drop its drop-down list, and the COBOL program is not notified of an exception.

When this variable is set to “0” (off, false, no), pressing <F4> with a combo box active causes the COBOL program to get the exception, but the combo box does not drop its drop-down list.

It is not possible to get both behaviors at the same time.

WIN_SPOOLER_PORT

This variable allows you to divert printer output to a file or port through the Windows print spooler. Files created in this way are stored in binary encoding. You may set the Windows print spooler with “-P SPOOLER” or “-Q <*printername*>” with or without the DIRECT option. However, if you omit the DIRECT option, the resulting file will include all the embedded control codes formatting the print job for the original target printer.

By default, the value of WIN_SPOOLER_PORT is undefined. Set WIN_SPOOLER_PORT to a valid filename or port. This can be done in a configuration file, in the environment, or in the program. For example:

```
WIN_SPOOLER_PORT  c:\mydir\myprint.prn
```

or

```
SET ENVIRONMENT "WIN_SPOOLER_PORT" TO "c:\mydir\myprint.prn".
```

This will affect all print jobs performed in the current instance of the runtime. Any graphics operations performed in the COBOL application, such as WINPRINT-BITMAP or WINPRINT-GRAPH-DRAW, are preserved in the file, and will print. However, these options may result in a very large binary file.

The resulting file can be copied directly to any printer that is compatible with the original target printer. For example, the following command:

```
COPY /B c:\mydir\myprint.prn LPT1
```

will send the file to LPT1, while the “/B” option tells the COPY command that the file contains binary encoding.

WIN3_CLIP_CONTROLS

This option is specific to the Windows versions of ACUCOBOL-GT. It affects the way in which updates to a window interact with graphical controls in a window. Normally, Windows allows updates to a parent window to show through any controls in that window. The controls are then updated to create the proper final appearance. This is very fast, but it can cause controls to flash when the background is being updated. When this option is set to "1" (on, true, yes), the controls are clipped from the update region in the parent window before the parent is repainted. This causes the controls to remain relatively stable; however, screen repaints can be significantly slower, particularly when the runtime is creating and destroying controls. The default setting for this option is "0" (off, false, no). We recommend that you experiment with both settings to see which you prefer. Note that this option is examined when a floating window is created. Once a window is created, changes to this option have no effect on that window.

Note: When turned on, this option causes the Windows WS_CLIPCHILDREN style to be used whenever floating windows are created.

WIN3_EF_PADDED

This configuration variable has meaning only on Windows systems. Under Windows, unboxed entry fields include a small amount of extra space so the cursor can be seen when it is placed after the last character position. This space can be a problem if you want to convert a program and align screen items. When WIN3_EF_PADDED is set to "0" (off, false, no), this extra space does not appear in unboxed entry fields, and the entry field has only enough space for its character positions. When this variable is set to the default "1" (on, true, yes), the extra space appears in unboxed entry fields.

WIN3_GRID

This option is specific to Windows. When set to a non-zero value, it causes a fine grid to be drawn in each floating window. The grid outlines the character cells in the windows. This is intended as a debugging tool, to help you see how various controls line up against the window's character cells. It can also help you adjust the layout of a screen.

The grid is drawn using the color number that WIN3_GRID is set to (see the COLOR phrase for the exact values). For example, setting WIN3_GRID to "4" will draw a cyan grid. The grid is drawn with dashed lines. Every fifth horizontal line and every tenth vertical line is drawn with a solid line.

WIN32_3D

This configuration variable causes the runtime to use the native 3-D features of Windows when drawing controls with the 3-D style. This has an effect only with the 32-bit Windows runtime. Turn this feature on by setting WIN32_3D to "1" (on, true, yes). When set to the default of "0" (off, false, no), the runtime supplies its own 3-D effects. The advantage of using the native Windows 3-D is that you get a slightly more modern appearance and a closer match to the appearance of other Windows programs. The disadvantages are:

1. Windows always draws the border using the colors selected in the system's control panel. As a result, the effect looks right only when placed on a window whose background is the USER-GRAY color. You can accomplish this easily by creating STANDARD windows that specify BACKGROUND-LOW.

2. The Windows 3-D effect is slightly larger than the runtime's 3-D effect. Windows draws a 1-pixel wide border around the control that is the same color as the USER-GRAY color. This border is essentially invisible against a window with the USER-GRAY background. However, this border can overwrite anything else that may be positioned there. The net effect is that you can't place controls as close together as you can with the runtime's 3-D.

3. This 3-D style can be used only with the 32-bit runtime.

The runtime adjusts for the physical differences between the two styles. Under either style, the position and usable size of the control's interior should be same.

Note: This configuration setting can affect the behavior of an application if it is using the latest Windows control styling, that is the **WIN32_NATIVECTLS** configuration variable set to 1, true, or on. If WIN32_3D has not been set by the user then the default value will be overridden and set to false (0 or off). If the user has set WIN32_3D then their settings will not be overridden and if it is set to true (1 or on) then 3D drawing will occur over the top.

WIN32_CTL_INPUT_STATUS

Setting this variable to the default of "1" (on, true, yes) causes ACCEPT…FROM INPUT STATUS to return a non-zero status if data is available in a control. If set to "0" (off, false, no), then the data in the control does not affect the status returned by ACCEPT…FROM INPUT STATUS.

This variable is only available in the Windows runtime and is not available to the thin client.

WIN32_NATIVECTLS

This variable enables your application to use the Windows control style that is in use on the workstation, (the workstation's theme is set to Windows XP or Vista). To enable these visual styles, your application must be running on an operating system that contains ComCtl32.dll version 6, which is included with Windows XP and Vista.

When set to "1" (on, true, yes), the application will display the current control styling available on that operating system, the XP look and feel on the Windows XP OS or the Vista look and feel on the Windows Vista OS.

Note: In addition to visual differences, some XP and Vista controls have different behaviors than their Windows classic counterpart (by Microsoft design). The behavior differences if any, that our internal testing has identified are documented in Book 2, Chapter 5 under the applicable control. Alternatively, you can find a consolidated list in the 8.1 ECN List (ECN 3734) located at the support section of the Micro Focus website.

The default setting is "0" (off, false, no) which prevents the runtime from using the Windows control styling, and forces the “gray chiseled” or classic Windows look.

Note: The configuration variable **WIN32_3D** can also change the look of controls in an ACUCOBOL-GT application. It is generally recommended that you leave WIN32_3D set to its default behavior of off or false when setting WIN32_NATIVECTLS to on or true.

Note: The Windows OS allows users to configure (accessibility options) whether or not keyboard shortcut names appear with underlines. For example “ctrl+c” vs. “ctrl+c”. The WIN32_NATIVECTLS respects this setting and will display shortcut names accordingly.

WINDOW_INTENSITY

This configuration variable controls whether the color settings specified in the COLOR phrase of the DISPLAY WINDOW statement are used or ignored by the runtime. When the value of this variable is set to “0” (off, false, no), the COLOR intensity settings in all DISPLAY WINDOW statements are ignored. When the value of this variable is set to “1” (on, true, yes), which is the default, the runtime sets the windows intensity as specified.

WINDOW_TITLE

This variable has meaning only on graphical systems such as Windows. The ACUCOBOL-GT runtime system automatically sets the title of the application window to the base name of the initial object file. For example, if you run a program called “notepad.cbx”, then the title on the main window will be set to “Notepad”. The title is shown in lower-case except for the first letter, which is made upper case.

You may provide an alternate title by setting WINDOW_TITLE to the desired text. No translation of the text is done, so you should enter it using the desired case.

Note: Setting WINDOW_TITLE from within a program has no effect, because the WINDOW_TITLE setting determines only the window’s initial title.

To change the title from within your program, use a DISPLAY statement. The syntax is:

```
DISPLAY text UPON GLOBAL WINDOW TITLE
```

where *text* is an alphanumeric literal or variable. Enter the title with the desired case. The title is always shown in the ANSI font, so if you are using a different font, your text will be translated to ANSI.

To ensure that the WINDOW_TITLE variable operates as expected, make sure that the first screen operation in your program is not DISPLAY WINDOW with a title (the DISPLAY WINDOW title is stored in the same place as the WINDOW_TITLE). Instead, do some other screen operation first, such as “DISPLAY WINDOW, ERASE”.

WINPRINT_NAMES_ONLY

This variable allows you to generate a list of the names of printers installed on a Windows PC. It does this by altering the behavior of some of the operations of the **WIN$PRINTER** library routine. When WINPRINT_NAMES_ONLY is set to a value of “1” (on, true, yes), the

WIN$PRINTER operations that retrieve printer information return only the names of installed printers, rather than the real-time status of all available printer capabilities.

This variable can be set in the configuration file or directly in your program with the following code:

```
SET ENVIRONMENT "WINPRINT-NAMES-ONLY" TO "1".
```

Note on WIN$PRINTER library routine:

When this variable is turned on, the following operations of the WIN$PRINTER library routine are affected:

```
WINPRINT-GET-PRINTER-INFO
WINPRINT-GET-PRINTER-INFO-EX
WINPRINT-GET-CURRENT-INFO
WINPRINT-GET-CURRENT-INFO-EX
```

Instead of returning detailed information about the capabilities of each printer (duplex, copying, etc.), the routine returns only the name of the printer. This can provide a significant performance improvement, particularly with networked printers.

If you are using the default printer settings, set the WINPRINT_NAMES_ONLY variable to “1”, generate a list of printer names using WINPRINTER-GET-PRINTER-INFO-EX (see the **WIN$PRINTER** documentation in the Appendices of the ACUCOBOL-GT manual set, or refer to the sample program “prndemox.cbl”), and select the desired printer.

If you want to modify the printer settings, such as the number of copies or the paper orientation, you should perform the steps described above, and then set WINPRINT_NAMES_ONLY back to the default of “0” (off, false, no). You may then use WINPRINT-GET-PRINTER-INFO-EX to obtain detailed information about the capabilities of the selected printer.

For more information about Windows printing, refer to WIN$PRINTER in Appendix I.

WRAP

The setting of this variable determines whether a DISPLAY statement will wrap around or be truncated when it extends past one line. When it is set to "0" (off, false, no), DISPLAY statements will be truncated. Also, any DISPLAY statement that references a column past the right edge of the current window will be ignored. An ACCEPT statement that references a column past the right edge will be placed in the home position of the window. The default value for this setting is "1" (on, true, yes).

XFD_DIRECTORY

This variable tells the runtime system the name of the directory that contains the data dictionaries built by the ACUCOBOL-GT compiler. The default value is the current directory.

For example, to tell the runtime that the dictionaries are stored in the directory "/usr/inventory/dictionaries" you would enter:

```
xfd_directory   /usr/inventory/dictionaries
```

See also the "-Fo" compile-time option, which tells the compiler where to put the dictionaries. Unless you have moved the dictionaries, you should use the same value for XFD_DIRECTORY that you used with the "-Fo" option.

If you have embedded an XFD file in an object library, the runtime will read that file instead of an XFD file that has the same name but is stored in the directory specified by XFD_DIRECTORY. The exception to this is when the XFD_DIRECTORY configuration variable uses remote name notation.

Remote name notation is allowed for the XFD_DIRECTORY variable if your runtime is client-enabled. See *ACUCOBOL-GT User's Guide* sections 5.2.1 and 5.2.2 for more information about client-enabled runtimes and remote name notation.

XFD_PREFIX

This variable defines a series of directories to search for XFD files, rather than indicating only one (as in XFD_DIRECTORY). Each directory is searched in order until an XFD matching the name of the file is found. Once a file with the same name is found, the runtime stops searching, even if other files of the same name are located in a subsequent directory in the search parameter. Only named directories are searched, not subdirectories.

Note: If the XFD you are searching for does not match the file specifications (max-keys, max-rec-size, min-rec-size, and key parameters, for example) of the file you are trying to open, the runtime will not continue searching the directories listed in XFD_PREFIX until a correct XFD file is found.

The default for XFD_PREFIX is empty. If this variable is set to any other value, the configuration variable XFD_DIRECTORY (in which you specify only one directory) is ignored. You can specify a directory path that contains embedded spaces if you surround the path with quotation marks. Separate entries using a semi-colon (;). For example:

```
XFD_PREFIX C:\ "Sales Data";C:\Customers
```

You may specify up to 4096 characters for this variable. Remote name notation is allowed for the XFD_PREFIX variable if your runtime is client-enabled. See *ACUCOBOL-GT User's Guide* sections 5.2.1 and 5.2.2 for more information about client-enabled runtimes and remote name notation.

XTERM_PROGRAM

Some users may want to debug with an xterm, but don't actually want to debug with the xterm executable because it doesn't have some of the abilities they need (such as displaying non-ASCII characters). You can specify the executable used to show the debugger on UNIX by setting the XTERM_PROGRAM runtime configuration variable.

Its default value is “xterm”, but it can be set to any compatible program such as dtterm or kterm. The runtime executes this program when it tries to create the program for background debugging. Note that the runtime passes some arguments to this program, so this program must be able to execute with those arguments. These arguments are:

-title “title of the window”

-Sccn

-display Xserver-name

The “-Sccn” option allows the program to be used as the input and output channel for the runtime, and is absolutely required. Without this option, the program won't know to display data from the runtime.

Library Routines

Key Topics

I.1 General Syntax and Library List

ACUCOBOL-GT has a large set of library routines built into the runtime system. These routines may be accessed via the CALL verb. This appendix describes each of these routines in detail. The routines are listed in alphabetical order.

In the following descriptions, the phrase “Numeric parameter” indicates a data item or literal that contains a numeric value in any of the following formats:

- Signed or unsigned COMP-4 (or internal equivalent such as COMP-X)
- Unsigned PIC 9 USAGE DISPLAY
- PIC X containing digits (other data ignored)
- Unsigned numeric literal
- Alphanumeric literal containing digits (other data ignored)

Any routine that has a GIVING phrase specified in its USAGE may omit that phrase. If this is done, then the routine’s return value will be placed into the special register RETURN-CODE instead.

ASCII2HEX

ASCII2HEX converts binary data to its hexadecimal format. This routine is the inverse of the HEX2ASCII routine.

Usage

```
CALL "ASCII2HEX"
    USING ASCII-VALUE, HEX-VALUE
```

Parameters

ASCII-VALUE PIC X(2)

The input data area containing the ASCII representation of a unit of data.

HEX-VALUE PIC X(4)

The output data area to contain the hexadecimal value.

When you define the parameters, use the exact field sizes specified in the calling conventions above, otherwise the runtime may terminate abnormally.

ASCII2OCTAL

ASCII2OCTAL converts binary data to octal format. This routine is the inverse of the OCTAL2ASCII routine.

Usage

```
CALL "ASCII2OCTAL"
    USING ASCII-VALUE, OCTAL-VALUE
```

Parameters

ASCII-VALUE PIC X(2)

The input data area containing the ASCII representation of a unit of data.

OCTAL-VALUE PIC X(8)

The output data area to contain the octal value.

When you define the parameters, use the exact field sizes specified in the calling conventions above, otherwise the runtime may terminate abnormally.

CBL_AND

CBL_AND performs a binary, bitwise “and” operation on a series of bytes.

Usage

```
CALL "CBL_AND"
    USING SOURCE, DEST, LENGTH
    GIVING STATUS
```

Parameters

SOURCE PIC X(n)

The source bytes for the operation.

DEST PIC X(n)

The destination bytes for the operation.

LENGTH Numeric parameter (optional)

The number of bytes to combine. If omitted, then CBL_AND uses the minimum of the size of SOURCE and the size of DEST.

STATUS Any numeric data item

The return status of the operation. Returns “0” if successful, “1” if not. This routine always succeeds, so STATUS always contains a zero.

Description

For LENGTH bytes, each byte of SOURCE is combined with the corresponding byte of DEST. The result is stored back into DEST. The bytes are combined by performing an “and” operation between each bit of the bytes. The “and” operation uses the following table to determine the result:

And	**0**	**1**
0	0	0
1	0	1

CBL_CLEAR_SCR

The CBL_CLEAR_SCR routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine clears the entire screen using a specified character and attribute.

Usage

```
CALL "CBL_CLEAR_SCR"
    USING CHARACTER, ATTRIBUTE
    RETURNING STATUS-CODE
```

Parameters

CHARACTER PIC X COMP-X.

On entry, contains the character to write

ATTRIBUTE PIC X COMP-X.

On entry, contains the attribute to write

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

CBL_CLOSE_FILE

Usage

```
CAll "CBL_CLOSE_FILE"
     USING HANDLE
     RETURNING STATUS-CODE
```

Parameters

HANDLE (pic x(4) comp-x)

This is the handle returned from CBL_OPEN_FILE or CBL_CREATE_FILE. Once this routine is called, the file handle should not be used in future calls to READ or WRITE or CLOSE, or undefined results will occur, including the possibility of a MAV.

Description

This routine is used for closing files and returns “0” on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_CLOSE_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains “9” and fs-byte-2 contains the ANSI-74 error code, or a runtime system error number.

Note: This routine is written in C and is called via the “direct” method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_COPY_FILE

CBL_COPY_FILE creates a copy of an existing file.

Usage

```
CALL "CBL_COPY_FILE"
    USING SOURCE-FILE, DEST-FILE,
    GIVING COPY-STATUS
```

Parameters

SOURCE-FILE PIC X(n)

Contains the name of the file to copy. The name can contain a path and is terminated by a space. If no path is given, the current directory is assumed. Remote name notation and "@[DISPLAY]:" is allowed for this parameter.

DEST-FILE PIC X(n)

Contains the destination file name. The name can contain a path and is terminated by a space. If no path is given, the current directory is assumed. Remote name notation and "@[DISPLAY]:" is allowed for this parameter.

COPY-STATUS Any numeric type

Returns "0" if successful, or "1" if not.

Description

CBL_COPY_FILE creates an exact duplicate of SOURCE-FILE in DEST-FILE.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

To transfer files between the application host and display host in a thin client environment, add the prefix "@[DISPLAY]:" to the name of any source or destination file that resides on the client machine. For example:

```
CBL_COPY_FILE "@[DISPLAY]:C:\path\file1.ext" "/usr/data/
file1.ext"
```

To copy from one path on the client to another, specify the "@[DISPLAY]:" prefix for both the SOURCE-FILE and the DEST-FILE.

Thin Client and Windows special directories

If the file name on the client starts with special directory specifiers, the thin client attempts to locate those files in special Windows directories. The special directory names are as follows:

Identifier	Directory
<APPDATA>	C:\Documents and Settings\<user>\Application Data
<COMMON_APPDATA>	C:\Documents and Settings\All Users\Application Data
<COMMON_DOCUMENTS>	C:\Documents and Settings\All Users\Documents
<DESKTOP>	C:\Documents and Settings\<user>\Desktop
<LOCAL_APPDATA>	C:\Documents and Settings\<user>\Local Settings\Application Data
<MYDOCUMENTS>	C:\Documents and Settings\<user>\My Documents

Note: These directories are not necessarily the same for all versions of Windows, and may even be on network drives.

CBL_CREATE_DIR

CBL_CREATE_DIR creates a subdirectory. All of the directories in the given path, except the last, must already exist.

Usage

```
CALL "CBL_CREATE_DIR"
    USING DIR-NAME,
    GIVING STATUS
```

Parameters

DIR-NAME PIC X(n)

Contains the name of the directory to be created. This should be either a full path name or a name relative to the current directory. You may use remote name syntax in combination with AcuServer to create a directory on a remote machine. CBL_CREATE_DIR can make a directory only one level lower than an existing directory and cannot create more than one level at a time.

STATUS Any numeric type

Returns “0” if successful, or “1” if not.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

CBL_CREATE_FILE

Usage

```
CAll "CBL_CREATE_FILE"
     USING FILENAME, ACCESS-MODE, DENY-MODE, DEVICE, HANDLE
     RETURNING STATUS-CODE
```

Parameters

FILENAME (PIC X(n))

This is the name of a file to create. The filename parameter can be blank-terminated or terminated with low-values.

ACCESS-MODE (pic x comp-x)

The mode to open the file in. 1 for Input, 2 for Output, 3 for IO. Any other value will result in the creation failing, with a return-code of 1.

If the file exists, it is truncated to 0 bytes, and if it does not exist, it is created.

DENY-MODE (pic x comp-x)

Determines how other users can access the file: “0” to deny read and write access by other users, “1” to deny write access, “2” to deny read access, and “3” to allow all other users. This flag has an effect only on Windows systems.

DEVICE (pic x comp-x)

This is not used and must be “0”.

HANDLE (pic x(4) comp-x)

This is set to the handle of the file created. Use this handle in the other functions.

Description

This routine is used for creating files and returns “0” on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_CREATE_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains “9” and fs-byte-2 contains the ANSI-74 error code, or a runtime system error number.

Note: This routine is written in C and is called via the “direct” method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_DELETE_DIR

CBL_DELETE_DIR deletes the indicated directory. The directory is deleted only if it is empty. You may use remote name syntax in combination with AcuServer to delete a directory on a remote machine.

Usage

```
CALL "CBL_DELETE_DIR"
    USING PATH-NAME,
    GIVING STATUS
```

Parameters

PATH-NAME PIC X(n)

Contains the relative or absolute path name, terminated by space or NULL. The "@[DISPLAY]:" for Thin Client support is allowed. For example:

```
CBL_DELETE_DIR "@[DISPLAY]:C:\path"
```

See the section **Thin Client and Windows special directories** for more information.

STATUS Any numeric type

Returns “0” if successful, or “1” if not.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

CBL_DELETE_FILE

CBL_DELETE_FILE deletes the indicated file.

Usage

```
CALL "CBL_DELETE_FILE"
    USING FILE-NAME,
```

```
    GIVING STATUS
```

Parameters

FILE-NAME PIC X(n)

Contains the name of the file to be deleted. This should either be a full path name or a name relative to the current directory. The "@[DISPLAY]:" annotation for Thin Client is supported. See the section **Thin Client and Windows special directories** for more information.

STATUS Any numeric data type

Returns “0” if successful, or “1” if not.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

CBL_EQ

CBL_EQ performs a binary, bitwise “equals” operation on a series of bytes.

Usage

```
CALL "CBL_EQ"
    USING SOURCE, DEST, LENGTH
    GIVING STATUS
```

Parameters

SOURCE PIC X(n)

The source bytes for the operation.

DEST PIC X(n)

The destination bytes for the operation.

LENGTH Numeric parameter (optional)

The number of bytes to combine. If omitted, then CBL_EQ uses the minimum of the size of SOURCE and the size of DEST.

STATUS Any numeric data item

The return status of the operation. Returns "0" if successful, "1" if not. This routine always succeeds, so STATUS always contains a zero.

Description

For LENGTH bytes, each byte of SOURCE is combined with the corresponding byte of DEST. The result is stored back into DEST. The runtime combines the bytes by performing an "equals" operation between each bit of the bytes. The "equals" operation uses the following table to determine the result:

Eq	**0**	**1**
0	1	0
1	0	1

CBL_ERROR_PROC

CBL_ERROR_PROC installs or removes error procedures to be called automatically if and when the current run unit generates any of certain runtime errors. This implementation calls error procedures only when a run unit generates what is called an *intermediate* runtime error. The list of intermediate runtime errors includes:

- "File error #"
- "File error # on #"
- "Illegal MERGE"
- "Illegal RELEASE"
- "Illegal RETURN"

- “Illegal SORT”
- “Index out of bounds”
- “INSPECT REPLACING size mismatch”
- “Invalid or missing parameter”
- “Non-numeric data in numeric field”
- “Passed USING item smaller than corresponding LINKAGE item”
- “Program missing or inaccessible”
- “Reference modifier range error”
- “Transaction error #”
- “Transaction error # on #”
- “Use of a LINKAGE data item not passed by the caller”

The “#” signs are replaced at run time by error names, numbers, or other information.

Usage

```
CALL "CBL_ERROR_PROC"
    USING INSTALL-FLAG PROGRAM-NAME
    [RETURNING STATUS-CODE]
```

Parameters

INSTALL-FLAG Numeric data item or literal

Zero if the error procedure is to be installed; nonzero if it is to be removed.

PROGRAM-NAME Alphanumeric data item or literal

Name of the error procedure to be installed or removed.

STATUS-CODE Any numeric data item

Always zero. (It is returned only for Micro Focus compatibility.)

Description

A run unit can dynamically build a single queue of one or more error procedures to be called if and when the run unit generates any of certain runtime errors. Not all runtime errors are treated in this way. Some, such as memory allocation errors, are so severe that the runtime cannot continue and must be aborted. Others, such as size errors, are handled by ordinary COBOL code. Runtime errors which call error procedures are called *intermediate* errors.

When an error procedure is installed, it is placed at the beginning of the queue (or moved to the beginning if it is already in the queue).

When an intermediate runtime error occurs, and there are error or exit procedures in the queues, the error procedures in the queue are called, one by one, as though by a CALL statement, in sequential order (the opposite of installation order), with a single PIC X(325) argument containing the text of an appropriate error message. This error message may contain newline characters and is null-terminated like a C string. Each procedure is removed from the queue just before it is called.

Note: You can cause the runtime to also include the program name and the address of the program failure in the string passed to the error procedure by setting the INCLUDE_PGM_INFO runtime configuration variable. See the entry for **INCLUDE_PGM_INFO** in Appendix H.

An error procedure must end with an EXIT PROGRAM RETURNING statement which contains an appropriate return value. If the return value is zero, then subsequent error procedures are removed from the queue and are not called.

After all error procedures are called, the exit procedures, if any, are called. Then the run unit is terminated.

If an error procedure generates an intermediate runtime error, it is terminated and the remaining error procedures, if any, are called. Then the exit procedures, if any, are called, and the run unit is terminated.

If an intermediate runtime error occurs when there are no error or exit procedures in the queues, it is handled in the usual way. In some cases, the error is ignored. In other cases, the runtime issues an error message and terminates the entire application, not just the run unit which generated the error.

Note: An error procedure may install or remove exit procedures and other error procedures. This practice is not generally recommended, however, because it may lead to hard-to-predict behaviors resulting in part from the runtime modifications employed in the handling of error and exit procedures.

Error procedure names are case-insensitive and must not contain spaces.

To prevent the program from entering an infinite loop or non-terminating condition, the total number of error and exit procedures installed or called is limited to the value of the configuration parameter MAX_ERROR_AND_EXIT_PROCS. The default value is 64. Any attempt to exceed this limit aborts the application.

CBL_EXIT_PROC

CBL_EXIT_PROC installs, removes, and queries exit procedures to be called automatically when the current run unit terminates normally.

In some cases exit procedures can be called when a run unit generates an error. See the **CBL_ERROR_PROC** Routine for error-caused terminations of the run unit.

Usage

```
CALL "CBL_EXIT_PROC"
    USING PRIORITY-NUMBER PROGRAM-NAME
    [RETURNING STATUS-CODE]
```

Parameters

PRIORITY-NUMBER Numeric data item or literal

Priority number of the queue or the special value “254” or “255”.

PROGRAM-NAME Alphanumeric data item or literal

Name of the exit procedure to be installed, removed, or queried.

STATUS-CODE Any numeric data item

Queue number when the exit procedure is queried.

Description

A run unit can dynamically build one or more queues of exit procedures to be called if and when the run unit terminates normally or when an error procedure terminates normally.

Exit procedures are kept in queues, one queue for each priority level. Each run unit has its own set of queues, with priorities ranging from 0 to 127, inclusive. The priority queue for an exit procedure is determined when the procedure is installed. Because it is possible for an error or exit procedure to install or remove error or exit procedures, the priorities and queues can change dynamically. The queues and priorities that apply in the end are those in effect at the time when the runtime chooses an error or exit procedure to call.

When the program calls CBL_EXIT_PROC, if the priority-number is in the range from 0 to 127, inclusive, then:

1. The exit procedure is removed from the current run unit, if present;
2. The exit procedure is installed at the beginning of the corresponding queue for the current run unit.

If the priority-number is “254”, the exit procedure is simply removed from the current run unit.

If the priority-number is “255”, the priority number of the queue in the current run unit containing the exit procedure is returned in status-code (or RETURN-CODE, if no RETURNING phrase is present).

(RETURN-CODE is described in Book 3, *ACUCOBOL-GT Reference Manual*, section 6.6, under **CALL Statement**, General Rule 21.) If it is not in any queue in the current run unit, the value "255" is returned.

The installed exit procedures for the current run unit are called when the current run unit terminates by executing a STOP RUN, CALL PROGRAM or CHAIN statement.

An exit procedure cannot start a new run unit by executing a CALL RUN or CHAIN statement. If it tries to do so, the runtime displays an error message and stops without calling any other procedures.

An exit procedure must not call STOP RUN. Its effects are undefined.

Exit procedures in queues with lower priority numbers are called before those in queues with higher priority numbers. Exit procedures in the same queue are called in sequential order (the opposite of installation order). Each exit procedure is removed from its queue just before it is called.

An exit procedure may install, remove, or query other exit procedures.

An exit procedure is called as though by a CALL statement and must return by an EXIT PROGRAM statement.

Exit procedure names are case-insensitive and must not contain spaces.

No arguments are passed to an exit procedure, and an exit procedure may not return a value.

To prevent the program from entering an infinite loop or non-terminating condition, the total number of exit procedures installed or called is limited to the value of the configuration parameter MAX_ERROR_AND_EXIT-PROCS. The default value is 64. Any attempt to exceed this limit aborts the application.

CBL_FLUSH_FILE

Usage

```
CAll "CBL_FLUSH_FILE"
```

```
USING HANDLE
RETURNING STATUS-CODE
```

Parameters

HANDLE (pic x(4) comp-x)

This is the handle returned from **CBL_OPEN_FILE** or **CBL_CREATE_FILE**. Any buffers that have not been flushed to disk will be flushed.

Description

This routine is used for flushing files and returns “0” on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_FLUSH_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains “9” and fs-byte-2 contains the ANSI-74 error code, or a runtime system error number.

Note: This routine is written in C and is called via the “direct” method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_GET_CSR_POS

The CBL_GET_CSR_POS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine returns the cursor position.

Usage

```
CALL "CBL_GET_CSR_POS"
    USING SCREEN-POSITION
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On exit, contains the screen position of the cursor. The top left corner is row 0, column 0.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the position of the cursor on the screen.

Note: This routine is not supported with Thin Client.

CBL_GET_EXIT_INFO

CBL_GET_EXIT_INFO provides an exit procedure with certain information about the termination that invoked it. It is included mainly for compatibility with other COBOL implementations.

Usage

```
CALL "CBL_GET_EXIT_INFO"
    USING EXIT-INFO
    [RETURNING STATUS-CODE]
```

Parameters

EXIT-INFO Group item.

Group item containing four elementary data items laid out as follows:

```
exit-info.
   p-block-size   PIC X(4) COMP-N VALUE 16.
   p-return-code  PIC X(4) COMP-N.
   p-rts-error    PIC X(4) COMP-N.
   p-exit-flags   PIC X(4) COMP-N.
```

STATUS-CODE Any numeric data item.

Return status of the operation.

Description

Termination information is returned in the last three items of exit-info and in status-code. The items returned and their values are as follows:

p-return-code most recent value of RETURN-CODE

p-rts-error error number of the most recent runtime error, or zero if none

p-exit-flags a 32-bit word with flag bits as follows (bit 0 is the least significant bit):

bit	meaning of "1" in this bit
0	always zero
1	always zero
2	terminated by STOP RUN, CALL PROGRAM, or CHAIN
3	always zero
4	terminated by operator
5	always zero
6-31	reserved; always zero

status-code the status of the operation as follows:

Value	meaning
0	success
1006	called from outside exit procedure
1009	value of p-block-size is not 16

CBL_GET_SCR_SIZE

The CBL_GET_SCR_SIZE routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine returns the screen size.

Usage

```
CALL "CBL_GET_SCR_SIZE"
    USING DEPTH, WIDTH
    RETURNING STATUS-CODE
```

Parameters

DEPTH PIC X COMP-X.

On exit, contains the number of lines in the screen

WIDTH PIC X COMP-X.

On exit, contains the number of columns in the screen

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses the DEPTH and WIDTH parameters to return information about the size of the screen.

Note: This routine is not supported with Thin Client.

CBL_NOT

CBL_NOT performs a binary, bitwise “not” operation on a series of bytes.

Usage

```
CALL "CBL_NOT"
    USING DEST, LENGTH
    GIVING STATUS
```

Parameters

DEST PIC X(n)

Data area containing the bytes for the operation.

LENGTH Numeric parameter (optional)

Describes the number of bytes to combine. If omitted, then CBL_NOT uses the size of DEST.

STATUS Any numeric data item

Contains the return status of the operation. Returns zero if successful, 1 if not. This routine always succeeds, so STATUS always contains a zero.

Description

For LENGTH bytes, each byte of DEST is negated in a bitwise fashion. For each bit of each byte, the result is the opposite bit value. The “not” operation uses the following table to determine the result:

Not	
0	1
1	0

CBL_OPEN_FILE

Usage

```
CAll "CBL_OPEN_FILE"
     USING FILENAME, ACCESS-MODE, DENY-MODE, DEVICE, HANDLE
     RETURNING STATUS-CODE
```

Parameters

FILENAME (PIC X(n))

This is the name of a file to open. If this file does not exist, the open fails and sets return-code to “1”. The filename parameter can be blank-terminated or terminated with low-values.

ACCESS-MODE (pic x comp-x)

This is the mode in which to open the file: “1” for Input, “2” for Output, “3” for IO. Any other value will result in the open failing, with a return-code of “1”.

DENY-MODE (pic x comp-x)

Determines how other users can access the file: “0” to deny read and write access by other users, “1” to deny write access, “2” to deny read access, and “3” to allow all other users. This flag has an effect only on Windows systems.

DEVICE (pic x comp-x)

This is not used and must be “0”.

HANDLE (pic x(4) comp-x)

This is set to the handle of the file opened. Use this handle in the other functions.

Description

This routine is used for opening files for reading and/or writing and returns “0” on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_xxx_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains “9” and fs-byte-2 contains the ANSI-74 error code, or a runtime system error number.

Note: This routine is written in C and is called via the “direct” method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_OR

CBL_OR performs a binary, bitwise “or” operation on a series of bytes.

Usage

```
CALL "CBL_OR"
    USING SOURCE, DEST, LENGTH
    GIVING STATUS
```

Parameters

SOURCE PIC X(n)

The source bytes for the operation.

DEST PIC X(n)

The destination bytes for the operation.

LENGTH Numeric parameter (optional)

The number of bytes to combine. If omitted, then CBL_OR uses the minimum of the size of SOURCE and the size of DEST.

STATUS Any numeric data item

The return status of the operation. Returns “0” if successful, “1” if not. This routine always succeeds, so STATUS always contains a zero.

Description

For LENGTH bytes, each byte of SOURCE is combined with the corresponding byte of DEST. The result is stored back into DEST. The runtime combines the bytes by performing an “or” operation between each

bit of the bytes. The “or” operation uses the following table to determine the result:

Or	**0**	**1**
0	0	1
1	1	1

CBL_READ_FILE

Usage

```
CAll "CBL_READ_FILE"
     USING HANDLE, OFFSET, COUNT, FLAGS, BUF
     RETURNING STATUS-CODE
```

Parameters

HANDLE (pic x(4) comp-x)

This is the handle returned from CBL_OPEN_FILE.

OFFSET (pic x(8) comp-x)

This is the offset from which to read from the file, based at “0” to read from the first byte. Note that this is a 64-bit value, allowing access to files larger than 4GB. Note that if your OS or File System does not allow such files, setting this to a value larger than your OS or file system can support will cause undefined results.

COUNT (pic x(4) comp-x)

This is the number of bytes to read. This should not be set to a value larger than the size of the buffer (described below), or you will get undefined results, including a potential MAV.

FLAGS (pic x comp-x)

One special value is recognized. If this parameter is “128”, the size of the file returned is the offset parameter, and the file is not read. The only other allowable value is “0”.

BUF (pic x(n))

This is the buffer that will store the values read from the file. This buffer should be at least count bytes long.

Description

This routine is used for reading files and returns “0” on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_READ_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains “9” and fs-byte-2 contains the ANSI-74 error code or a runtime system error number.

Note: This routine is written in C and is called via the “direct” method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_READ_SCR_ATTRS

The CBL_READ_SCR_ATTRS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine reads a string of attributes from the screen.

Usage

```
CALL "CBL_READ_SCR_ATTRS"
    USING SCREEN-POSITION, ATTRIBUTE-BUFFER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, the screen position at which to start reading (the top left corner is row 0, column 0)

ATTRIBUTE-BUFFER PIC X(N).

On exit, this data item contains the attributes read from the screen. It must be at least the length specified by STRING-LENGTH. Positions in the data item beyond that length are unchanged.

STRING-LENGTH PIC XX COMP-X.

On entry, contains the length of the string to read

On exit, contains the length of the string read when the end of the screen is reached

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location to start the read operation. STRING-LENGTH specifies the length of the string of attributes to be read, starting from SCREEN-POSITION. When the routine exits, ATTRIBUTE-BUFFER contains the string of attributes read.

Note: This routine is not supported with Thin Client.

CBL_READ_SCR_CHARS

The CBL_READ_SCR_CHARS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine reads a string of characters from the screen.

Usage

```
CALL "CBL_READ_SCR_CHARS"
    USING SCREEN-POSITION, CHARACTER-BUFFER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start reading (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On exit, this data item contains the characters read from the screen. It must be at least the length specified by STRING-LENGTH. Positions in the data item beyond that length are unchanged.

STRING-LENGTH PIC XX COMP-X.

On entry, contains the length of the string to read

On exit, contains the length of the string read when the end of the screen is reached

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location to start the read operation. STRING-LENGTH specifies the length of the string of characters to be read, starting from SCREEN-POSITION. When the routine exits, CHARACTER-BUFFER contains the string of characters read.

Note: This routine is not supported with Thin Client.

CBL_READ_SCR_CHATTRS

The CBL_READ_SCR_CHATTRS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine reads a string of characters and their corresponding attributes from the screen.

Usage

```
CALL "CBL_READ_SCR_CHATTRS"
    USING SCREEN-POSITION, CHARACTER-BUFFER, ATTRIBUTE-BUFFER,
        STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start reading (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On exit, this data item contains the characters read from the screen. It must be at least the length specified by STRING-LENGTH. Positions in it beyond that length are unchanged.

ATTRIBUTE-BUFFER PIC X(N).

On exit, this data item contains the attributes read from the screen. It must be at least the length specified by STRING-LENGTH. Positions in the data item beyond that length are unchanged.

STRING-LENGTH PIC XX COMP-X.

On entry, contains the length of the string to read

On exit, contains the length of the string read when the end of the screen is reached

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location to start the read operation. STRING-LENGTH specifies the length of the string to be read, starting from SCREEN-POSITION. When the routine exits, CHARACTER-BUFFER contains the string of characters read, and ATTRIBUTE-BUFFER contains those characters’ attributes.

Note: This routine is not supported with Thin Client.

CBL_SET_CSR_POS

The CBL_SET_CSR_POS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine moves the cursor.

Usage

```
CALL "CBL_SET_CSR_POS"
    USING SCREEN-POSITION
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to put the cursor (the top left corner is row 0, column 0)

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to move the cursor to a particular location on the screen.

Note: This routine is not supported with Thin Client.

CBL_SUBSYSTEM

This routine is used to define a COBOL subsystem, which allows cancelling of all COBOL programs in the subsystem with a single statement.

Usage

This library routine takes two parameters. The first parameter is an opcode, and the second parameter depends on the opcode used:

```
CALL "CBL_SUBSYSTEM" using op-code, parameter GIVING
status-code
```

Parameters

op-code (pic x comp-x)

This contains one of the following values:

0 - declare subsystem

1 - cancel subsystem

2 - remove from subsystem

When op-code = 0, “parameter” is a group item that looks like:

```
ss-handlepic x(2) comp-x.
ss-name-lenpic x(2) comp-x.
ss-namepic x(n).
```

When op-code = 1, “parameter” is:

```
ss-handlepic x(2) comp-x.
```

When op-code = 2, “parameter” is ignored.

On entry:

op-code has the value of the operation to perform, 0, 1, or 2.

When op-code = 0, ss-name-len holds the length of the subsystem program-name field. ss-name holds the subsystem program-name. This must be a COBOL program.

When op-code = 0, ss-handle holds the subsystem handle returned by a function 0 call.

On exit:

When op-code = 0, ss-handle holds the subsystem handle value.

Comments

1. A subsystem is defined as a specified program in an application, plus any subprograms subsequently called by programs already in the subsystem that do not already belong to any other subsystems.
2. opcode 0 declares a subsystem. The routine returns a subsystem handle in ss-handle. The next time the program named is called in an initial state, it becomes part of the subsystem. Any subprograms it calls also become part of the subsystem, unless they are not in an initial state.
3. A program belonging to a subsystem is cancelled only under the following circumstances: it is the object of a CANCEL verb, the program cancels the entire subsystem using opcode 1, or the application executes a STOP RUN or CHAIN statement.
4. opcode 1 cancels all programs in the specified subsystem. If any program in the subsystem is still active, that program is released from the subsystem and is not cancelled.
5. opcode 2 removes the program that called it from any subsystem the program is in. To ensure a program is never included in any subsystem, call this function at the start of each entry into the program.

CBL_SWAP_SCR_CHATTRS

The CBL_SWAP_SCR_CHATTRS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of characters and their attributes over another character string on the screen.

Usage

```
CALL "CBL_SWAP_SCR_CHATTRS"
    USING SCREEN-POSITION, CHARACTER-BUFFER, ATTRIBUTE-BUFFER,
        STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position to start writing (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On entry, contains the characters to write

On exit, this data item contains the characters overwritten on the screen. It must be at least the length specified by STRING-LENGTH. Positions in the data item beyond that length are unchanged.

ATTRIBUTE-BUFFER PIC X(N).

On entry, contains the attributes to write

On exit, this data item contains the attributes overwritten on the screen. It must be at least the length specified by STRING-LENGTH. Positions in the data item beyond that length are unchanged.

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

On exit, this item contains the length of the overwritten string (in cells, that is character-attribute pairs).

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine where to begin the write operation. On entry, CHARACTER-BUFFER and ATTRIBUTE-BUFFER contain the characters to write and their corresponding attributes, respectively, and STRING-LENGTH contains that string's length. On exit, CHARACTER-BUFFER and ATTRIBUTE-BUFFER contain the characters that were replaced on the screen and their corresponding attributes, respectively. STRING-LENGTH is the length of the string that was overwritten.

Note: This routine is not supported with Thin Client.

CBL_WRITE_FILE

Usage

```
CAll "CBL_WRITE_FILE"
     USING HANDLE, OFFSET, COUNT, FLAGS, BUF
     RETURNING STATUS-CODE
```

Parameters

HANDLE (pic x(4) comp-x)

This is the handle returned from CBL_OPEN_FILE.

OFFSET (pic x(8) comp-x)

This is the offset from which to write from the file, based at "0" to read from the first byte. Note that this is a 64-bit value, allowing access to files larger than 4GB. Note that if your OS or File System does not allow such files, setting this to a value larger than your OS or file system can support will cause undefined results.

COUNT (pic x(4) comp-x)

This is the number of bytes to read. This should not be set to a value larger than the size of the buffer (described below), or you will get undefined results, including a potential MAV.

FLAGS (pic x comp-x)

One special value is recognized. This value must be set to "0".

BUF (pic x(n))

This is the buffer that will write to the file at "offset". This buffer should be at least count bytes long.

Description

This routine is used for writing files and returns "0" on success and non-zero if an error occurred. The error is a special encoding of the digit 9 with the ANSI-74 error code, or the runtime system error number if no ANSI-74 error code pertains to the error. If RETURN-CODE is non-zero after calling this routine, you must process it as a file status, for example:

```
01  file-status-group.
    03  file-status     pic xx comp-x.
    03  redefines file-status.
        05  fs-byte-1  pic x.
        05  fs-byte-2  pic x comp-x.
. . .
call "CBL_WRITE_FILE" using parameters
if return-code not = 0
    move return-code to file-status
. . .
```

At this point fs-byte-1 contains "9" and fs-byte-2 contains the ANSI-74 error code or a runtime system error number.

Note: This routine is written in C and is called via the "direct" method, so it is not possible for the runtime to validate parameters for accuracy. Passing unexpected parameters will result in undefined behavior and possibly even a MAV.

CBL_WRITE_SCR_ATTRS

The CBL_WRITE_SCR_ATTRS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of attributes to the screen.

Usage

```
CALL "CBL_WRITE_SCR_ATTRS"
    USING SCREEN-POSITION, ATTRIBUTE-BUFFER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

ATTRIBUTE-BUFFER PIC X(N).

On entry, contains the attributes to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine where on the screen to begin the write operation. ATTRIBUTE-BUFFER contains the string of attributes to write, and STRING-LENGTH contains the length of that string of attributes.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_CHARS

The CBL_WRITE_SCR_CHARS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of characters to the screen.

Usage

```
CALL "CBL_WRITE_SCR_CHARS"
    USING SCREEN-POSITION, CHARACTER-BUFFER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On entry, contains the characters to write

STRING-LENGTH PIC XX COMP-X.

On entry, contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. CHARACTER-BUFFER contains the string of characters to write, and STRING-LENGTH contains the length of that character string.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_CHARS_ATTR

The CBL_WRITE_SCR_CHARS_ATTR routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of characters to the screen, giving them all the same attribute.

Usage

```
CALL "CBL_WRITE_SCR_CHARS_ATTR"
    USING SCREEN-POSITION, CHARACTER-BUFFER, STRING-LENGTH,
        ATTRIBUTE
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On entry, contains the characters to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

ATTRIBUTE PIC X COMP-X.

On entry, contains the attribute to write

STATUS-CODE Any numeric type

Returns "1" if successful, or "0" if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. CHARACTER-BUFFER contains the string of characters to write, and STRING-LENGTH contains the length of that character string. The ATTRIBUTE parameter contains the attribute to apply to the character string.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_CHATTRS

The CBL_WRITE_SCR_CHATTRS routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of characters and their attributes to the screen.

Usage

```
CALL "CBL_WRITE_SCR_CHATTRS"
    USING SCREEN-POSITION, CHARACTER-BUFFER, ATTRIBUTE-BUFFER,
        STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

CHARACTER-BUFFER PIC X(N).

On entry, contains the characters to write

ATTRIBUTE-BUFFER PIC X(N).

On entry, contains the attributes to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. STRING-LENGTH specifies the length of the string to be written,

starting from SCREEN-POSITION. When the routine exits, CHARACTER-BUFFER contains the string of characters written, and ATTRIBUTE-BUFFER contains those characters' attributes.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_N_ATTR

The CBL_WRITE_SCR_N_ATTR routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a specified attribute to a string of positions on the screen.

Usage

```
CALL "CBL_WRITE_SCR_N_ATTR"
    USING SCREEN-POSITION, ATTRIBUTE, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

ATTRIBUTE PIC X COMP-X.

On entry, contains the attribute to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. The ATTRIBUTE parameter contains the attribute to write to the screen location, and STRING-LENGTH is the length of that attribute string.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_N_CHAR

The CBL_WRITE_SCR_N_CHAR routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a specified character to a string of positions on the screen.

Usage

```
CALL "CBL_WRITE_SCR_N_CHAR"
    USING SCREEN-POSITION, CHARACTER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
    03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

CHARACTER PIC X COMP-X.

On entry, contains the character to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. The CHARACTER parameter contains the character to write to the screen location, and STRING-LENGTH is the length of that character string.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_N_CHATTR

The CBL_WRITE_SCR_N_CHATTR routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a specified character and attribute to a string of positions on the screen.

Usage

```
CALL "CBL_WRITE_SCR_N_CHATTR"
   USING SCREEN-POSITION, CHARACTER, ATTRIBUTE, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

SCREEN-POSITION Group item

Group item is defined as follows:

```
01  SCREEN-POSITION.
    03  ROW-NUMBER      PIC X COMP-X.
```

```
03  COLUMN-NUMBER   PIC X COMP-X.
```

On entry, contains the screen position at which to start writing (the top left corner is row 0, column 0)

CHARACTER PIC X COMP-X.

On entry, contains the character to write

ATTRIBUTE PIC X COMP-X.

On entry, contains the attribute to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. Note that the write stops at the end of the screen.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine uses SCREEN-POSITION (in row and column coordinates) to determine the location on the screen to begin the write operation. The CHARACTER parameter contains the character to write to the screen location, and ATTRIBUTE contains that character’s corresponding attribute. STRING-LENGTH is the length of that character/attribute string.

Note: This routine is not supported with Thin Client.

CBL_WRITE_SCR_TTY

The CBL_WRITE_SCR_TTY routine is one of a set of library routines that facilitate reading and writing attributes on the screen. This routine writes a string of characters to the screen starting at the current position and scrolls the screen if the write operation extends beyond the end of the screen.

Usage

```
CALL "CBL_WRITE_SCR_TTY"
    USING CHARACTER-BUFFER, STRING-LENGTH
    RETURNING STATUS-CODE
```

Parameters

CHARACTER-BUFFER PIC X(N).

On entry, contains the characters to write

STRING-LENGTH PIC XX COMP-X.

On entry, this item contains the length of the string to write. If the string length extends off the screen, the screen is scrolled up a line and the write continues on the bottom line.

STATUS-CODE Any numeric type

Returns “1” if successful, or “0” if not successful

Description

This library routine writes the character string in the CHARACTER-BUFFER parameter on the screen. STRING-LENGTH contains the length of that character string. If necessary, the routine allows the write operation to continue past the end of the screen with a scrolling action.

Note: This routine is not supported with Thin Client.

CBL_XOR

CBL_XOR performs a binary, bitwise “exclusive or” operation on a series of bytes.

Usage

```
CALL "CBL_XOR"
```

```
USING SOURCE, DEST, LENGTH
GIVING STATUS
```

Parameters

SOURCE PIC X(n)

The source bytes for the operation.

DEST PIC X(n)

The destination bytes for the operation.

LENGTH Numeric parameter (optional)

Describes the number of bytes to combine. If omitted, then CBL_XOR uses the minimum of the size of SOURCE and the size of DEST.

STATUS Any numeric data item

Contains the return status of the operation. Returns “0” if successful, “1” if not. This routine always succeeds, so STATUS always contains a zero.

Description

For LENGTH bytes, each byte of SOURCE is combined with the corresponding byte of DEST. The result is stored back into DEST. The bytes are combined by performing an “exclusive or” operation between each bit of the bytes. The “exclusive or” operation uses the following table to determine the result:

Xor	**0**	**1**
0	0	1
1	1	0

C$ASYNCPOLL

C$ASYNCPOLL is used only with AcuConnect. It checks the status of the server program while the client is running.

Usage

```
CALL "C$ASYNCPOLL"
    USING HANDLE-OF-CALL, STATE-OF-CALL, PARAMETER(S)
```

Parameters

HANDLE-OF-CALL USAGE HANDLE

Contains the handle of the CALL previously run with C$ASYNCRUN.

STATE-OF-CALL PIC S9

Contains a value of "0" if the run is not completed, or "1" if the run is completed.

PARAMETER(S) Any COBOL data type (optional)

Contains a parameter of the CALL. You may include as many parameters as you choose, separated by a space or comma, to define the CALL. The list of parameters used in C$ASYNCPOLL must match the list of parameters used in C$ASYNCRUN.

Comments

C$ASYNCPOLL tells AcuConnect to query the server about the status of the remote application. AcuConnect returns a status that you can DISPLAY on the client. If the status is "1" or CALL completed, C$ASYNCPOLL terminates the connection with the remote application. This library routine works only with AcuConnect.

C$ASYNCRUN

Call C$ASYNCRUN along with your remote applications to allow AcuConnect to run asynchronously.

Usage

```
CALL "C$ASYNCRUN"
    USING HANDLE-OF-CALL, PROGRAM-NAME, PARAMETER(S)
```

Parameters

HANDLE-OF-CALL USAGE HANDLE

Contains the handle of the CALL defined in working storage.

PROGRAM-NAME String Literal or PIC X(n)

Contains the name of the CALLed program.

PARAMETER(S) Any COBOL data type (optional)

Contains a parameter of the CALL. You may include as many parameters as you choose, separated by a space or comma, to define the CALL.

Comments

By default, AcuConnect performs synchronous CALLs to remote applications. The client application CALLs the remote application and waits for a response from the server to continue processing. C$ASYNCRUN tells AcuConnect to allow the client application to continue processing even while the server application is active. This library routine works only with AcuConnect.

C$CALLEDBY

This routine returns the name of the caller of the currently running COBOL program or spaces if no caller exists or if the caller is unknown.

Usage

```
CALL "C$CALLEDBY"
    USING CALLING-PROGRAM
    GIVING CALL-STATUS
```

Parameters

CALLING-PROGRAM PIC X(n)

Contains the name of the calling program or spaces if no caller exists or if the caller is unknown. The runtime will use as much space for the name or spaces as the COBOL program allows. If the object being called is in an object library, the program returns the PROGRAM-ID. If the object is not in an object library, the disk name is returned.

CALL-STATUS PIC S99

This parameter receives one of the following values:

1	Routine called by another COBOL program
0	Routine is the main program; no caller exists
-1	Caller unknown; routine not called by a COBOL program

C$CALLERR

This routine may be called to retrieve the reason why the last CALL statement failed. For accurate information, it must be called before any other CALL statement is executed.

Usage

```
CALL "C$CALLERR"
    USING ERR-CODE, ERR-MESSAGE
```

Parameters

ERR-CODE PIC X(2)

The single PIC X(2) parameter in this routine receives one of the following values:

01	Program file missing or inaccessible
02	Called file not a COBOL program

03	Corrupted program file
04	Inadequate memory available to load program
05	Unsupported object code version number
06	Recursive CALL of a program
07	Too many external segments
08	Large-model program not supported (returned only by runtimes that do not support large-model programs)
09	Exit Windows and run “share.exe” to run multiple copies of “wrun32.exe” (returned only by Windows runtimes)
14	Japanese objects are not supported (returned only by runtimes that do not support Japanese objects)

ERR-MESSAGE PIC X(n) (optional)

This routine may optionally be passed a second alphanumeric parameter. This parameter is filled in with a descriptive message about the error encountered.

C$CHAIN

The C$CHAIN routine replaces the running program and runtime system with another program.

Usage

```
CALL "C$CHAIN"
    USING PROG-NAME
```

Parameter

PROG-NAME PIC X(n)

Contains an operating system command line to execute.

Description

This routine functions in the same manner as the SYSTEM library routine, except that there is no return to the running program. Instead, the current program shuts down (like a STOP RUN) and the runtime system then replaces itself with the passed program. This is similar to the CHAIN verb except that the called program is not a COBOL program; it is any program available on the host machine.

The usual reason for calling this routine is to make more memory available to the called program. The runtime system can occupy a significant amount of memory that may be needed by the called program. Calling C$CHAIN ensures that the runtime system is removed from memory along with the various COBOL programs that have been active.

Often it is desirable to return from the called program to the caller. One way to do this is to use an operating system script to re-execute the calling program. You can control the script with various exit statuses--for a description of the runtime's exit statuses, see the entry for the **STOP Statement** in section 6.6 of the *ACUCOBOL-GT Reference Manual*. Usually the script will sit in a loop calling the ACUCOBOL-GT runtime system until it receives a special exit status. You can use the STOP RUN statement to pass that special status back to the script when you want to shut down. You should remember that when the runtime system chains to another program, the script sees the exit status of the called program as the exit status of the runtime system.

For example, suppose on a UNIX system that you have a program (called **main**) that executes the C$CHAIN routine. After the called program terminates, you want to re-execute the caller. You decide to use exit value "100" to indicate that the program should terminate. You could use the following "sh" script:

```
runcbl main
while test $? != 100; do runcbl main; done
```

This script runs **main** once, and then continues to run it until it executes a "STOP RUN 100" statement. Writing the script this way allows the program that is called by C$CHAIN to have a non-zero exit value without stopping the entire run.

You may want to distinguish between the first execution of the calling program and subsequent executions. For example, you may want to display copyright information on the first execution. You can do this by using a SPECIAL-NAMES switch to distinguish between the first execution and subsequent ones. For instance, in the preceding example you could add “-1” to the second **runcbl** command to set SWITCH-1 for the subsequent executions. Your program can then test the value of this switch to determine what to do.

C$CHDIR

The C$CHDIR routine is used to change the current working directory.

Usage

```
CALL "C$CHDIR"
    USING DIR-NAME, ERR-NUM
```

Parameters

DIR-NAME PIC X(n)

Contains the name of the new directory, or spaces. The "@[DISPLAY]:" for Thin Client support is allowed. For example:

```
C$CHDIR "@[DISPLAY]:C:\path"
```

In Thin Client environments, to get the current default directory on the display host, DIR_NAME should contain "@[DISPLAY]:" followed by spaces.

See the section **Thin Client and Windows special directories** for more information.

ERR-NUM PIC 9(9) COMP-4 (optional)

Holds the returned error number, or zero on success.

Comments

If a second USING parameter is passed, it must be described as PIC 9(9) COMP-4. This parameter will be set to ZERO if the directory change is successful. Otherwise, it will contain the operating system's error number.

If DIR-NAME contains spaces, then the current default directory is returned in it. In this case, ERR-NUM is not used. Otherwise, DIR-NAME should contain the name of a directory to make the new default directory. On Windows machines, this can include a drive letter. If you pass ERR-NUM, it will be set to zero if the change was successful. Otherwise, ERR-NUM will contain the error value returned by the operating system.

On some systems (such as VMS), it is legal to switch to a directory that does not exist, while other systems (Windows, UNIX) do not allow it.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

IMPORTANT

If you use C$CHDIR, create a **CODE_PREFIX** configuration entry to locate your object files. Ensure that all of the search locations specified by the CODE-PREFIX are full path names. Do not use the current directory or any relative path names in the CODE_PREFIX. Without a full path name, the runtime system may be unable to find your object files if it needs to re-open them.

For example, the runtime system must occasionally re-open an object file when:

- you are using the source debugger
- the program contains segmentation (overlays)
- you are using object libraries

If the object file was initially found in the current directory or a directory specified relative to the current directory, and you then change the current directory with the C$CHDIR routine, the runtime system will not be able to find the object file if it needs to re-open it. This will cause a fatal error and your program will halt.

If you use C$CHDIR and you are running in debug mode, be sure to set CODE_PREFIX in the configuration file, *not* in the environment. You may set CODE_PREFIX in the environment when you are not in debug mode.

Considerations for AcuBench users

When you import a screen with the Screen Import Utility, a file called "import.out" is created in the current working directory. The Screen Designer uses this file to load the description(s) of the imported screen(s) into its designer grid and property sheet.

When you invoke the Screen Import Utility from within the Workbench, the act of exiting the runtime also causes the Screen Designer to load the import.out file which is located in the root directory of the project currently open.

If your application changes the current working directory, the import.out file will not be created in the root directory of the currently open project, so different, perhaps unexpected behaviors can occur.

Import.out files will be generated correctly, and the Screen Designer will be able to import them correctly, but you will have to locate the files for the Screen Designer and open them "manually" using the File/Open dialog box in the Screen Designer.

C$CODESET

This routine supports the translation of a text string from EBCDIC to ASCII, or ASCII to EBCDIC.

Usage

```
CALL "C$CODESET"
    USING TRANS-FLAG, LENGTH, TRANS-STRING
```

```
    GIVING RETURN-VALUE.
```

Parameters

TRANS-FLAG PIC 9(2) COMP-X.

Indicates the type of text in TRANS-STRING, and whether to apply LENGTH when performing the translation. TRANS-FLAG takes one of the following values:

0 Indicates that TRANS-STRING contains EBCDIC and that LENGTH specifies the length of the string to translate to ASCII.

1 Indicates that TRANS-STRING contains ASCII and that LENGTH specifies the length of the string to translate to EBCDIC.

2 Indicates that TRANS-STRING contains EBCDIC and that 256 bytes of data should be translated to ASCII. The LENGTH parameter is ignored.

3 Indicates that TRANS-STRING contains ASCII and that 256 bytes of data should be translated to EBCDIC. The LENGTH parameter is ignored.

LENGTH PIC 9(9) COMP-X

Specifies the length of the string to translate.

TRANS-STRING PIC X(n)

Contains the string to translate and the result of the translation.

RETURN-VALUE Numeric data item

Returns the number of characters translated, or zero if an error occurred.

C$CONFIG

C$CONFIG resets configuration values to a known set of values. Note, however, that certain values are set only at runtime start up, and are therefore not affected by this routine.

Usage

```
CALL "C$CONFIG"
    USING OP-CODE, CONFIG-FILE
```

Parameters

OP-CODE PIC 9

The only valid value is 1. If the op-code is specified with no arguments, C$CONFIG resets all known configuration values to their default. Values are removed from any user-defined configuration variables, leaving them blank, and the original configuration file is reloaded.

CONFIG-FILE PIC X(n) (optional)

Specifies a new configuration file to be loaded after the configuration variable values have been reset.

Description

C$CONFIG resets all runtime configuration variables first to their default values, then to the values specified in CONFIG-FILE, if CONFIG-FILE is specified.

If you pass a blank argument for CONFIG-FILE, as shown below, all configuration variables are reset and no configuration file is loaded.

```
CALL "C$CONFIG" USING 1, ""
```

C$COPY

C$COPY creates a copy of an existing file.

Usage

```
CALL "C$COPY"
    USING SOURCE-FILE, DEST-FILE, FILE-TYPE,
    GIVING COPY-STATUS
```

Parameters

SOURCE-FILE PIC X(n)

Contains the name of the file to copy. Remote name notation and "@[DISPLAY]:" notation are allowed for this parameter. See the section **Thin Client and Windows special directories** for more information.

DEST-FILE PIC X(n)

Contains the destination file name. Remote name notation and "@[DISPLAY]:" notation are allowed for this parameter.

FILE-TYPE PIC X (optional)

Indicates the file type. If the FILE-TYPE parameter is supplied, it must be either "S", "R", "I", or "T", indicating that the source file is a sequential, relative, indexed file, or text file. Note that the "T" implies that the file is a line sequential file, since copying relative or indexed files as text is counter-intuitive and would likely corrupt those types of files. Copying text files applies when copying between UNIX and Windows systems, where text files have different line terminator characters.

The FILE-TYPE parameter can be useful in cases where the original file is held in more than one physical disk file (for example, C-ISAM indexed files are physically held in two separate files). If the FILE-TYPE parameter is omitted, then only the single physical file named in SOURCE-FILE is copied.

COPY-STATUS Any numeric type

Returns zero if successful, or non-zero if not. Currently, an unsuccessful status code is always "1", but future versions may return additional information.

Description

C$COPY creates an exact duplicate of SOURCE-FILE in DEST-FILE.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

To transfer files between the application host and display host in a thin client environment, add the prefix "@[DISPLAY]:" to the name of any source or destination file that resides on the client machine.

```
C$COPY "@[DISPLAY]:C:\path\file1.ext" "/usr/data/file1.ext"
```

To copy from one path on the client to another, specify the "@[DISPLAY]:" prefix for both the SOURCE-FILE and the DEST-FILE.

If the file name on the client starts with special directory specifiers, the thin client attempts to locate those files in special Windows directories. The special directory names are as follows:

Identifier	**Directory**
<APPDATA>	C:\Documents and Settings\<user>\Application Data
<COMMON_APPDATA>	C:\Documents and Settings\All Users\Application Data
<COMMON_DOCUMENTS>	C:\Documents and Settings\All Users\Documents
<DESKTOP>	C:\Documents and Settings\<user>\Desktop
<LOCAL_APPDATA>	C:\Documents and Settings\<user>\Local Settings\Application Data
<MYDOCUMENTS>	C:\Documents and Settings\<user>\My Documents

Note that these directories are not necessarily the same for all versions of Windows, and may in fact be on network drives.

If you use the "@[DISPLAY]:" prefix, you may not use the FILE-TYPE parameter. Only the single, specified source file is copied.

See the *AcuConnect User's Guide*, section 7.2.1, for more information about using C$COPY in a thin client environment.

C$DELETE

C$DELETE deletes the indicated file.

Usage

```
CALL "C$DELETE"
    USING FILE-NAME, FILE-TYPE,
    GIVING STATUS
```

Parameters

FILE-NAME PIC X(n)

Contains the name of the file to be deleted. This should either be a full path name or a name relative to the current directory. The "@[DISPLAY]:" annonation for Thin Client is allowed. For example:

```
C$DELETE "@[DISPLAY]:C:\path\filename"
```

See the section **Thin Client and Windows special directories** for more information.

FILE-TYPE PIC X (optional)

Indicates the file type. If the FILE-TYPE parameter is supplied, it must be either “S”, “R”, or “I” indicating that the source file is a sequential, relative, or indexed file. This can be useful in cases where the original file is held in more than one physical disk file (for example, C-ISAM indexed files and Vision 4 and 5 files are physically held in two separate files). If the FILE-TYPE parameter is omitted, then only the single physical file named in FILE-NAME is deleted.

If you use the "@[DISPLAY]:" prefix, you may not use the FILE-TYPE parameter. Only the single, specified source file is deleted."

STATUS PIC 9(n)

Returns “0” if successful, or “1” if not.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

C$DISCONNECT

C$DISCONNECT disconnects the executing AcuServer client from the specified server at the time of the call.

Under normal circumstances, the AcuServer client disconnects from the server automatically (without requiring you to call the C$DISCONNECT routine) at shutdown. Also, whenever a server loses its connection to a client, before a new connection is made, all old client connections are automatically closed.

Usage

```
CALL "C$DISCONNECT"
    USING SERVER-NAME, PORT-NUMBER
    GIVING STATUS
```

Parameters

SERVER-NAME PIC X(n)

Contains the name of the server computer from which you are disconnecting. The name will be terminated at the first space character.

PORT-NUMBER PIC 9(n) (optional)

Specifies the port number of SERVER-NAME. If omitted, this parameter defaults to the value of the configuration variable SERVER-PORT.

STATUS PIC 9(n)

This routine returns the one of the following status codes after the operation has been performed:

0	Client is now disconnected from the server
1	Server/port combination was not found; no such connection
2	Client access is not enabled

Comments

Any open files on the server should be closed before calling this routine. When these files are reopened, it is as if the original connection had never existed. In particular, any passwords needed to connect to the server will need to be reentered by the user.

C$EXCEPINFO

C$EXCEPINFO retrieves information about an object exception that has been raised.

Usage

```
CALL "C$EXCEPINFO"
    USING ERROR-INFO, ERR-SOURCE, ERR-DESCRIPTION,
        ERR-HELP-FILE, ERR-HELP-CONTEXT, ERR-OBJECT-HANDLE,
        ERR-CONTROL-ID
```

Parameters

ERROR-INFO Group item to receive returned information.

ERROR-INFO must have the following structure (defined in “activex.def”):

```
01 ERROR-INFO.
   03  ERROR-INFO-RESULT    USAGE UNSIGNED-INT.
   03  ERROR-INFO-FACILITY  USAGE UNSIGNED-SHORT.
   03  ERROR-INFO-CODE      USAGE UNSIGNED-SHORT.
```

ERROR-INFO is described in the Comments section below.

ERR-SOURCE PIC X(n) (optional)

A text string identifying the source of the exception. Typically, this is an application name.

ERR-DESCRIPTION PIC X(n) (optional)

A text description of the error intended for the user. If no description is available, ERR-DESCRIPTION is filled with spaces.

ERR-HELP-FILE PIC X(n) (optional)

The fully qualified drive, path, and file name of a help file with more information about the error. If no help is available, ERR-HELP-FILE is filled with spaces.

ERR-HELP-CONTEXT Usage unsigned-long (optional)

The help context ID of the topic within the help file. This parameter is filled in only if the ERR-HELP-FILE parameter is not spaces. The help context ID will be between 0 and 4294967295.

ERR-OBJECT-HANDLE Usage handle (optional)

The handle of the COM object or ActiveX control that generated the exception.

ERR-CONTROL-ID Usage PIC XX COMP-X (optional)

The ID of the ActiveX control that generated the exception. The value is "0" if the object that generated the exception is not an ActiveX control.

Comments

C$EXCEPINFO is typically called from an error handling procedure specified with a Format 2 USE statement in the declaratives.

The following are the parameters for the ERROR-INFO group item:

ERROR-INFO-RESULT is a 32-bit number identifying the error. The error number will be greater than 1000 and less than 4294967296. This error number is divided into 4 fields: a severity code, a reserved portion, a facility field, and an error code.

The severity code is the high-order bit (31). The next 4 bits are reserved (30-27). The next eleven bits are the facility field (26-16), and the last sixteen bits are the error code (15-0). The severity code is usually "1", and the reserved bits are usually set to "0". The error code is defined by the facility that raised the exception. The facility field is one of the following:

FACILITY-ACU (4)	For status codes of exceptions raised by the ACUCOBOL-GT runtime.
FACILITY-ACTIVE-X (10)	For status codes of exceptions raised by an ActiveX control.

The standard facility field values and error codes are defined as condition names (level 88) in "activex.def".

Note: If you receive a facility value that is anything other than the two listed above, you will need to look up the ERROR-INFO result number in the documentation for the specific ActiveX control you are using or in Microsoft's documentation.

ERROR-INFO-FACILITY contains the facility field extracted from ERROR-INFO-RESULT.

ERROR-INFO-CODE contains the error code extracted from ERROR-INFO-RESULT. ERROR-INFO-CODE will be listed in the ACUCOBOL-GT error codes if the facility field is FACILITY-ACU. If the facility is FACILITY-ACTIVE-X, the ERROR-INFO-CODE may be listed in an enumeration in the ActiveX control's COPY file or it may be one of the standard COM status codes for ActiveX controls.

When called to get information about an object exception, the error code can either be an ACUCOBOL-GT defined code or a COM status code.

ACUCOBOL-GT Error Codes (defined in "activex.def"):

Name	Description
ACU-E-UNEXPECTED	Unexpected error
ACU-E-INVALIDPARAMNAME	Invalid parameter name
ACU-E-INVALIDHANDLE	Invalid handle

Name	Description
ACU-E-INITIALSTATE	Error loading INITIAL-STATE from resource file
ACU-E-NOEXCEPINFO	No exception information available
ACU-E-INVALIDPROPNUM	Invalid property number
ACU-E-TOOMANYPARAMS	Too many parameters
ACU-E-TOOFEWPARAMS	Too few parameters
ACU-E-NOTPROPERTYGET	Property can be modified but not inquired
ACU-E-NOTPROPERTYPUT	Property can be inquired but not modified
ACU-E-CREATE	Error creating ActiveX control

COM Status Codes for ActiveX controls (defined in “activex.def”):

Name	Description
AX-E-ILLEGALFUNCTIONCALL	Illegal function call
AX-E-OVERFLOW	Overflow
AX-E-OUTOFMEMORY	Out of memory
AX-E-DIVISIONBYZERO	Division by zero
AX-E-OUTOFSTRINGSPACE	Out of string space
AX-E-OUTOFSTACKSPACE	Out of stack space
AX-E-BADFILENAMEORNUMBER	Bad file name or number
AX-E-FILENOTFOUND	File not found
AX-E-BADFILEMODE	Bad file mode
AX-E-FILEALREADYOPEN	File already open
AX-E-DEVICEIOERROR	Device I/O error
AX-E-FILEALREADYEXISTS	File already exists
AX-E-BADRECORDLENGTH	Bad record length
AX-E-DISKFULL	Disk full
AX-E-BADRECORDNUMBER	Bad record number
AX-E-BADFILENAME	Bad file name

Name	Description
AX-E-TOOMANYFILES	Too many files
AX-E-DEVICEUNAVAILABLE	Device unavailable
AX-E-PERMISSIONDENIED	Permission denied
AX-E-DISKNOTREADY	Disk not ready
AX-E-PATHFILEACCESSERROR	Path/file access error
AX-E-PATHNOTFOUND	Path not found
AX-E-INVALIDPATTERNSTRING	Invalid pattern string
AX-E-INVALIDUSEOFNULL	Invalid use of NULL
AX-E-INVALIDFILEFORMAT	Invalid file format
AX-E-INVALIDPROPERTYVALUE	Invalid property value
AX-E-INVALIDPROPERTYARRAYINDEX	Invalid property array index
AX-E-SETNOTSUPPORTEDATRUNTIME	Set not supported at run time
AX-E-SETNOTSUPPORTED	Set not supported (read-only property)
AX-E-NEEDPROPERTYARRAYINDEX	Need property array index
AX-E-SETNOTPERMITTED	Set not permitted
AX-E-GETNOTSUPPORTEDATRUNTIME	Get not supported at run time
AX-E-GETNOTSUPPORTED	Get not supported (write-only property)
AX-E-PROPERTYNOTFOUND	Property not found
AX-E-INVALIDCLIPBOARDFORMAT	Invalid clipboard format
AX-E-INVALIDPICTURE	Invalid picture
AX-E-PRINTERERROR	Printer error
AX-E-CANTSAVEFILETOTEMP	Can’t save file to TEMP
AX-E-SEARCHTEXTNOTFOUND	Search text not found
AX-E-REPLACEMENTSTOOLONG	Replacements too long

For an example of how this works, let’s look at the standard COM error code for “Out of Memory”.

The code is hex 800A0007.

In binary this is “1000 0000 0000 1010 0000 0000 0000 0111”.

This error code can be broken down as follows:

1	000 0	000 0000 1010	0000 0000 0000 0111
Severity Code	Reserved	Facility Field	Error Code

The severity code is 1 which makes ERROR-INFO-RESULT greater than or equal to 2147483648 (hex 80000000). The reserved portion is 0. The facility code is 10 (hex A) which is defined in “activex.def” as FACILITY-ACTIVE-X. The error code is 7 (hex 7) which is defined in “activex.def” as AX-E-OUTOFMEMORY.

Example

```
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY "ACTIVEX.DEF"
77  ERR-SOURCE          PIC X(30).
77  ERR-DESCRIPTION     PIC X(200).
77  ERR-HELP-FILE       PIC X(200).
77  ERR-HELP-CONTEXT    USAGE UNSIGNED-LONG.
77  CHOICE              PIC 9.

PROCEDURE DIVISION.
DECLARATIVES.
OBJECT-EXCEPTION-HANDLING SECTION.
    USE AFTER EXCEPTION ON OBJECT.
OBJECT-EXCEPTION-HANDLER.
    CALL "C$EXCEPINFO" USING ERROR-INFO, ERR-SOURCE,
        ERR-DESCRIPTION, ERR-HELP-FILE,
        ERR-HELP-CONTEXT.
    IF ERR-HELP-FILE = SPACES THEN
        DISPLAY MESSAGE BOX ERR-DESCRIPTION
            TITLE ERR-SOURCE
            ICON MB-ERROR-ICON
    ELSE
        DISPLAY MESSAGE BOX ERR-DESCRIPTION H'0d'
            "Do you want help ?"
            TITLE ERR-SOURCE
            ICON MB-ERROR-ICON
            TYPE IS MB-YES-NO
            DEFAULT IS MB-YES
```

```
                GIVING CHOICE
            IF CHOICE = 1 THEN
                CALL "$WINHELP" USING ERR-HELP-FILE, HELP-CONTEXT
                    ERR-HELP-CONTEXT
            END-IF
        END-IF.
        EVALUATE TRUE
            WHEN FACILITY-ACU
                EVALUATE TRUE
                    WHEN ACU-E-UNEXPECTED
                        STOP RUN
                END-EVALUATE
            END-EVALUATE.
END DECLARATIVES.

MAIN-PROGRAM SECTION.
{ . . . }
* The following line causes an AX-E-SETNOTSUPPORTEDATRUNTIME
* exception since Microsoft Chart's BorderStyle property is not
* allowed to be modified at runtime
    MODIFY MS-CHART-1 BorderStyle = VtBorderStyleBold.
```

C$EXITINFO

The C$EXITINFO library routine returns information about an exit from an END procedure in the declaratives.

Usage

```
CALL "C$EXITINFO"
    USING EXIT-MESSAGE, EXIT-CODE, OS-EXIT-CODE, SIGNAL-NUMBER
```

Parameters

EXIT-MESSAGE Alphanumeric

Contains the lines of text that the runtime outputs to the error file or shutdown message box. Each line of text is separated by a newline character H‘0A’.

EXIT-CODE Numeric (optional)

Contains one of the following codes (listed in “lib/sub.h”):

```
1     COBOL_EXIT_PROGRAM
2     COBOL_REMOTE_CALL
3     COBOL_STOP_RUN
4     COBOL_CALL_ERROR
5     COBOL_SIGNAL
6     COBOL_FATAL_ERROR
7     COBOL_NONFATAL_ERROR
8     COBOL_DEBUGGER
```

OS-EXIT-CODE Numeric (optional)

Contains the value that the process passes to the system exit routine

SIGNAL-NUMBER Numeric (optional)

This is always “0” on Windows. On UNIX, it contains the signal number if EXIT-CODE is “5” (COBOL_SIGNAL).

C$FILEINFO

C$FILEINFO retrieves some operating system information about a given file.

Usage

```
CALL "C$FILEINFO"
    USING FILE-NAME, FILE-INFO,
    GIVING STATUS-CODE
```

Parameters

FILE-NAME PIC X(n)

Contains the name of the file to check. This should either be a full path name or a name relative to the current directory. Remote name notation and “@[DISPLAY:]” notation are allowed for this parameter. If run in standalone mode, and the “@[DISPLAY]” syntax is used, the file will be searched for on the local machine.

FILE-INFO Group item to receive returned information.

FILE-INFO must have the following structure:

```
01  FILE-INFO.
    02  FILE-SIZE    PIC X(8) COMP-X.
    02  FILE-DATE    PIC 9(8) COMP-X.
    02  FILE-TIME    PIC 9(8) COMP-X.
```

STATUS-CODE Any numeric data item.

This receives the return status. It will be zero if successful, or "1" if the file does not exist or is not a regular disk file.

Description

This routine checks to see if the passed file exists and is a regular disk file. If it is, then FILE-INFO is filled in with the appropriate information. The FILE-SIZE field is the size of the file in bytes. The FILE-DATE and FILE-TIME fields indicate the time the file was last modified. FILE-DATE has the form "YYYYMMDD" (year/month/day, note the 4-digit year) and FILE-TIME has the form "HHMMSShh" (hours/minutes/seconds/hundredths--just like ACCEPT FROM TIME). On all current implementations, the hundredths field is always set to zero.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

C$FILESYS

C$FILESYS performs one of three functions, depending on the operations code passed to it. It retrieves the names of file systems known to the runtime, determines whether a particular file system is known, or returns a count of all known file systems.

Usage

```
CALL "C$FILESYS"
    USING OP-CODE, FILE-SYSTEM
```

Parameters

OP-CODE PIC 9

The value indicates the desired operation.

FILE-SYSTEM PIC X(5)

Passes or returns a file system name. The names that are recognized are up to five characters long and are listed in the program **filetbl.c**, which is linked into the runtime. (For example, use “visio” for Vision, “infor” for Informix, “oracl” for Oracle, and “sybas” for Sybase.)

Comments

This routine can determine which file systems are known to the runtime, and can give a count of known systems.

The file system names are up to five characters long, and are listed in **filetbl.c**, which must be linked into the runtime. The (activated) names listed in **filetbl.c** are the names that are returned by this routine, and the only names that the routine recognizes.

If OP_CODE is “0”, both parameters are required. The routine returns the first file system known to the runtime and gets set up to return the rest. RETURN-CODE will be set to “1” if there was a file system to return, or “0” if not.

If op-code is “1”, both parameters are required. The routine returns the next file system known to the runtime. RETURN-CODE is set to “1” if there was a file system to return, or “0” if not (there are no more). You can call the routine with this parameter repeatedly, until RETURN-CODE is set to “0”.

If op-code is “2”, both parameters are required. The routine checks to see if the passed file system is known to the runtime. You must pass one of the names used in the program **filetbl.c**. (The routine checks only the first five characters of the name you pass; the rest are ignored.) The case of the name does not matter. For example, “SYBAS” and “sybas” are equivalent. RETURN-CODE is set to “1” if yes, or “0” if no.

If op-code is “3”, the second parameter is not required. RETURN-CODE is set to the number of known file systems.

C$FULLNAME

C$FULLNAME locates a file by using the runtime’s filename translation and search logic. The routine returns the full name of the corresponding file.

Usage

```
CALL "C$FULLNAME"
    USING FILE-NAME, FULL-NAME, FILE-INFO
    GIVING STATUS-CODE
```

Parameters

FILE-NAME PIC X(n)

Specifies the name of the file to be located.

FULL-NAME PIC X(n)

Data item to receive the full name of the file.

FILE-INFO Group item (optional)

Group item to receive information about the file located. Must have this structure:

```
01  FILE-INFO.
    02  FILE-SIZE    PIC X(8) COMP-X.
    02  FILE-DATE    PIC 9(8) COMP-X.
    02  FILE-TIME    PIC 9(8) COMP-X.
```

The FILE-SIZE field is the size of the file in bytes. The FILE-DATE and FILE-TIME fields indicate the time the file was last modified. FILE-DATE has the form “YYYYMMDD” (year/month/day, note the 4-digit year) and FILE-TIME has the form “HHMMSShh” (hours/minutes/seconds/hundredths--just like ACCEPT FROM TIME). On all current implementations, the hundredths field is always set to zero.

STATUS-CODE Any numeric data item.

This receives the return status. It will be zero if successful, or "1" if FILE-NAME could not be found on disk or if FILE-NAME is not the name of a regular disk file.

Description

C$FULLNAME invokes the runtime's data file search logic on FILE-NAME. This involves using the configuration variables **FILE_PREFIX**, **FILE_SUFFIX** and **FILE_CASE**. The disk is searched using the rules described in **Section 2.9** of the *ACUCOBOL-GT User's Guide*. The first matching file is returned in FULL-NAME. Note that the returned name is not automatically a full path name. Instead, it is the name of the file as the runtime found it. This may be based on the current directory.

If FILE-INFO is specified, information about the file located is returned in it.

Note: The search techniques do not involve any file system specific techniques. For example, the Vision file system has a name mapping facility for its additional segments. This facility is not used when the runtime is searching for the file. Also note that remote file names are searched for when used in conjunction with AcuServer.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

C$GETCGI

The C$GETCGI routine retrieves CGI (Common Gateway Interface) variables.

Usage

```
CALL "C$GETCGI"
    USING VARIABLE-NAME, DEST-ITEM, VALUE-INDEX
    GIVING VALUE-SIZE
```

Parameters

VARIABLE-NAME PIC X(n)

Contains the name of the CGI variable.

DEST-ITEM PIC X(n)

Receives the value of the given CGI variable.

VALUE-INDEX Numeric value

Contains the CGI value index. This optional parameter contains an index that is used when a CGI variable has multiple values in the CGI input data. This typically happens when multiple items have been selected from a “choose many” list box. For example, to receive the third selected value, pass 3 for VALUE-INDEX. If VALUE-INDEX is greater than the total number of values in the input stream for the given CGI variable, then spaces are moved to DEST-ITEM.

VALUE-SIZE Signed numeric value

Receives the size of the resulting value. This may be “0” to indicate that the variable exists but has no value, or “-1” to indicate that the variable does not exist.

Description

This library routine is intended to be called from COBOL CGI scripts to retrieve CGI variables from either the environment or the standard input stream, “stdin”.

C$GETCGI automatically determines whether to read the CGI variable from the environment or “stdin” depending on the value of the “REQUEST_METHOD” environment variable, which is set by the Web Server. The first time C$GETCGI is called, it reads all of the CGI variables and values into a variable length buffer. If REQUEST_METHOD is “GET”, then the data is read from the “QUERY_STRING” environment variable. If the REQUEST_METHOD is “POST”, then it is read from “stdin”.

Each time C$GETCGI is called, it searches for the variable name passed in the first parameter, translates the value from CGI format into standard format, and moves the result to the destination item passed in the second parameter. An optional third parameter specifies a CGI value index. This index is used when a CGI variable has multiple values in the CGI input data, such as the case where multiple items have been selected from a "choose-many" list.

The size of the resulting value is returned in the special RETURN-CODE register. This may be "0" to indicate that the variable has no value or "-1" to indicate that the variable does not exist. The value of RETURN-CODE is moved to VALUE-SIZE.

The value is truncated to the size of the destination item.

You may use C$GETCGI instead of or in combination with external forms. Use C$GETCGI if you must know the exact size of a CGI variable or if you are converting an existing COBOL CGI program.

C$GETERRORFILE

This routine returns the name of the runtime error file as specified with the runtime "-e" command-line option or with a call to the **C$SETERRORFILE** routine.

Usage

```
CALL "C$GETERRORFILE"
    USING ERROR-FILE-NAME
```

Parameters

ERROR-FILE-NAME PIC X(n)

Contains the name of the error file or spaces if no error file was specified.

C$GETEVENTDATA

When an ActiveX control or COM object generates an event, it makes information about the event available through event parameters. These parameters are stored in the control and can be retrieved by calling C$GETEVENTDATA from the control's event procedure.

Usage

```
CALL "C$GETEVENTDATA"
   USING EVENT-CONTROL-HANDLE, DEST-ITEM-1, [DEST-ITEM-2, ...]
    GIVING RESULT-CODE
```

Parameters

EVENT-CONTROL-HANDLE USAGE HANDLE

Handle to the control that generated the event.

DEST-ITEM-1 Any COBOL data type

The first destination data item.

DEST-ITEM-2, ... Any COBOL data type (optional)

Any number of destination items.

RESULT-CODE Signed numeric value

Receives the result-code for the operation. This will be 0 to indicate success or a negative value to indicate failure. (Microsoft defines many standard "result codes" in their documentation. Note that these are usually in hexadecimal notation.)

Comments

C$GETEVENTDATA converts the event parameters to COBOL-type data in the destination items.

You are responsible for specifying compatible types. For instance, if the event parameter is a character string and you specify a numeric item to receive its value, C$GETEVENTDATA will try to read a number from the string and move it to your numeric item. This is not a programming error and neither the compiler nor runtime will warn you about it.

Example

Suppose you have displayed an ActiveX control that triggers an event called AxEventOne which has three parameters. You would use the following COBOL syntax to get the event parameters into data items PARAM-1, PARAM-2 and PARAM-3:

```
WHEN AxEventOne
  CALL "C$GETEVENTDATA" USING EVENT-CONTROL-HANDLE,
     PARAM-1, PARAM-2, PARAM-3
```

For more examples of how to get event parameters for ActiveX events, refer to section 4.4 in *A Guide to Interoperating with ACUCOBOL-GT.*

C$GETEVENTPARAM

C$GETEVENTPARAM is an alternate way to get event parameters for ActiveX controls. Use it to get a single event parameter when there are several for an event. To use this routine you must know the actual name of the parameter. You may determine these names by reading the ActiveX control's documentation or by looking at the definitions in the copy book for the ActiveX control.

It is common for an ActiveX event to receive many parameters. C$GETEVENTPARAM allows you to get the values of only the parameters you care about.

Please note that C$GETEVENTPARAM cannot be used to get event parameters for COM objects. You must use C$GETEVENTDATA for COM objects.

Usage

```
CALL "C$GETEVENTPARAM"
    USING EVENT-CONTROL-HANDLE, PARAM-NAME, PARAM-VALUE
```

```
    GIVING RESULT-CODE
```

Parameters

EVENT-CONTROL-HANDLE USAGE HANDLE

Handle to the control that generated the event.

PARAM-NAME PIC X(n)

The symbolic name of the event parameter.

PARAM-VALUE Any COBOL data type

Destination item to receive the event parameter's value.

RESULT-CODE Signed numeric value

Receives the result-code for the operation. This will be 0 to indicate success or a negative value to indicate failure. (Microsoft defines many standard "result codes" in their documentation. Note that these are usually in hexadecimal notation.)

Comments

C$GETEVENTPARAM converts the named event parameter's value to COBOL-type data in the destination item. Using this routine instead of C$GETEVENTDATA will make your code more readable. The object code will be a little larger and calls to this routine will take a little longer than calls to C$GETEVENTDATA. However, these differences will probably be unnoticeable and the benefits of readable code outweigh the performance and size considerations.

You are responsible for specifying a compatible types. For example, if the event parameter is a character string and you specify a numeric item to receive its value, C$GETEVENTPARAM will try to read a number from the string and move it to your numeric item. This is not a programming error and neither the compiler nor runtime will warn you about it.

Example

Suppose you have displayed an ActiveX control that triggers an event called AxEventOne which has three parameters. Then suppose that you only need to know the value of PARAM-2. Since PARAM-2 is the second parameter, to get its value you would have to pass a "dummy" first parameter using C$GETEVENTDATA. For example:

```
CALL "C$GETEVENTDATA" USING EVENT-CONTROL-HANDLE, 0, PARAM-2.
```

However, if you determined that the name of PARAM-2 in the ActiveX control was"Param2". You could then use C$GETEVENTPARAM to accomplish this task in a more elegant and readable way. For example:

```
CALL "C$GETEVENTPARAM"
    USING EVENT-CONTROL-HANDLE, "Param2", PARAM-2.
```

For more examples of how to get event parameters for ActiveX events, refer to section 4.4 in *A Guide to Interoperating with ACUCOBOL-GT.*

C$GETLASTFILEOP

C$GETLASTFILEOP retrieves information about the last file operation performed.

Usage

```
CALL "C$GETLASTFILEOP"
    USING OPERATION, ADDR, X-ADDR
```

Parameters

OPERATION PIC X(n)

Receives the name of the last file operation performed in the current thread. If the current thread has not performed any file operations, then OPERATION is set to spaces. The possible values are:

Close	ReadPreviousLock
Commit	ReadPreviousNoLock

Delete	Rewrite
DeleteFile	Rollback
Open	Start
ReadLock	StartTransaction
ReadNextLock	Unlock
ReadNextNoLock	UnlockAll
ReadNoLock	Write

If the current thread has not performed any file operations, then OPERATION is set to spaces.

ADDR Numeric (optional)

Receives the virtual address of the instruction that performed the last file operation in the current thread. If no operation has been performed, zero is returned.

X-ADDR PIC X(n) (optional)

Receives a 6-character string that contains the hexadecimal equivalent of the value returned in ADDR.

Description

C$GETLASTFILEOP should be called from Declaratives when an unexpected I/O error has occurred. The return values can be useful in some debugging or customer support scenarios.

The ADDR parameter is provided primarily for compatibility with other COBOL systems. You would normally convert the value returned by ADDR to hexadecimal and look up the result in the program's compiler listing. As a convenience, the X-ADDR parameter performs the hexadecimal conversion for you. You can omit the ADDR parameter by specifying "NULL" or "OMITTED" in the CALL statement, for example:

```
77  FILE-OP     PIC X(20).
77  X-ADDR      PIC X(6).
CALL "C$GETLASTFILEOP" USING FILE-OP, NULL, X-ADDR
```

This routine always returns a value of “0” for ADDR and X-ADDR when called by a Version 5.1, or earlier, native-code object file. Non-native object files return correct values regardless of version.

C$GETNETEVENTDATA

When a .NET assembly generates an event, it makes information about the event available through event parameters. These parameters are stored in the assembly and can be retrieved by calling C$GETNETEVENTDATA from the assembly’s event procedure.

Usage

```
CALL "C$GETNETEVENTDATA"
   USING EVENT-CONTROL-HANDLE, DEST-ITEM-1, [DEST-ITEM-2, ...]
    GIVING RESULT-CODE
```

Parameters

EVENT-CONTROL-HANDLE USAGE HANDLE

Handle to the .NET assembly that generated the event.

DEST-ITEM-1 Any COBOL data type

The first destination data item.

DEST-ITEM-2, ... Any COBOL data type (optional)

Any number of destination items.

RESULT-CODE Signed numeric value

Receives the result-code for the operation. This will be “0” to indicate success or a negative value to indicate failure. (Microsoft defines many standard “result codes” in their documentation. Note that these are usually in hexadecimal notation.)

Comments

C$GETNETEVENTDATA converts the event parameters to COBOL-type data in the destination items.

You are responsible for specifying compatible types. For instance, if the event parameter is a character string and you specify a numeric item to receive its value, C$GETNETEVENTDATA will try to read a number from the string and move it to your numeric item. This is not a programming error and neither the compiler nor runtime will warn you about it.

Example

Suppose you have displayed a .NET assembly that triggers an event called NetEventOne that has three parameters. You would use the following COBOL syntax to get the event parameters into data items PARAM-1, PARAM-2 and PARAM-3:

```
WHEN NetEventOne
   CALL "C$GETNETEVENTDATA" USING EVENT-CONTROL-HANDLE,
      PARAM-1, PARAM-2, PARAM-3
```

C$GETPID

This routine returns the Process ID (PID) of the current process. This PID can be compared with the PID returned by the **C$LOCKPID** to determine if the current process is the one holding a file lock or record locked condition.

Usage

```
CALL "C$GETPID"
    GIVING PROCESS-ID.
```

Parameters

PROCESS-ID PIC 9(n)

This contains a numeric data item large enough to hold a PID. On most platforms, PIC 9(5) is sufficient. On 64-bit systems, PIC 9(7) is recommended.

Comments

When called from a non-UNIX or non-Windows runtime, C$GETPID returns a PROCESS-ID of "0". This behavior should not be used for system identification. Use ACCEPT ... FROM SYSTEM-INFO instead (see Chapter 6 of Book 3).

C$GETVARIANT

This routine retrieves data referenced by a Variant handle. Typically, this is data passed to an ACUCOBOL-GT program from another language such as Visual Basic. Note that there is also a **C$SETVARIANT** routine that sets data items referenced by Variant handles

C$GETVARIANT converts Variant type data to COBOL type data. Programs that call ACUCOBOL-GT using the ACUCOBOL-GT Automation Server or runtime DLL pass their parameters (by reference) as Variant types. The COBOL program receives handles to the Variant data. C$GETVARIANT gets the data associated with a particular handle and moves it to a COBOL data item. The data is automatically converted to the proper COBOL format.

Usage

```
CALL "C$GETVARIANT"
    USING H-VARIANT, DEST-ITEM
    GIVING RESULT-CODE
```

Parameters

H-VARIANT USAGE HANDLE

Handle to Variant type data. Data passed in from a program calling ACUCOBOL-GT using the ACUCOBOL-GT Automation Server or runtime DLL is in the form of handles to Variant type data.

DEST-ITEM Any COBOL data type

The destination data item.

RESULT-CODE Signed numeric value

Receives the result code for the operation. This will be 0 or a positive value to indicate success or a negative value to indicate failure.

Under Microsoft Windows this is a code of type HRESULT that can be looked up in Microsoft documentation to determine the reason for the failure or additional information about the success.

Comments

The Variant data associated with H-VARIANT is moved to DEST-ITEM using standard MOVE rules. Data items must be defined in the LINKAGE section.

C$GETVARIANT is not supported by the ACUCOBOL-GT Thin Client technology, and will only run with a standalone Windows runtime.

C$JAVA

This routine enables your ACUCOBOL-GT program to invoke a Java program. It causes the Java Virtual Machine (JVM) to be loaded (if it is not already) and the specified Java class to be loaded.

Using this routine, you can create a Java object, call the methods of a Java object, create and use Java arrays, and use Java logging.

COBOL/Java interoperability is described fully in Chapter 2 of *A Guide to Interoperating with ACUCOBOL-GT*. Section 2.3.1, “Calling the C$JAVA Routine,” provides detailed information and best practices for working with the C$JAVA routine.

Usage

```
CALL "C$JAVA"
    USING OP-CODE, CLASS-NAME, METHOD-NAME, METHOD-SIGNATURE,
        FIELD-INT, FIELD-RETURN
    GIVING STATUS-VAL
```

Note: To call Java via the C$JAVA routine, HP-UX users must relink the runtime so that it is statically linked to “libjvm.sl”. To do this, modify lib/Makefile (uncommenting and editing the lines regarding relinking for Java), then run make to relink the runtime.

Parameters

OP-CODE Numeric parameter

Specifies the operation to perform. These are defined in the description section below.

CLASS-NAME Alphanumeric data item or literal

Specifies the fully qualified class name, with package name if necessary, of the Java class to load.

METHOD-NAME Alphanumeric data item or literal

Specifies the name of the specific method in the Java class that you want to call.

METHOD-SIGNATURE Alphanumeric data item or literal

Specifies the method signature ID of the exact method to be called. You can get this value by running the Java utility “javap.exe”on the desired jar file or class. This utility comes with the Java JRE and automatically produces the exact signature of a given method. You can copy and paste this signature ID into your CALL “C$JAVA” statement. You can also determine this value manually. See section 2.3.1.1 in *A Guide to Interoperating with ACUCOBOL-GT*, for information determining method signatures including syntax examples.

FIELD-INT Numeric parameter

Specifies the input parameters that the method requires.

FIELD-RETURN Numeric parameter

Specifies the parameter to hold the Java return value from the method.

STATUS-VAL Numeric parameter

Specifies a status value be returned after the call is complete. Possible return values are:

```
CJAVA-SUCCESS           VALUE 0.
CJAVA-INVALIDARG        VALUE -1.
CJAVA-INVALIDSIGNATURE  VALUE -2.
CJAVA-CLASSNOTFOUND     VALUE -3.
CJAVA-METHODNOTFOUND    VALUE -4.
CJAVA-INVALIDPARAMTYPE  VALUE -5.
CJAVA-JAVALIBNOTLOADED  VALUE -6.
CJAVA-MEMNOTALLOC       VALUE -7.
CJAVA-INVALIDOPCODE     VALUE -8.
CJAVA-NOMEMORY          VALUE -9.
CJAVA-METHODFAILED      VALUE -10.
CJAVA-MISSINGPARAM     VALUE -11.
CJAVA-INVALIDINDEX     VALUE -12.
CJAVA-EXCEPTIONOCCURRED  VALUE -13.
```

CJAVA-MISSINGPARAM is returned if a required value is not passed into a Java method. For example, say a method required five parameters, but only four were passed in.

CJAVA-INVALIDINDEX is returned if the region specified is not in the array or the region is larger or goes past the end of the array.

CJAVA-EXCEPTIONOCCURRED is returned when a Java exception was thrown during the call to C$JAVA. Use the CJAVA-GETEXCEPTIONOBJECT op-code to obtain the object of the Java exception.

Description

In all of the op-codes below, you can pass either the constant name or numeric value shown in parentheses. A “java.def” file is located in the sample/def directory where you installed ACUCOBOL-GT. It contains the op-codes for C$JAVA, as well as array types and return values. You can

include the file “java.def” in your COBOL program using the following statement if the “java.def” file is located in the same directory as the “.cbl” file:

```
working-storage section.
COPY "java.def".
```

CJAVA-NEW (op-code 1)

Create a new Java object on the local Java Virtual Machine (JVM). You must pass a fully qualified package/classname. Use the GIVING statement to return the object handle. For example:

```
CALL "C$JAVA" USING CJAVA-NEW, "MyJavaClass", GIVING
OBJECT-HANDLE.
```

CJAVA-DELETE (op-code 2)

Delete an existing Java object.

CJAVA-CREATE (op-code 3)

Create a new Java object. This is the same as using the CJAVA-NEW op-code. See op-code 1 for more details.

CJAVA-DESTROY (op-code 4)

Destroy a Java object. You must pass a valid object handle:

```
CALL "C$JAVA" USING CJAVA-DESTROY, OBJECT-HANDLE
   GIVING STATUS-VAL.
```

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.3, “Creating and Using Java Objects in COBOL,” for additional details.

CJAVA-INVOKE (op-code 7)

Same as CJAVA-CALLSTATIC. See op-code 10 for details.

CJAVA-CALL (op-code 8)

Call a virtual Java method. For example:

```
CALL "C$JAVA" USING CJAVA-CALL, OBJECT-HANDLE,
"CobolCallingJavaStringV", "(X)X", FIELD-STRING,
FIELD-STRINGRET GIVING STATUS-VAL.
```

Note that the object handle and method name are passed, but the class name is not required. Because an object handle has already been created, C$JAVA knows what type the handle is.

Methods can be called as static methods, virtual methods, or non-virtual methods by using op-codes 8 -10. If an op-code is not used, the default runtime behavior is to attempt to call the method statically, and then as a virtual method by attempting to create an object using a default constructor. A non-virtual method is called on the specific object that is being used. A virtual method can be called on a method that is inherited from one of the object's superclasses.

CJAVA-CALLNONVIRTUAL (OP-CODE 9)

Call a non-virtual Java method on the specific object that is being used. For example:

```
CALL "C$JAVA" USING CJAVA-CALLNONVIRTUAL, OBJECT-HANDLE,
"CobolCallingJavaStringV", "(X)X", FIELD-STRING,
FIELD-STRINGRET GIVING STATUS-VAL.
```

CJAVA-CALLSTATIC (op-code 10)

Call a static Java method. For example:

```
CALL "C$JAVA" USING CJAVA-CALLSTATIC, "acuCobolGT/CAcuCobol",
   "CobolCallingJavaDouble", "(D)D",
   FIELD-DOUBLE, FIELD-DOUBLERET
   GIVING STATUS-VAL
```

Note that you can call static methods directly without having to create an object first.

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.3, subsection "Calling Methods on Java Objects," for additional details.

CJAVA-NEWARRAY (op-code 11)

Create a new Java array. This is the same as using the CJAVA-CREATEARRAY op-code. See op-code 18 for more details. See also *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, "Creating and Using Java Arrays in COBOL."

CJAVA-DESTROYARRAY (op-code 12)

Destroy a Java array. Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, "Creating and Using Java Arrays in COBOL," for additional details.

CJAVA-SETARRAYELEMENT (op-code 13)

Set Java array elements. Pass in an array handle, the position in the array to set, and the value to set. In the following example, the first element of an array is set with the first value from an integer table that is USAGE IS SIGNED-INT OCCURS 10.

```
CALL "C$JAVA" USING CJAVA-SETARRAYELEMENT, ARRAY-HANDLE, 1,
INT-TABLE(1), GIVING STATUS-VAL.
```

Note: Although Java array elements start at "0", COBOL arrays start at "1". Because C$JAVA is a COBOL statement, the SETARRAYELEMENT and GETARRAYELEMENT op-codes, and all array access from C$JAVA, use "1" as the beginning of the array.

CJAVA-GETARRAYELEMENT (op-code 14)

Get a Java array element. This call requires an array handle, the position in the array to get, and the variable into which the array value will be placed. Here is an example:

```
CALL "C$JAVA" USING CJAVA-GETARRAYELEMENT,
   ARRAY-HANDLE, 5, INT-TABLE(1),
   GIVING STATUS-VAL.
```

In this case, we are getting element 5 from the array and placing it in the first element of an integer table.

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, “Creating and Using Java Arrays in COBOL,” for additional details.

CJAVA-CLEARARRAY (op-code 15)

Clear a Java array. Pass in the array handle of the array to be cleared. Here is an example:

```
CALL "C$JAVA" USING CJAVA-CLEARARRAY, ARRAY-HANDLE
   GIVING STATUS-VAL.
```

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, “Creating and Using Java Arrays in COBOL,” for additional details.

CJAVA-CONVERTARRAYTOTABLE (op-code 16)

Convert a Java array to a COBOL table. Although ACUCOBOL-GT normally does this automatically, this op-code gives the COBOL programmer more precise control over the conversion process. The call takes the array handle, the number of elements to convert, the starting element position in the array, and the COBOL table variable in which to place the converted array.

Here is an example of an array of Java ints being converted to a USAGE SIGNED-INT OCCURS 10 table:

```
CALL "C$JAVA" USING CJAVA-CONVERTARRAYTOTABLE,
   ARRAY_HANDLE, 10, 0, INT-TABLE(1)
   GIVING STATUS-VAL.
```

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, “Creating and Using Java Arrays in COBOL,” for additional details.

CJAVA-LOGMESSAGE (op-code 17)

This operation places an entry in the Java log. Java messages can be logged from a COBOL program using the CJAVA-LOGMESSAGE as follows:

```
CALL "C$JAVA" USING CJAVA-LOGMESSAGE, "Message to log".
```

The advantage of using the Java log is that it is thread-safe, and all of the messages from a given thread of execution are written to the same log whether that thread is executing COBOL or Java. Also, logs in Java are

highly configurable. Note that the log output above is formatted to report date, time, class, method, and log level before the message. You can configure logging by modifying the "logging.properties" file found in the runtime directory.

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.5, "Using Java Logging from COBOL," for additional details.

CJAVA-CREATEARRAY (op-code 18)

Create a new Java array. Following are the array types supported by C$JAVA:

```
CJAVA-OBJECTARRAY     VALUE 300.
CJAVA-BOOLEANARRAY    VALUE 301.
CJAVA-BYTEARRAY       VALUE 302.
CJAVA-CHARARRAY       VALUE 303.
CJAVA-SHORTARRAY      VALUE 304.
CJAVA-INTARRAY        VALUE 305.
CJAVA-LONGARRAY       VALUE 306.
CJAVA-FLOATARRAY      VALUE 307.
CJAVA-DOUBLEARRAY     VALUE 308.
CJAVA-STRINGARRAY     VALUE 309.
```

To create an array of primitive types, pass the type of array, the size of the array, and return the array handle through the GIVING statement. For example:

```
CALL "C$JAVA" USING CJAVA-CREATEARRAY,
   CJAVA-INTARRAY, ARRAY-SIZE
   GIVING ARRAY-HANDLE.
```

To create an object array:

```
CALL "C$JAVA" USING CJAVA-CREATEARRAY,
   CJAVA-OBJECTARRAY, 10
   GIVING ARRAY-HANDLE.
```

In this case, the array consists of an array of object handles.

To create a string array:

Even though strings in Java are objects, they are treated separately for the convenience of using them with PIC X tables. Here is an example of creating a string array:

```
CALL "C$JAVA" USING CJAVA-CREATEARRAY,
   CJAVA-STRINGARRAY, 10
   GIVING ARRAY-HANDLE.
```

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, "Creating and Using Java Arrays in COBOL," for additional details.

CJAVA-CONVERTTABLETOARRAY (op-code 19)

Convert a COBOL table to a Java array. Although ACUCOBOL-GT normally does this automatically, this op-code gives the COBOL programmer more precise control over the conversion process. The call requires the COBOL table from which the values will be taken, the number of elements, the position of the first element, and the handle of the destination array.

Here is an example of call that converts a table to an array:

```
CALL "C$JAVA" USING CJAVA-CONVERTTABLETOARRAY,
   INT-TABLE(1), 10, 0, ARRAY-HANDLE,
   GIVING STATUS-VAL.
```

Refer to *A Guide to Interoperating with ACUCOBOL-GT*, Chapter 2, section 2.3.1.4, "Creating and Using Java Arrays in COBOL," for additional details.

CJAVA-LOADCLASS (op-code 20)

Manually load a Java class prior to use. Normally, java doesn't require classes to be manually loaded. When you create new object, java typically searches the classpath and loads the class.

CJAVA-DBCONNECT (OP-CODE 21)

Connect to a Java Database Connectivity (JDBC) data source.

Because the "CVM.jar" package that comes with ACUCOBOL-GT contains a Java class for connecting to JDBC data sources and querying those data sources for ResultSet objects, you do not have to use the C$JAVA routine to

do this. However, using C$JAVA is somewhat more efficient because the Connection and ResultSet handles do not have to be created prior to being used.

To use op-codes 21 and 22, you must include the “java.def” file in the working storage section of your COBOL program. Be sure that “java.def” file is located in the same directory as the “.cbl” file.

Refer to the Interoperability Guide section 2.3.1.6 for more information on achieving connectivity with a JDBC data source.

CJAVA-DBQUERY (OP-CODE 22)

Query the JDBC ResultSet. See op-code 21.

CJAVA-NEWREMOTEOBJECT (OP-CODE 23)

Connect to a Remote Object Invocation (RMI) server, and create an instance of a remote object on that server. This operation takes three parameters: the host name, the server name, and the port number. Here is an example of the call:

```
CALL "C$JAVA" USING CJAVA-NEWREMOTEOBJECT, "localhost",
"RemoteInterfaceServer", PORT-NUMBER GIVING REMOTE-OBJ.

CALL "C$JAVA" USING CJAVA-CALL, REMOTE-OBJ, "acuUtilities/
RemoteInterfaceServer", "RemoteMethod", "()X", FIELD-STRINGRET
GIVING STATUS-VAL.

CALL "C$JAVA" USING CJAVA-DELETE, REMOTE-OBJ GIVING
STATUS-VAL.
```

CJAVA-STARTREMOTESERVER (OP-CODE 24)

Start the RMI server. For example:

```
CALL "C$JAVA" USING CJAVA-NEW, "acuUtilities/AcuRMIServer",
"()V" GIVING REMOTE-SERVER.
CALL "C$JAVA" USING CJAVA-STARTREMOTESERVER, REMOTE-SERVER,
"RemoteInterfaceServer", PORT-NUMBER GIVING STATUS-VAL.
```

CJAVA-SETARRAYREGION (OP-CODE 25)

Take a Java array object and copy the elements from a COBOL table data item into a specified range.

CJAVA-GETARRAYREGION (OP-CODE 26)

Take a Java array object, get the specified range of elements, and copy them into a COBOL table data item.

CJAVA-GETEXCEPTIONOBJECT (OP-CODE 27)

Return the exception object of the last Java exception thrown. Once the exception object is returned, you can call any of the methods on the exception object that are documented in the Java documentation.

CJAVA-SETSYSTEMPROPERTY (OP-CODE 28)

Change a system property. Useful for setting CLASSPATH and LIBRARYPATH environment variables. For example:

```
CALL "C$JAVA" USING CJAVA-SETSYSTEMPROPERTY,
"java.class.path",
"d:\MyClasses.jar;d:\otherDir" GIVING STATUS-VAL.

CALL "C$JAVA" USING CJAVA-SETSYSTEMPROPERTY,
"java.library.path",
"C:\Acucorp\Acucbl800\AcuGT\bin;d:\otherDir" GIVING
STATUS-VAL.
```

CJAVA-CALLJAVAMAIN (OP-CODE 29)

Call a Java main method. For example:

```
CALL "C$JAVA" USING CJAVA-CALLJAVAMAIN, "CobolCallingJava", "StrParam1",
    "StrParam2", "StrParam3", "StrParam4" GIVING STATUS-VAL.
```

This example calls a Java main method with the following signature:

```
public static void main( String[] args );
```

The main method signature can include optional exceptions that have been specified by the Java programmer. The first parameter is the op-code, the second parameter is the name of the class containing the main method. This directory or JAR file containing this class must be placed on the CLASSPATH environment variable. The number of string parameters can be zero or as many as is possible to pass using the CALL statement. The COBOL calling Java samples in the sample/java directory have been updated to include a sample of such a call.

C$JUSTIFY

C$JUSTIFY performs left or right justification of data and centering of data.

Usage

```
CALL "C$JUSTIFY"
    USING DATA-ITEM, JUSTIFY-TYPE
```

Parameters

DATA-ITEM Any data item

This data item contains the data to be justified.

JUSTIFY-TYPE PIC X

This optional parameter contains one of three literal values:

L indicates left justification

R indicates right justification

C indicates centering

If this parameter is omitted, then “R” is implied.

Description

This routine removes all leading and trailing spaces from DATA-ITEM and justifies the remaining data as indicated by JUSTIFY-TYPE. The resulting string is returned in DATA-ITEM. If centering is chosen, there will be one more space on the right than on the left if an odd number of spaces is used.

C$KEYMAP

The C$KEYMAP routine maintains a stack of keyboard configurations. This routine is used to save the current keyboard configuration just before you change it. You can then later restore the original configuration by another call to this routine. The ACUCOBOL-GT debugger uses this routine before reprogramming the keyboard for its use.

Usage

```
CALL "C$KEYMAP"
    USING KEYSTROKE-SETTING, STATUS-VAL
```

Parameters

KEYSTROKE-SETTING Numeric parameter

Dictates whether to save (“1”) or restore (“0”) the keyboard settings.

STATUS-VAL COMP-1 (optional)

Returns “1” if successful, “0” otherwise.

Comments

If KEYSTROKE-SETTING is set to “1”, then the current KEYSTROKE settings are pushed onto a stack. If it is “0”, then the KEYSTROKE settings are restored from the stack and the stack is popped. For a description of KEYSTROKE settings, see the *ACUCOBOL-GT User’s Guide*, **section 4.3.2.2, “The KEYSTROKE variable.”**

The keyboard stack has space for 10 entries. One of these should be reserved for use by the debugger. If the stack overflows, or inadequate dynamic memory is available to save the keyboard configuration, then this routine does nothing.

This routine may optionally be passed a second parameter. This must be a COMP-1 field. It is set to "1" if the routine succeeds. It is set to "0" if the stack is full or inadequate memory is available.

The NOTEPAD.CBL sample program provided with ACUCOBOL-GT contains an example of the use of this routine.

C$KEYPROGRESS

The C$KEYPROGRESS routine is used in a USE FOR REPORTING declarative procedure to obtain information about the progress of the system when it is adding keys to a file open for BULK-ADDITION.

Usage

```
CALL "C$KEYPROGRESS"
    USING KEYPROGRESS-DATA
```

Parameters

KEYPROGRESS-DATA Group item as follows:

```
01  KEYPROGRESS-DATA, SYNC.
    03  KEYPROG-CUR-KEY    PIC XX COMP-N.
    03  KEYPROG-NUM-KEYS   PIC XX COMP-N.
    03  KEYPROG-CUR-REC    PIC X(4) COMP-N.
    03  KEYPROG-NUM-RECS   PIC X(4) COMP-N.
```

This item holds the results of the C$KEYPROGRESS routine call. A copy of this data item can be found in the COPY library "keyprog.def".

Description

When you call C$KEYPROGRESS from inside a USE FOR REPORTING declarative procedure, the KEYPROGRESS-DATA group item is filled in with information about the progress the runtime is making in adding keys to a file that has been opened for BULK-ADDITION. Calling C$KEYPROGRESS at any other time has undefined effects.

The individual fields of the group item hold the following information:

KEYPROG-CUR-KEY -- this is the current key being worked on by Vision. The primary key is key "1", the first alternate is key "2", and so on.

KEYPROG-NUM-KEYS -- this is the total number of keys in the file.

KEYPROG-CUR-REC -- this is the number of the last record written for the current key, ranging from 1 to the total number of records to write.

KEYPROG-NUM-RECS -- this is the total number of records to be keyed.

For more information about how and when to use C$KEYPROGRESS, see **Section 6.1.6.3** in the *ACUCOBOL-GT User's Guide*.

C$LIST-DIRECTORY

The C$LIST-DIRECTORY routine lists the contents of a selected directory. Each operating system has a unique method for performing this task. C$LIST-DIRECTORY provides a single method that will work for all operating systems.

Usage

```
CALL "C$LIST-DIRECTORY"
    USING OP-CODE, parameters
```

Parameters

OP-CODE PIC 99 COMP-X

Indicates which C$LIST-DIRECTORY operation to perform. The operations are described below.

Parameters vary depending on the op-code chosen.

Provides information and hold results for the operations specified. These parameters are described below.

Description

C$LIST-DIRECTORY allows you to get the names of files residing in a given directory. It accomplishes this through three distinct operations. The first operation opens the specified directory. The second operation returns the filenames in the list, one-at-a-time. The third operation closes the directory and deallocates all memory used by the routine. C$LIST-DIRECTORY has the following operation codes (defined in "acucobol.def"):

LISTDIR-OPEN (VALUE 1) Opens the specified directory. It has two parameters:

Directoryname **PIC X(n)**

Contains the name of the directory to open. This directory must exist, and you must have permissions to read the directory. You may use remote name syntax if AcuServer is installed on the remote machine. The "@[DISPLAY]:" for Thin Client support may be used. For example:

```
 C$LIST-DIRECTORY using listdir-open,
"@[DISPLAY]:C:\path", pattern
```

See the section **Thin Client and Windows special directories** for more information.

Pattern **PIC X(n)**

Specifies the type of filename for which to search. This routine supports "wildcards," meaning that the character "*" will match any number of characters, and the character "?" will match any single character. For example, you can search by file suffix (*.def) or by a common part of a file name (acu*).

Note: On VMS platforms, when searching for all files in a directory, you must specify "*.*" instead of "*".

If the call to LISTDIR-OPEN is successful, RETURN-CODE contains a handle to the list. The value in RETURN-CODE should be moved to a data item that is USAGE HANDLE. That data item should be passed as the directory handle to the other C$LIST-DIRECTORY operations. If the call to LISTDIR-OPEN fails (if the directory does not exist, contains no files, or you do not have permission to read the directory), RETURN-CODE is set to a NULL handle.

LISTDIR-NEXT (VALUE 2) Reads each filename from the open directory. It has two parameters:

Handle **USAGE HANDLE**

The handle returned in the LISTDIR-OPEN operation.

Filename **PIC X(n)**

The location of the next filename to be returned. If the directory listing is finished, it is filled with spaces.

The call to LISTDIR-NEXT can include an additional argument, LISTDIR-FILE-INFORMATION (defined in "acucobol.def"), which receives information about the returned file name. This is an optional group item which returns information about the following data items:

LISTDIR-FILE-TYPE	The file type can be one of the following: B = block device C = character device D = directory F = regular file P = pipe (FIFO) S = socket U = unknown
LISTDIR-FILE-CREATION-TIME	The creation time is the date (and time) that the file was originally created.
LISTDIR-FILE-LAST-ACCESS-TIME	The last access time is the date (and time) that the file was last accessed by some application (usually when the file was queried in some way).

LISTDIR-FILE-LAST-MODIFICATION-TIME	The last modification time is the date (and time) the file was last written to.
LISTDIR-FILE-SIZE	The size of the file is given in bytes.

Note: Because the supported file types vary by operating system, These data items have slightly different meanings depending on your operating system. Even on operating systems that support these values, some file systems may not. Some versions of the UNIX® operating system may change these values when permissions are changed. Refer to your operating system documentation for specific definitions.

LISTDIR-CLOSE (VALUE 3) Releases the resources used by the other operations. It must be called to avoid memory leaks. It has one parameter, *handle*, which is the same data item used by the LISTDIR-NEXT operation.

Handle **USAGE HANDLE**

The handle returned in the LISTDIR-OPEN operation.

Example

The following example lists the contents of a directory with repeated calls C$LIST-DIRECTORY:

```
WORKING-STORAGE SECTION.
copy "def/acucobol.def".
01  pattern       pic x(5) value "*.vbs".
01  directory     pic x(20) value "/virusscan".
01  filename      pic x(128).
01  mydir         usage handle.
PROCEDURE DIVISION.
MAIN.
* CALL LISTDIR-OPEN to get a directory handle.
    call "C$LIST-DIRECTORY"
       using listdir-open, directory, pattern.
    move return-code to mydir.
    if mydir = 0
       stop run
    end-if.
* CALL LISTDIR-NEXT to get the names of the files.
* Repeat this operation until a filename containing only
```

```
* spaces is returned.  The filenames are not necessarily
* returned in any particular order.  Filenames may be
* sorted on some machines and not on others.
    perform with test after until filename = spaces
       call "C$LIST-DIRECTORY"
          using listdir-next, mydir, filename
    end-perform.
* CALL LISTDIR-CLOSE to close the directory and deallocate
* memory. Omitting this call will result in memory leaks.
    call "C$LIST-DIRECTORY" using listdir-close, mydir.
    stop run.
```

C$LOCALPRINT

The C$LOCALPRINT routine provides access to the ACUCOBOL-GT window manager's local print mechanism. This allows the program to write to a printer or other device attached directly to the user's terminal. One or two USING parameters should be provided.

Usage

```
CALL "C$LOCALPRINT"
    USING SOURCE-DATA, LINE-SPACE
```

Parameters

SOURCE-DATA PIC X(n)

This contains the data that will be sent to the local device. Any trailing spaces will be removed before the data is sent.

LINE-SPACE Numeric parameter (optional)

The second parameter, if specified, determines the line spacing provided by the routine. Its value is interpreted as follows:

0 No additional characters are sent after the data

1 A carriage return is sent after the data

2 A carriage return and a line feed are sent after the data

3 A carriage return and a form feed are sent after the data

If the second parameter is not supplied, then it is treated as if it were a “2”.

You may add either or both of the following values to the value contained in the second parameter:

+10 Any line-advancing control sequences are sent before the data instead of after.

+20 Inhibits the sending of the disable-printer terminal sequence after sending the data.

For example, a second parameter value of “12” causes the printer to be advanced one line prior to sending the data (since a value of “2” causes the printer to advance one line after sending the data).

A second parameter value of “8” (with a dummy first parameter) causes the disable-print sequence to be sent to the terminal on its own.

Description

When this routine executes, it performs the following steps in this order:

1. The terminal’s Enable Print command is sent. This causes subsequent data sent to the terminal to be also sent to the attached device.
2. The data specified in the first USING parameter (less trailing spaces) is sent to the terminal. No interpretation or modification is done to this data.
3. The appropriate line spacing characters specified by the second USING parameter are sent to the terminal.
4. The terminal’s Disable Print command is sent. This turns off the pass-through mode.
5. The current cursor location is set to the home position of the current window. The window manager is instructed to define the current screen state as “undefined”. This causes the window manager to

behave correctly if the terminal's pass-through mode is destructive to the screen (e.g., characters appear on the screen as well as being sent to the attached device).

Note: Some terminals handle pass-through printing in an invisible fashion (the characters go to the printer, but do not show up on the screen), while others echo the characters onto the screen too. For maximum portability, you should assume that a call to C$LOCALPRINT will cause garbage to appear on the screen and should arrange to repaint the screen when you are finished with the local device.

The routine assumes that the attached terminal is able to do pass-through printing. You may use the ACCEPT FROM TERMINAL-INFO verb to test for this ability beforehand if you desire. You need to ensure that any communication restrictions imposed by the terminal are followed. For example, some terminals require that the attached device run at the same baud rate as the terminal.

With programs that use the Window's console (DOS-box) runtime, the C$LOCALPRINT routine simulates local printing via the following procedure. When the routine is called for the first time, the runtime system opens a file to which it writes the passed data. The name of the file is determined by the setting of the environment variable LPRINTER. Usually this will be set to the name of a device such as PRN, LPT1, and so forth. If this variable is not set, then the default name PRN is used. Serial ports are *not* supported.

Once this file is open, it is not closed by the runtime system. If you need to close it, you can call C$LOCALPRINT with a dummy first parameter and a second parameter of "9". Non-Windows console runtimes ignore this form of the call. Also note that no error checking is possible. If the runtime cannot open the LPRINTER file, it prints an error message and stops.

C$LOCKPID

This routine returns the Process ID (PID) of the process holding the lock responsible for the previous file lock or record locked condition encountered. This library routine works only with the Vision file system and the UNIX platform.

Usage

```
CALL "C$LOCKPID"
    GIVING PROCESS-ID.
```

Parameter

PROCESS-ID PIC 9(n)

This contains a numeric data item large enough to hold a PID. On most platforms, PIC 9(5) is sufficient. On 64-bit systems, PIC 9(7) is recommended.

Comments

C$LOCKPID returns a PROCESS-ID of "0" if you have not yet encountered a locked file or record, if the PID is otherwise not found, or if you've used this routine with a non-UNIX runtime.

C$MAKEDIR

C$MAKEDIR creates a new directory. C$MAKEDIR can make a directory only one level lower than an existing directory and cannot create more than one level at a time.

Usage

```
CALL "C$MAKEDIR"
    USING DIR-NAME GIVING STATUS-CODE
```

Parameters

DIR-NAME PIC X(n)

Contains the name of the directory to be created. This should be either a full path name or a name relative to the current directory. You may use remote name syntax in combination with AcuServer to create a directory on a remote machine. The "@[DISPLAY]:" annotation for Thin Client support may also be specified. For example:

```
C$MAKEDIR "@[DISPLAY]:C:\path"
```

See the section **Thin Client and Windows special directories** for more information.

STATUS-CODE Numeric data item.

Receives the return status of the call to create a directory. A return status of zero indicates that the directory was successfully created; a status of one (“1”) indicates otherwise.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

C$MEMCPY (Dynamic Memory Routine)

Copies bytes between any two memory locations.

Usage

```
CALL "C$MEMCPY"
    USING, BY VALUE, DEST-PTR, SRC-PTR, NUM-BYTES
```

Parameters

DEST-PTR USAGE POINTER or USING BY REFERENCE

Contains the address of the first byte of the destination.

SRC-PTR USAGE POINTER or USING BY REFERENCE

Contains the address of the first byte of the source.

NUM-BYTES USAGE UNSIGNED-INT or an unsigned numeric literal

Indicates the number of bytes to copy.

Description

This routine copies NUM-BYTES bytes of memory from the address contained in SRC-PTR to the address contained in DEST-PTR. This routine is functionally similar to the **M$COPY (Dynamic Memory Routine)** routine except that parameters are passed by value instead of by reference. This routine can be used in cases where M$PUT and M$GET are not adequate. Note that this routine is relatively dangerous to use. It does not perform any error checking and can easily cause memory access violations if you pass it incorrect data. In other words, this routine is a very low-level routine and should be used cautiously.

You do not need to pass POINTER data items for SRC-PTR and DEST-PTR. If you prefer, either or both can be replaced by a data item passed BY REFERENCE. If you do this, then the address of the data item is passed to C$MEMCPY. For example, you can copy 10 bytes to DEST-ITEM from the memory address contained in SRC-PTR with:

```
CALL "C$MEMCPY"
   USING BY REFERENCE DEST-ITEM, BY VALUE SRC-PTR, 10
```

C$MYFILE

This routine returns the filename of the disk file containing the currently executing program. This is especially useful if the disk file is an object library.

Usage

```
CALL "C$MYFILE"
    USING PROGRAM-NAME
    GIVING CALL-STATUS
```

Parameters

PROGRAM-NAME PIC X(n)

Indicates the name of the disk file containing the currently executing program, if known. The runtime will use as much space for the name of the file as the COBOL program allows. This parameter will contain the filename just as the runtime received it. For example, if an object library is loaded as "-y ../ardir/myarlib.lib", and a program in "myarlib.lib" calls this routine, PROGRAM-NAME will have a value of "../ardir/myarlib.lib".

CALL-STATUS PIC S99.

This parameter receives one of the following values:

1 PROGRAM-NAME was filled successfully

-1 Program name unknown

C$NARG

This routine returns the number of parameters passed to the current program.

Usage

```
CALL "C$NARG"
    USING NUM-PARAM
```

Parameter

NUM-PARAM COMP-1

Description

This routine must be called with one USING parameter that must be a COMP-1 data item. This data item is filled in with the number of parameters. If the calling program is a subprogram, then this will be the number of USING items in the CALL statement that initiated the program. If the calling program is a main program, then this will be the number of CHAINING parameters passed from the **runcbl** command line or the CHAIN statement that initiated the program. C$NARG works only when the program is a called subroutine. It does not work with the "CALL RUN" form of the CALL verb.

C$OPENSAVEBOX

C$OPENSAVEBOX provides a facility for creating an Open or Save As dialog box. These dialogs allow the user to browse the system's file directories and select a file or folder.

This routine can be used by applications that are deployed in ***extend's*** Thin Client environment. In this scenario, C$OPENSAVEBOX allows the user to choose a directory or file on the *client* (Windows) machine.

Not all systems support C$OPENSAVEBOX. However, you can determine at runtime whether the host system supports it.

Usage

```
CALL "C$OPENSAVEBOX"
    USING OP-CODE, OPENSAVE-DATA
    GIVING OPENSAVE-STATUS
```

Parameters

OP-CODE Numeric value

Selects which C$OPENSAVEBOX function to perform. The values are described below.

OPENSAVE-DATA Group item as follows:

```
01  OPENSAVE-DATA.
    03  OPNSAV-FILENAME         PIC X(256).
    03  OPNSAV-FLAGS            PIC 9(4) COMP-X.
    03  OPNSAV-DEFAULT-EXT      PIC X(12).
    03  OPNSAV-TITLE            PIC X(80).
    03  OPNSAV-FILTERS          PIC X(512).
    03  OPNSAV-DEFAULT-FILTER   PIC 9(4) COMP-X.
    03  OPNSAV-DEFAULT-DIR      PIC X(128).
    03  OPNSAV-BASENAME         PIC X(128).
```

This item holds the results of a C$OPENSAVEBOX routine call. The values are described below.

OPENSAVE-STATUS Signed numeric data item

This item returns the status of the operation. A value of "1" indicates that the operation completed successfully. A zero or negative value indicates that the operation failed.

Description

C$OPENSAVEBOX performs a variety of operations depending on the operation passed. All of the data items and definitions required by this routine can be found in "opensave.def". The operations are as follows:

OPENSAVE-SUPPORTED (op-code 1)

This operation returns a value that indicates whether the host system supports C$OPENSAVEBOX. If the system supports it, OPENSAVE-STATUS is set to "1". Otherwise, it is set to OPNSAVERR-UNSUPPORTED (value "0"). The OPENSAVE-DATA parameter is not used with this op-code and should be omitted. (Note that Microsoft Windows hosts support C$OPENSAVEBOX.)

OPENSAVE-OPEN-BOX (op-code 2)

This operation initiates an Open File dialog with the user. The OPENSAVE-DATA structure initializes the dialog box and returns the results.

OPENSAVE-SAVE-BOX (op-code 3)

This operation initiates a Save As dialog with the user. The OPENSAVE-DATA structure initializes the dialog box and returns the results. On some systems, there is no difference between an Open and Save As dialog box. On other systems, there are some differences.

OPENSAVE-BROWSE-FOLDER (op-code 4)

This operation initiates a Browse for Folder dialog with the user. The OPENSAV-TITLE and OPENSAV-FILENAME fields in the OPENSAVE-DATA structure initialize the dialog box and return the result, respectively. When C$OPENSAVE BOX is called, the Browse for Folder dialog box displays the contents of OPENSAV-FILENAME as the root folder to be browsed. Only folders which are descendants of this folder are

shown in the dialog box. If OPENSAV-FILENAME is blank when the routine is called, the dialog shows all folders in the user's default working folder, and a number of other items, such as the Recycle Bin. You may select an empty folder, however, if the folder specified as a root does not exist or is inaccessible, the dialog shows all folders in the user's default working folder, just as though no root was specified. The "OK" button is disabled if any item other than a folder is selected.

OPENSAVE-DATA

You should use the INITIALIZE verb on OPENSAVE-DATA before you fill in the data fields. This ensures that you have set all the fields to the default values and protects you from possible future changes to the OPENSAVE-DATA structure.

The OPENSAVE-DATA item is fairly large (1120 bytes). You can conserve memory by using C$OPENSAVEBOX from a utility subprogram that you write. This subprogram would include OPENSAVE-DATA. After using the subprogram, you can free the memory with the CANCEL verb. In this way, you need to keep OPENSAVE-DATA in memory only while you are using it. Alternatively, you can use the M$ALLOC library routine to allocate memory to hold OPENSAVE-DATA, and then free that memory after you are done.

The fields contained in the OPENSAVE-DATA structure are used as follows:

OPNSAV-FILENAME -- On input to the routine, this item contains the default file name to use as the initial prompt. Set OPNSAV-FILENAME to spaces if there should be no default name. When the routine returns, this item contains the name of the file selected by the user. When used with the OPENSAVE-BROWSE-FOLDER operation, this item returns the file specifications of the selected folder, if any. If the user selects a folder which is not accessible, this item is blank.

OPNSAV-FLAGS -- This item is used to pass information to the OPENSAVE-OPEN-BOX, OPENSAVE-SAVE-BOX, and OPENSAVE-BROWSE-FOLDER operations to modify the behavior of the associated dialog boxes. Constants for these flags are defined in "opensave.def".

The following flags can be used with the OPENSAVE-OPEN-BOX and OPENSAVE-SAVE-BOX operations:

OPENSAVE-OVERWRITEPROMPT

This flag causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.

OPENSAVE-PATHMUSTEXIST

This flag specifies that the user can type only valid paths and file names. If this flag is used and the user types an invalid path and file name in the File Name entry field, the dialog box function displays a warning in a message box.

OPENSAVE-FILEMUSTEXIST

This flag specifies that the user can type only names of existing files in the File Name entry field. If this flag is specified and the user enters an invalid name, the dialog box procedure displays a warning in a message box. If this flag is specified, the OPENSAVE-PATHMUSTEXIST flag is also used.

OPENSAVE-CREATEPROMPT

If the user specifies a file that does not exist, this flag causes the dialog box to prompt the user for permission to create the file. If the user chooses to create the file, the dialog box closes and the function returns the specified name.

OPENSAVE-NOREADONLYRETURN

This flag specifies that the returned file does not have the Read Only check box selected and is not in a write-protected directory.

The following flags can be used with the OPENSAVE-BROWSE-FOLDER operation:

OPENSAVE-BROWSE-DONTGOBELOWDOMAIN

When this flag is set, network folders below the domain level in the dialog box's tree view control are not included.

OPENSAVE-BROWSE-RETURNFSANCESTORS

When this flag is set, only file system ancestors are returned. An ancestor is a subfolder that is beneath the root folder in the namespace hierarchy. If the user selects an ancestor of the root folder that is not part of the file system, the OK button is disabled.

OPENSAVE-BROWSE-EDITBOX

This flag includes an edit control in the browse dialog box allowing the user to type the name of an item.

OPENSAVE-BROWSE-NEWDIALOGSTYLE

This flag provides the user with a larger dialog box that can be resized. The dialog box has several new capabilities including: drag and drop capability within the dialog box, reordering, shortcut menus, new folders, delete, and other shortcut menu commands.

OPENSAVE-BROWSE-BROWSEINCLUDEURLS

When this flag is set, the OPENSAVE-BROWSE-NEWDIALOGSTYLE, OPENSAVE-BROWSE-EDITBOX, and OPENSAVE-BROWSE-BROWSEINCLUDEFILES flags must also be set. Otherwise, the browser dialog box rejects URLs. Even when these flags are set, the browse dialog box displays URLs only if the folder containing the selected item supports them.

Note: When the folder's "IShellFolder::GetAttributesOf" method is called, the folder must set the SFGAO_FOLDER flag. If this is not set, the browse dialog box does not display the URL.

OPENSAVE-BROWSE-UAHINT

When combined with OPENSAVE-BROWSE-NEWDIALOGSTYLE, this flag adds a usage hint to the dialog box in place of the edit box. OPENSAVE-BROWSE-EDITBOX overrides this flag.

OPENSAVE-BROWSE-NONEWFOLDERBUTTON

When this flag is set, the New Folder button in the browse dialog box is not included.

OPENSAVE-BROWSE-BROWSEFORCOMPUTER

When this flag is set, if the user selects anything other than a computer, the OK button is disabled.

OPENSAVE-BROWSE-BROWSEFORPRINTER

When this flag is set, if the user selects anything other than a printer, the OK button is disabled.

OPENSAVE-BROWSE-BROWSEINCLUDEFILES

When this flag is set, the browse dialog box displays files as well as folders.

OPNSAV-DEFAULT-EXT
OPNSAV-TITLE
OPNSAV-FILTERS
OPNSAV-DEFAULT-FILTER
OPNSAV-DEFAULT-DIR
OPNSAV-BASENAME

OPNSAV-DEFAULT-EXT -- This item holds the default file name *extension*. The extension is the string of characters that appear after the "." in the file name. The value of OPNSAV-DEFAULT-EXT is added to the file name typed by the user, if the user does not type an extension. The default extension should not include the period ".". Set this item to spaces to avoid having a default extension.

OPNSAV-TITLE -- This item holds the title of the dialog box. If it is set to spaces, a generic title is applied. The generic title is host-specific. When used with the OPENSAVE-BROWSE-FOLDER operation, the title of the Browse for Folder dialog is always "Browse for Folder". The OPENSAV-TITLE item is displayed inside the Browse for Folder dialog box below the title bar and above the tree view control.

OPNSAV-FILTERS -- The value of OPNSAV-FILTERS describes the set of filters that the dialog box uses to restrict the set of files shown to the user. Filters make it easier for a user to navigate through a large directory by limiting the files shown at once.

Each filter consists of a pair of descriptors. These descriptors are separated by a vertical bar character ("|"). The first descriptor in the pair is displayed in the file type selection box of the Open or Save As dialog box. In Windows,

it appears in the List of File Types drop-down box (see the illustration below). The second descriptor is the file name pattern that defines the filter. The file name pattern is formatted as "A [. B]" where "A" and "B" are optional text followed by an optional asterisk. An asterisk matches any sequence of characters excluding periods. This descriptor is what the system uses to look for matching files.

Here is a sample OPNSAV-FILTERS setting that contains two filters:

```
"COBOL source files (*.cbl)|*.cbl|All files (*.*)|*.*"
```

The first filter in the example shows only ".cbl" files to the user. The second filter shows all files. The user selects which filter to use based on the descriptions supplied.

Filters do not restrict the user from entering names that do not match the supplied pattern. Filters do not limit the user's choices, they only simplify the process of choosing.

Set OPNSAV-FILTERS to spaces if you don't want any filters.

Some systems do not support multiple filters. In this case, only the initial filter is used. See OPNSAV-DEFAULT-FILTER to determine how to select the initial filter.

OPNSAV-DEFAULT-FILTER -- This item is used in conjunction with OPNSAV-FILTERS. The value of OPNSAV-DEFAULT-FILTER determines which of the given filters to use as the initial filter. A value of "1" selects the first filter pair, "2" selects the second pair, and so on. A value of zero also selects the first pair. This setting is not used if no filters are defined.

OPNSAV-DEFAULT-DIR -- This item holds the default directory to use for the selected file. The dialog box initially displays the files found in this directory. If this item is set to spaces, the current directory is used. Note that the value of this item only defines the default directory. It does not prevent the user from selecting files in a different directory.

OPNSAV-BASENAME -- When the routine returns, this item contains the base file name of the file chosen by the user. This differs from the value of OPNSAV-FILENAME in that all directory information is removed, leaving only the file name.

Error Handling

C$OPENSAVEBOX returns a value of "1" when successful. Otherwise, it returns one of the following values (found in "opensave.def"):

OPNSAVERR-UNSUPPORTED	This error indicates that the C$OPENSAVEBOX routine is not supported by the current host system.
OPNSAVERR-CANCELLED	This error indicates that the user clicked the "Cancel" button or typed the Escape key while using the dialog box.
OPNSAVERR-NO-MEMORY	This error indicates that not enough memory could be allocated to load the dialog box.
OPNSAVERR-NAME-TOO-LARGE	This error indicates that the name entered by the user does not fit in OPNSAV-FILENAME.

Example

This example uses C$OPENSAVEBOX to prompt for a text file name, and uses M$ALLOC to dynamically allocate OPENSAVE-DATA, freeing it after it is no longer needed.

```
WORKING-STORAGE SECTION.
77  OPENSAVE-DATA-SIZE    PIC 9(4) BINARY.
77  OPENSAVE-DATA-ADDR    POINTER.
77  OPENSAVE-STATUS       PIC S99.
    88  OPENSAVE-OK       VALUE 1.
77  FILE-NAME             PIC X(256).

LINKAGE SECTION.
COPY "opensave.def".

PROCEDURE DIVISION.
MAIN-LOGIC.
   SET OPENSAVE-DATA-SIZE TO SIZE OF OPENSAVE-DATA.
   CALL "M$ALLOC"
      USING OPENSAVE-DATA-SIZE, OPENSAVE-DATA-ADDR.
   IF OPENSAVE-DATA-ADDR = NULL
      {error handling here}
   ELSE
```

```
        SET ADDRESS OF OPENSAVE-DATA
            TO OPENSAVE-DATA-ADDR
        INITIALIZE OPENSAVE-DATA
        MOVE
            "Text files (*.txt)|*.txt|All files (*.*)|*.*"
            TO OPNSAV-FILTERS
        MOVE "txt" TO OPNSAV-DEFAULT-EXT
        CALL "C$OPENSAVEBOX" USING OPENSAVE-SAVE-BOX,
                                   OPENSAVE-DATA
                             GIVING OPENSAVE-STATUS

        IF OPENSAVE-OK
            MOVE OPNSAV-FILENAME TO FILE-NAME
        END-IF
        CALL "M$FREE" USING OPENSAVE-DATA-ADDR
    END-IF.
```

C$PARAMSIZE

This routine returns the number of bytes actually passed by the caller for a particular parameter.

Usage

```
CALL "C$PARAMSIZE"
    USING PARAM-NUM,
    GIVING PARAM-SIZE
```

Parameters

PARAM-NUM Numeric parameter

This value is the ordinal position in the Procedure Division's USING phrase of the parameter whose size you want to know.

PARAM-SIZE Any numeric data item

This item receives the number of bytes in the data item actually passed by the caller.

Description

This routine returns the actual size (in bytes) of a data item passed to the current program by its caller. You pass the number (starting with "1") of the data item in the Procedure Division's USING phrase, and C$PARAMSIZE will return the size of the corresponding item that was actually passed. This can be useful for handling data items of unknown size.

For example, suppose that you wanted to write a routine that could convert any data item to upper-case, up to 10000 bytes in size. This routine could look like this:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  MAKE-UPPERCASE.

DATA DIVISION.
WORKING-STORAGE SECTION.
77  PARAM-SIZE    PIC 9(5).

LINKAGE SECTION.
77  PASSED-ITEM   PIC X(10000).

PROCEDURE DIVISION USING PASSED-ITEM.
MAIN-LOGIC.
   CALL "C$PARAMSIZE" USING 1, GIVING PARAM-SIZE
   INSPECT PASSED-ITEM( 1 : PARAM-SIZE )
      CONVERTING "abcdefghijklmnopqrstuvwxyz"
      TO "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  EXIT PROGRAM.
```

In this example, if you do not use C$PARAMSIZE, you have to pass a full 10000 bytes to this routine or you get a memory usage error. By using C$PARAMSIZE and reference modification, only the memory actually passed is referenced, and there is no error. C$PARAMSIZE works only when the program is a called subroutine. It does not work with the "CALL RUN" form of the CALL verb.

C$PARSEXFD

This routine is used to parse XFD files and retrieve information about them, giving you a way to map field description information to file record areas. Similar functionality allows the **alfred** utility to display data in a logical way (rather than displaying full records).

A detailed description of the use and structure of XFD files can be found in Book 1, Chapter 5, **section 5.3** of the *ACUCOBOL-GT User's Guide*.

Note: The utility **alfred** is distributed with a file called "parsexfd.ws", which describes how to use the **alfred** object library to parse XFD files. Although the C$PARSEXFD routine supersedes the abilities of that program, you may continue to use it. The "ParseXFD" COBOL program has been rewritten to use C$PARSEXFD, but the program interface has not changed.

Usage

```
CALL "C$PARSEXFD"
    USING OP-CODE, parameters
    GIVING return-value.
```

Parameters

OP-CODE Numeric value

The op-codes, which are defined in "parsexfd.def", select which C$PARSEXFD function to perform. This table shows which operation corresponds to each operation code.

Code	Operation
0	parse XFD file
1	retrieve key information
2	retrieve condition information
3	retrieve field information
4	test record conditions
9	release XFD file from memory

Detailed information about the operations is given in the description below.

parameters

Op-code parameters vary depending on the operation code chosen. They provide information and hold results for the operations specified. The parameters that apply to C$PARSEXFD op-codes are all defined in "parsexfd.def".

return-value Numeric data item

This item returns information relevant to the operation. The type of return value varies by op-code.

Description

C$PARSEXFD performs a variety of operations depending on the specified op-code. These operations are as follows:

PARSEXFD-PARSE (op-code 0)

This operation parses a specified XFD file. The syntax is:

```
CALL "C$PARSEXFD"
    USING PARSEXFD-PARSE, xfd-name, filename, flags,
          xfd-description.
```

PARSEXFD-PARSE takes the following parameters:

xfd-name PIC X(n)

Specifies the name of the XFD file to parse, with or without path information. If there is no path information, the configuration variables XFD_PREFIX or XFD_DIRECTORY are used to find the XFD file. You may omit the ".xfd" extension.

filename PIC X(n) or NULL

Specifies an indexed data file to be compared against the parsed XFD file. If the characteristics of the specified data file do not match the XFD, the parsed XFD is freed and the return-value is set to NULL. If this parameter is NULL or empty, the XFD file is not compared to any file.

flags Numeric parameter

Flags modify the type of information returned from other op-codes. This parameter can be "0" (if no flags are set), or the sum of any of the following values:

PARSEXFD-FLAG-INCLUDE-COMMENTS (value 1): This option causes comments to be included in the parsed XFD. The routine cannot, however, currently retrieve those comments.

PARSEXFD-FLAG-INCLUDE-999 (value 2): This option includes fields with a condition code of 999, which indicates group items and other fields not normally included with XFD files.

PARSEXFD-FLAG-EXCLUDE-ARRAYS (value 4): All table elements are normally appended with a value indicating their index. For example, for a field that occurs five times, the returned XFD includes five fields with _1, _2, _3, _4, and _5 appended to the field names. When this flag is set, such fields are returned with no suffix indicating their array index value. The information is still included, however, with the field group item (see below).

PARSEXFD-FLAG-DEEP-FIRST (value 8): This flag modifies the order in which fields that are sub-elements of a table are returned. For example:

```
07  my-array occurs 5 times.
   09  elem-1     pic x.
   09  elem-2     pic x.
   09  elem-3     pic x.
```

Normally this is returned as elem-1(1), elem-1(2), elem-1(3), elem-1(4), elem-1(5), elem-2(1), elem-2(2), elem-2(3), elem-2(4), elem-2(5), elem-3(1), elem-3(2), elem-3(3), elem-3(4), elem-3(5).

If PARSEXFD-FLAG-DEEP-FIRST is specified, the items are instead returned as elem-1(1), elem-2(1), elem-3(1), elem-1(2), elem-2(2), and so on. The same data is returned, but in a different order.

xfd-description Group item

This parameter is structured as follows:

```
01  PARSEXFD-DESCRIPTION.
    03  PARSEXFD-HEADER-LINE.
        05  PARSEXFD-VERSION           PIC X COMP-N.
        05  PARSEXFD-SELECT-NAME       PIC X(30).
        05  PARSEXFD-FILENAME          PIC X(30).
        05  PARSEXFD-FILETYPE          PIC X COMP-N.
            88 PARSEXFD-SEQUENTIAL-FILE    VALUE 4.
            88 PARSEXFD-RELATIVE-FILE      VALUE 8.
            88 PARSEXFD-INDEXED-FILE       VALUE 12.
        05  PARSEXFD-COBOL-TRIGGER     PIC X(100).
    03  PARSEXFD-RECORD-LINE.
        05  PARSEXFD-MAX-REC-SIZE      PIC X(4) COMP-N.
        05  PARSEXFD-MIN-REC-SIZE      PIC X(4) COMP-N.
        05  PARSEXFD-NUM-KEYS          PIC X COMP-N.
    03  PARSEXFD-COMPILE-LINE.
        05  PARSEXFD-SIGN-FLAG          PIC X(2) COMP-N.
            88  PARSEXFD-SIGN-ACU           VALUE 0.
            88  PARSEXFD-SIGN-IBM           VALUE 4.
            88  PARSEXFD-SIGN-MF            VALUE 8.
            88  PARSEXFD-SIGN-NCR           VALUE 20.
            88  PARSEXFD-SIGN-VAX           VALUE 36.
            88  PARSEXFD-SIGN-MBP           VALUE 72.
            88  PARSEXFD-SIGN-REA           VALUE 128.
        05  PARSEXFD-MAX-DIGITS         PIC X(2) COMP-N.
            88  PARSEXFD-18-DIGITS           VALUE 40.
            88  PARSEXFD-31-DIGITS           VALUE 68.
        05  PARSEXFD-SIGN-ATOI          PIC X(2) COMP-N.
            88  PARSEXFD-ATOI-PLUS           VALUE 11.
            88  PARSEXFD-ATOI-MINUS          VALUE 13.
        05  PARSEXFD-PGM-PERIOD         PIC X.
        05  PARSEXFD-PGM-COMMA          PIC X.
    03  PARSEXFD-CONDITION-LINE.
        05  PARSEXFD-NUMBER-CONDITIONS  PIC XX COMP-N.
    03  PARSEXFD-FIELDS-LINE.
        05  PARSEXFD-NUMBER-FIELDS      PIC X(4) COMP-N.
```

The values of the “xfd-description” parameter are defined as follows:

PARSEXFD-VERSION is the version number of this XFD file.

PARSEXFD-SELECT-NAME is the SELECT name of the file.

PARSEXFD-FILENAME is the name of the data file described in the XFD.

PARSEXFD-FILETYPE is the data file type. Valid values are 4 (sequential file), 8 (relative file), and 12 (indexed file).

PARSEXFD-COBOL-TRIGGER specifies the name of the COBOL program to be executed as a trigger.

PARSEXFD-MAX-REC-SIZE and **MIN-REC-SIZE** are the maximum and minimum size values for a record in this file.

PARSEXFD-NUM-KEYS gives the number of keys described in the XFD.

PARSEXFD-SIGN-FLAG indicates sign compatibility and is set by the “-Dc” compiler options.

PARSEXFD-MAX-DIGITS indicates the maximum numeric digits. The possible values are “18” and “31”. This is set by the “-Dd31” compiler option.

PARSEXFD-PGM-PERIOD indicates the decimal value of the character used as the program period.

PARSEXFD-PGM-COMMA indicates the decimal value of the character used as the program comma.

PARSEXFD-NUMBER-CONDITIONS gives the number of conditions described in the XFD file.

PARSEXFD-NUMBER-FIELDS is the number of fields available in the XFD.

return-value

For this op-code, the return value is the handle to the XFD. This handle must be used in future calls to C$PARSEXFD to get more information about the XFD, and to free the XFD when you are finished.

If the return-value is “0”, an error occurred. You can get information about errors by examining f-errno and f-int-errno, which are defined in the “filesys.def” COPY file.

PARSEXFD-GET-KEY-INFO (op-code 1)

This operation retrieves information about the specified (single) key. It uses the following syntax:

```
CALL "C$PARSEXFD"
    USING PARSEXFD-GET-KEY-INFO, xfd-handle, keynum,
          key-description
```

The operation takes the following parameters:

xfd-handle

A valid handle returned by C$PARSEXFD PARSEXFD-PARSE.

keynum Numeric parameter between 0 and PARSEXFD-NUM-KEYS

Used to specify which key to parse. Takes a value between 0 and PARSEXFD-NUM-KEYS. Both 0 and “PARSEXFD-NUM-KEYS - 1” are valid key numbers, but “PARSEXFD-NUM-KEYS” is not valid. In other words, key numbers are a zero-based array.

key-description Group item

This parameter is structured as follows:

```
01 PARSEXFD-KEY-DESCRIPTION.
   03 PARSEXFD-NUMBER-SEGMENTS        PIC X COMP-N.
   03 PARSEXFD-DUP-FLAG               PIC X COMP-N.
      88 PARSEXFD-ALLOW-DUPLICATES    VALUE 1 FALSE 0.
   03 PARSEXFD-SEGMENT-DESCRIPTION
        OCCURS MAX-SEGS TIMES
        INDEXED BY PARSEXFD-SEG-IDX.
```

```
   05 PARSEXFD-SEGMENT-LENGTH      PIC X COMP-N.
   05 PARSEXFD-SEGMENT-OFFSET      PIC X(4) COMP-N.
03 PARSEXFD-NUMBER-KEY-FIELDS      PIC X COMP-N.
03 PARSEXFD-KEY-FIELDS
     OCCURS MAXNUMKEYFIELDS TIMES
     INDEXED BY PARSEXFD-KEY-FIELD-IDX.
   05 PARSEXFD-KEY-FIELD-NUM       PIC XX COMP-N.
```

The values of the "key-description" parameter are defined as follows:

PARSEXFD-NUMBER-SEGMENTS specifies the number of segments in this key.

PARSEXFD-DUP-FLAG shows whether duplicates are allowed in this key. A value of "1" indicates that duplicates are allowed; a value of "0" indicates no duplicates.

PARSEXFD-SEGMENT-LENGTH and **SEGMENT-OFFSET** are the length and offset of each segment. The offset value is zero-based, so offset 0 is the beginning of the record. There is one SEGMENT-LENGTH and SEGMENT-OFFSET value for each segment.

PARSEXFD-NUMBER-KEY-FIELDS gives the number of fields that make up this key. This is always at least as large as the number of segments, but may be larger (if a segment holds multiple fields).

PARSEXFD-KEY-FIELDS is a table of key fields. This table has PARSEXFD-NUMBER-KEY-FIELDS valid elements.

PARSEXFD-KEY-FIELD-NUM is the field number of this key field. Get information about the key field by looking at this field number.

return-value 0 or 1

A return value of "1" indicates that the operation was successful; a "0" indicates failure. For this operation, a return code of "0" means that you have entered an invalid key number (for instance, specifying a key number of 3 for a file that only has two keys). Note, however, that if an invalid handle is specified, the results are undefined and may result in a memory violation.

PARSEXFD-GET-COND-INFO (op-code 2)

This operation retrieves information about conditions that use the WHEN directive within the XFD file. It uses the following syntax:

```
CALL "C$PARSEXFD"
    USING PARSEXFD-GET-COND-INFO, xfd-handle, cond-index,
          cond-description.
```

PARSEXFD-GET-COND-INFO takes the following parameters:

xfd-handle

A valid handle returned by C$PARSEXFD PARSEXFD-PARSE.

cond-index Numeric parameter

The condition index determines which condition to evaluate. It takes a value between 0 and PARSEXFD-NUMBER-CONDITIONS. Because conditions are a zero-based array, "0" and "PARSEXFD-NUMBER-CONDITIONS - 1" are valid values, but PARSEXFD-NUMBER-CONDITIONS is not.

cond-description Group item

The condition description holds information about what condition has been set (EQUAL TO, AND, OR), whether the condition has been met (is true or false), and how the condition is structured.

This parameter is structured as follows:

```
01  PARSEXFD-CONDITION-DESCRIPTION.
    03  PARSEXFD-CONDITION-TYPE         PIC X COMP-N.
        88  PARSEXFD-EQUAL-CONDITION    VALUE 1.
        88  PARSEXFD-AND-CONDITION      VALUE 2.
        88  PARSEXFD-OTHER-CONDITION    VALUE 3.
        88  PARSEXFD-GT-CONDITION       VALUE 4.
        88  PARSEXFD-GE-CONDITION       VALUE 5.
        88  PARSEXFD-LT-CONDITION       VALUE 6.
        88  PARSEXFD-LE-CONDITION       VALUE 7.
        88  PARSEXFD-NE-CONDITION       VALUE 8.
        88  PARSEXFD-OR-CONDITION       VALUE 9.
       88  PARSEXFD-COMPARISON-COND    VALUES 1, 4 THROUGH 8.
    03  PARSEXFD-CONDITION-FLAG         PIC X.
        88  PARSEXFD-TRUE-CONDITION     VALUE 'Y' FALSE 'N'.
```

```
03  PARSEXFD-COMPARISON-CONDITIONS.
    05  PARSEXFD-COMP-FIELDNUM      PIC XX COMP-N.
    05  PARSEXFD-COMP-FIELDNAME     PIC X(30).
    05  PARSEXFD-COMP-FIELD-VAL     PIC X(50).
03  PARSEXFD-OTHER-CONDITIONS
     REDEFINES PARSEXFD-COMPARISON-CONDITIONS.
    05  PARSEXFD-OTHER-FIELDNUM     PIC XX COMP-N.
    05  PARSEXFD-OTHER-FIELDNAME    PIC X(30).
03  PARSEXFD-AND-OR-CONDITIONS
     REDEFINES PARSEXFD-COMPARISON-CONDITIONS.
    05  PARSEXFD-CONDITION-1        PIC XX COMP-N.
    05  PARSEXFD-CONDITION-2        PIC XX COMP-N.
03  PARSEXFD-CONDITION-TABLENAME    PIC X(30).
```

The values of the “cond-description” parameter are defined as follows:

PARSEXFD-CONDITION-TYPE tells whether this is an EQUAL condition, AND condition, etc.

PARSEXFD-CONDITION-FLAG tells whether this condition is TRUE. This is only valid after PARSEXFD-TEST-CONDITONS (see below) has been called.

PARSEXFD-CONDITION-TABLENAME is the table name specified in the TABLENAME directive of the WHEN directive.

For EQUAL, GT (greater than), GE (greater than or equal to), LT (less than), LE (less than or equal to), and NE (not equal to) conditions, the following fields are valid:

PARSEXFD-COMP-FIELDNUM is the field number of the field whose value will be tested against the value of the condition.

PARSEXFD-COMP-FIELDNAME is the name of that field.

PARSEXFD-COMP-FIELD-VAL is the value to be tested. This is the value specified in the WHEN directive of the FD used to create this XFD.

For OTHER conditions, the following fields are valid:

PARSEXFD-OTHER-FIELDNUM is the field number of the field whose value will be different than all the other conditions which use this field.

PARSEXFD-OTHER-FIELDNAME is the name of that field.

For AND and OR conditions, the following fields are valid:

PARSEXFD-CONDITION-1 and **PARSEXFD-CONDITION-2** are the conditions tested to determine whether this condition is true. For AND, both conditions must be true. For OR, one or both conditions must be true.

PARSEXFD-GET-FIELD-INFO (op-code 3)

This operation retrieves information about the field. It uses the following syntax:

```
CALL "C$PARSEXFD"
    USING PARSEXFD-GET-FIELD-INFO, xfd-handle, fieldnum,
          field-description.
```

The operation takes the following parameters:

xfd-handle

A valid handle returned by C$PARSEXFD PARSEXFD-PARSE.

fieldnum Numeric parameter

Takes a value between 0 and PARSEXFD-NUMBER-FIELDS. Because fields are a zero-based array, “0” and “PARSEXFD-NUMBER-FIELDS - 1” are valid values, but PARSEXFD-NUMBER-FIELDS is not valid.

field-description Group item

This parameter is structured as follows:

```
01  PARSEXFD-FIELD-DESCRIPTION.
    03  PARSEXFD-FIELD-OFFSET          PIC X(4) COMP-N.
    03  PARSEXFD-FIELD-LENGTH          PIC X(4) COMP-N.
    03  PARSEXFD-FIELD-TYPE            PIC X COMP-N.
```

```
    88  PARSEXFD-SIGNED-FIELD   VALUES NumSignSep
                                       NumSigned
                                       NumSepLead
                                       NumLeading
                                       CompSigned
                                       PackedSigned
                                       BinarySigned
                                       NativeSigned.
    88  PARSEXFD-NUM-FIELD      VALUES NumEdited THRU
                                       NativeUnsigned.
    88  PARSEXFD-FLOAT-FIELD    VALUE  Flt.
    88  PARSEXFD-ASCII-FIELD    VALUES Alphanum THRU Group.
    88  PARSEXFD-NAT-FIELD      VALUES Nat-type THRU NatEdited.
    88  PARSEXFD-WIDE-FIELD     VALUES Wide-type THRU
                                       WideEdited.
03  PARSEXFD-FIELD-DIGITS            PIC X COMP-N.
03  PARSEXFD-FIELD-SCALE             SIGNED-SHORT.
03  PARSEXFD-FIELD-USER-TYPE         PIC XX COMP-N.
03  PARSEXFD-FIELD-CONDITION         PIC XX COMP-N.
03  PARSEXFD-FIELD-LEVEL             PIC X COMP-N.
03  PARSEXFD-FIELD-NAME              PIC X(30).
03  PARSEXFD-FIELD-FORMAT            PIC X(30).
03  PARSEXFD-FIELD-OCCURS-DEPTH      PIC X COMP-N.
03  PARSEXFD-FIELD-OCCURS-TABLE
        OCCURS MaxNumKeyFields TIMES
        INDEXED BY PARSEXFD-FIELD-OCCURS-LEVEL.
    05  PARSEXFD-FIELD-OCC-MAX-IDX   PIC XX COMP-N.
    05  PARSEXFD-FIELD-OCC-THIS-IDX  PIC XX COMP-N.
03  PARSEXFD-FIELD-IN-KEY-FLAG       PIC X.
    88  PARSEXFD-FIELD-IS-IN-KEY     VALUE 'Y' FALSE 'N'.
03  PARSEXFD-FIELD-SECONDARY-FLAG    PIC X.
    88  PARSEXFD-FIELD-IS-SECONDARY  VALUE 'Y' FALSE 'N'.
03  PARSEXFD-FIELD-HIDDEN-FLAG       PIC X.
    88  PARSEXFD-FIELD-IS-HIDDEN     VALUE 'Y' FALSE 'N'.
03  PARSEXFD-FIELD-READ-ONLY-FLAG    PIC X.
    88  PARSEXFD-FIELD-IS-READ-ONLY  VALUE 'Y' FALSE 'N'.
```

The values of the “field-description” parameter are defined as follows:

PARSEXFD-FIELD-OFFSET is the offset of the beginning of this field (zero-based).

PARSEXFD-FIELD-LENGTH is the number of bytes this field requires.

PARSEXFD-FIELD-TYPE describes the type of field. The types are defined as they appear in lib/sub.h, and are also listed in “parsexfd.def”.

PARSEXFD-FIELD-DIGITS is either the number of digits in this numeric field, or the length if the field is non-numeric.

PARSEXFD-FIELD-SCALE is either the scale of the numeric field or "0" if the field is non-numeric. The scale is defined as the power of ten by which the numeric value must be multiplied in order to get the actual value. For example, if the scale is -2, then there are two digits to the right of the decimal point.

PARSEXFD-USER-TYPE describes some of the XFD directives, as listed in "parsexfd.def". The UserDate, UserBinary and UserVarLength values are mutually exclusive (only one of them is set). SecondaryTable may be added to the value to signify that the SECONDARY-TABLE directive was also used.

PARSEXFD-FIELD-CONDITION is the condition that the field depends on. A condition of "0" means that the field is always included; "999" means that the field is a group item. In the latter case, the value may not be completely meaningful (if there are binary items in the group item).

PARSEXFD-FIELD-LEVEL is the level number of the field in the FD used to create this XFD.

PARSEXFD-FIELD-NAME is the name of the field. If EXCLUDE-ARRAYS was NOT used when parsing the XFD, and the field is part of a table, then the field name may include array indices.

PARSEXFD-FIELD-FORMAT is the date format specified in the XFD DATE directive.

PARSEXFD-FIELD-OCCURS-DEPTH is the number of valid elements in the OCCURS-TABLE.

PARSEXFD-FIELD-OCCURS-TABLE gives information about this element of a table. The OCC-MAX-IDX is the maximum index allowed. The OCC-THIS-IDX is the index of this element.

PARSEXFD-FIELD-IN-KEY-FLAG indicates whether this field is part of a key. The value is "Y" if this field is a part of one or more keys, or "N" if not.

PARSEXFD-FIELD-SECONDARY-FLAG indicates whether the SECONDARY-TABLE directive was used. The value is “Y” if so or “N” if not.

PARSEXFD-FIELD-HIDDEN-FLAG indicates whether the HIDDEN directive was used on this field. The value is “Y” if so or “N” if not.

PARSEXFD-FIELD-READ-ONLY-FLAG indicates whether the READ-ONLY directive was used on this field. The value is “Y” if so or “N” if not.

return-value 0 or 1

A return-value of “1” indicates that the operation was successful; a “0” indicates failure. For this operation, you will only see a return code of “0” if you specify an invalid field number (for example, if you try to retrieve information about field number 17 in a record that only has 15 fields).

PARSEXFD-TEST-CONDITIONS (op-code 4)

This operation tests the conditions of a particular record. It uses the following syntax:

```
CALL "C$PARSEXFD"
    USING PARSEXFD-TEST-CONDITIONS, xfd-handle,
          record-pointer.
```

The operation takes the following parameters:

xfd-handle

A valid handle returned by C$PARSEXFD PARSEXFD-PARSE.

record-pointer

This is a pointer to the record area on which to test conditions. (Because conditions are true or false depending on the value of particular fields, the values of those fields must be known. The only way to do this is to have a record from a file, specified with the PARSEXFD-PARSE op-code, to test against.)

After calling with this op-code, you can get each condition and tell whether fields that depend on that condition should be included in this record.

PARSEXFD-RELEASE (op-code 9)

This operation frees all memory associated with the XFD. It has the following syntax:

```
CALL "C$PARSEXFD" USING PARSEXFD-RELEASE, xfd-handle
```

This operation takes a single parameter:

xfd-handle

A valid handle returned by C$PARSEXFD PARSEXFD-PARSE.

After calling this op-code, do not reference the XFD handle. Doing so will result in undefined behavior, and may cause a memory access violation.

C$RECOVER

This routine opens the transaction log file defined in the **LOG_FILE** configuration variable and replays all of the file operations recorded in it on the appropriate data files.

Note that file systems other than Vision may require you to use their own recovery routines, instead of C$RECOVER. Please refer to your file system manufacturer's documentation to learn how the file system handles recovery procedures.

Usage

```
CALL "C$RECOVER"
```

Description

This routine allows you to recover from a hardware failure or power outage that may have left your data files in a corrupt state. C$RECOVER is helpful for recovering from error 98s and from any error that leaves your data damaged or possibly destroyed.

In order to use this routine, you need to have been logging transactions with the START TRANSACTION and COMMIT TRANSACTION verbs. These cause file operations to be logged in log files that you designate with the LOG-FILE and other configuration variables.

You also need a good backup of your data files. This backup should have been created just before you cleared or deleted the log file. You use your backup to restore the damaged files. Then you use C$RECOVER to read the log and recreate all of the file operations in the log, in the same sequence they were logged. These file operations are replayed on the data files.

The result is that your data files are brought up to date. After recovery, all files will be in a consistent state, because only committed (completed) transactions will be replayed from the log.

One way to use this routine is described here. First get everyone off the system. Then restore your clean backup files to their original locations. (It's a good idea to have more than one backup, because you will need a clean backup in order to recover.) Then run a COBOL program that calls the C$RECOVER routine (for example, CALL "C$RECOVER"). Make sure that the LOG-FILE configuration variable is set correctly. Repeat this procedure for each log file.

To make sure that your data can be recovered, follow these guidelines:

1. Commit all transactions before you make your data file backups.
2. Clear (or delete) your old log files as soon as your backup is complete and you know you have a good backup.
3. Don't permit any file activity between the time you make your backup and the time you clear or delete the logs.
4. It's all right to have multiple log files. You may periodically back up the log file and start a new one, if you are careful not to do this while there is any activity on the log or data files. Just be sure to run C$RECOVER on each log file in the same order in which they were written.
5. Don't permit any activity on the data files during recovery.
6. If any log file gets corrupted or destroyed, immediately make data file backups and clear or delete the old log.

C$REDIRECT

This routine is used to install and uninstall file I/O handlers. The routine gives you the ability to *redirect* file I/O operations to other, separate COBOL programs (handlers) that perform file I/O operations in addition to, or instead of the original operation.

For example, you can augment a file I/O operation to act on an additional file without rewriting the original application. C$REDIRECT specifies the name of the handler(s) that performs the additional operations. The handler is simply a COBOL program. This feature is activated or deactivated by calling C$REDIRECT. Information is passed to the handler program using standard COBOL linkage items. Each program has its own set of I/O handlers. This means that only the module (or subprogram) calling C$REDIRECT is affected.

Usage

```
CALL "C$REDIRECT"
    USING HANDLER-FUNCTION, HANDLER-VERSION, HANDLER-NAME,
        [PREVIOUS-HANDLER-NAME],
    GIVING HANDLER-STATUS
```

Parameter

HANDLER-FUNCTION any numeric value

This parameter has three possible values, HANDLER-FUNCTION-PRE, HANDLER-FUNCTION-REPLACE, and HANDLER-FUNCTION-POST (defined in “filesys.def”).

HANDLER-VERSION any numeric value

Specifies the version of the linkage items to be passed to the handler program (defined in “filesys.def”). The linkage items are defined in “handler.cpy”. If later releases change the format of the linkage items, you can use HANDLER-VERSION to specify which version of the linkage items to use. Currently, the only legal values for this parameter are “1” and “2”.

HANDLER-NAME PIC X(n)

Specifies the name of the handler program to be installed. Use NULL to uninstall a handler that has been previously installed.

PREVIOUS-HANDLER-NAME PIC X(n) (optional)

After a successful call to C$REDIRECT, this data item holds the name of the previously installed HANDLER-FUNCTION. It will contain spaces if there was no previous handler of this type.

HANDLER-STATUS signed numeric value (optional)

After a call to C$REDIRECT, this data item contains the status of the action. A "1" indicates that the install or uninstall action has succeeded and "0" indicates failure. Possible reasons for failure include an unsupported HANDLER-VERSION or an unsupported operation version.

Description

C$REDIRECT allows you to install as many as three discrete handlers:

- The "pre" handler executes before the file I/O statement.
- The "replace" handler executes in place of the file I/O statement.
- The "post" handler executes after the file I/O statement.

These handlers may be used together in any combination, but each must be installed with a separate call to C$REDIRECT. All standard file I/O statements trigger the installed handlers (standard file I/O statements include: OPEN, CLOSE, READ, READ NEXT, READ PREVIOUS, WRITE, REWRITE, DELETE, DELETE FILE, START, COMMIT, and ROLLBACK). Information is passed to the handlers via standard COBOL linkage items. These items are described in the file "handler.cpy" that is installed with the ACUCOBOL-GT development system. Once a handler is installed, all file I/O for all files is redirected through the handler.

This routine can be used with the SORT and MERGE statements. The I/O handlers are used on the files listed in the USING and GIVING phrases when these verbs execute implicit OPEN, READ, NEXT, WRITE, AND CLOSE operations. In this situation, the handlers will be called more than once for each execution of the verb.

When an installed handler executes, it can return its status via the linkage item HANDLER-STATUS-CODE. This item is meant to return the standard COBOL file status code that is normally returned by a file operation. The value returned is made available in the file's status variable. The set of codes used is up to the developer, as long as they follow these rules:

- Any code that starts with a "0" is considered successful.
- Any code that starts with a "1" is considered to be an "at end" condition.
- Any code that starts with a "2" is considered to be an "invalid key" condition.

The first handler to return an unsuccessful status code will be the last portion of the file operation to be executed, whether the remaining operation is the regular file operation or another handler. For example, a "pre" handler returning an error will preclude the execution of the normal file operation (or a "replace" handler, if defined) and the "post" handler, if defined. Declaratives are run as appropriate when a handler returns an error.

Note: C$RERR and other library functions that report internal file error codes may not return the expected results after executing a handler program.

C$REGEXP

This routine allows you to search strings using regular expressions. This section includes the following topics:

Usage

```
CALL "C$REGEXP"
    USING OP-CODE, parameters
    GIVING return-value
```

Parameters

OP-CODE Numeric data item

Specifies the operation to perform. Each operation is defined in "acucobol.def" and is described in detail in the "OP-CODES and parameters" section below. Op-codes include:

Code	Operation
1	AREGEXP-GET-LEVEL
2	AREGEXP-COMPILE
3	AREGEXP-MATCH
4	AREGEXP-RELEASE-MATCH
5	AREGEXP-RELEASE
6	AREGEXP-NUMGROUPS
7	AREGEXP-GETMATCH
20	AREGEXP-LAST-ERROR

parameters Type varies (defined in "acucobol.def")

Parameters vary depending on the operation selected. They provide information and hold results.

return-value Numeric data item

Unless otherwise noted, each operation returns a value or a status in return-value. Its contents vary by operation and the result of the operation.

Description

This routine allows you to use a regular expression to search a text string.

A regular expression is a formula for matching strings that have a certain pattern. For a complete description of regular expressions, refer to the POSIX 1003.2 standard appropriate for your platform. Windows platforms use the CAtlRegExp library; UNIX platforms use the POSIX C routines native to the platform.

A simple use of C$REGEXP is outlined in the following steps.

1. Use the AREGEXP-GET-LEVEL op-code to validate that the host platform provides support for regular expressions.
2. Validate and compile your regular expression with op-code AREGEXP-COMPILE. Your program should include an error handling routine in the event that the compiler finds an error in the expression.
3. Use op-code AREGEXP-MATCH to apply a compiled regular expression to a string to search for a match. You may want to do this iteratively to find all matches in the string.
4. Use op-codes AREGEXP-NUMGROUPS and AREGEXP-GETMATCH to work with subexpression matches.
5. Manage the memory used by this routine with op-codes AREGEXP-RELEASE-MATCH and AREGEXP-RELEASE.

OP-CODES and parameters

AREGEXP-GET-LEVEL (op-code 1)

This operation indicates whether regular expression support is available on the host. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-GET-LEVEL GIVING return-value
```

The value of return-value can be one of the following (defined in “acucobol.def”):

AREGEXP-NONE	0	regular expression processing is not available
AREGEXP-WINDOWS	1	Windows regular expressions supported
AREGEXP-POSIX	2	POSIX regular expressions supported

AREGEXP-COMPILE (op-code 2)

This operation compiles a regular expression to ensure that it has a valid form, returning a handle to the compiled regular expression or NULL if there is an error. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-COMPILE, reg-expr, flags
                GIVING return-value
```

reg-expr must be a NULL-terminated regular expression. It must be NULL-terminated because trailing spaces are allowed in regular expressions.

flags (optional) is the sum of one or more of the following values (defined in "acucobol.def"):

AREGEXP_COMPILE_IGNORECASE	1	Ignore case when matching patterns. (Windows or UNIX)
AREGEXP_COMPILE_BASIC	2	Change the type of regular expression from *extended* to *basic*. (UNIX only) (For an explanation of *extended* and *basic*, see the POSIX 1003.2 standard.)
AREGEXP_COMPILE_NO_SPECIAL	4	Treat all characters as ordinary characters with no special meaning. (UNIX only)
AREGEXP_COMPILE_NO_SUB	8	When matching, determine only if there is a match, without returning the offsets of the match. (UNIX only)
AREGEXP_COMPILE_NEWLINE	16	Treat newlines as special (end-of-line marker) characters. (UNIX only)

return-value contains a handle to the compiled expression, or NULL if an error occurred.

AREGEXP-MATCH (op-code 3)

This operation applies a regular expression to a string, and returns a handle. To see if there is a match you need to check match-start. If match-start is "0" there is no match. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-MATCH,
```

```
reg-expr-handle, string, length, match-start, match-end
GIVING return-value
```

reg-expr-handle is a handle to a regular expression returned by AREGEXP-COMPILE.

string is the string to test for a match.

length is the length of *string*. If *length* is zero, the size of *string* is used.

match-start returns the index of the start of the pattern that matched.

match-end returns one byte beyond the pattern that matched. To test the string for additional matches, start a new AREGEXP-MATCH at the match-end offset.

return-value contains a handle to the match or zero if no match is found or an error occurred.

AREGEXP-RELEASE-MATCH (op-code 4)

This operation frees memory that is allocated when AREGEXP-MATCH is called. Return-value is not used. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-RELEASE-MATCH match-handle
```

match-handle is a handle to a match returned by AREGEXP-MATCH.

AREGEXP-RELEASE (op-code 5)

This operation frees the memory allocated when AREGEXP-COMPILE is called. Return-value is not used. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-RELEASE reg-expr-handle
```

reg-expr-handle is a handle to a regular expression returned by AREGEXP-COMPILE.

AREGEXP-NUMGROUPS (op-code 6)

This operation returns the number of substrings that matched any subgroups in the regular expression. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-NUMGROUPS match-handle
                GIVING return-value
```

match-handle is a handle returned by AREGEXP-MATCH.

return-value returns the number of matches.

Depending on the construction of a regular expression, it is possible for a subgroup of the regular expression to match multiple substrings. This operation reports the number of instances found in the last AREGEXP-MATCH operation. For more information, rules, and examples, see the POSIX 1003.2 documentation or one of the many books available on regular expressions.

AREGEXP-GETMATCH (op-code 7)

This operation returns a set of indices into a string passed to AREGEXP-MATCH that match the subexpression of the regular expression. Its usage is:

```
CALL "C$REGEXP"
    USING AREGEXP-GETMATCH, match-handle, group,
    idx-start, idx-end
    GIVING return-value
```

The parameters are defined as follows:

match-handle is a handle returned by AREGEXP-MATCH.

group is a number between “1” and the value returned by AREGEXP-NUMGROUPS.

idx-start returns an index into the beginning of the string that matches the subexpression of the regular expression.

idx-end returns an index to the end of the string that matches the subexpression of the regular expression.

return-value returns “1” if the operation succeeds, and zero if there is an error.

AREGEXP-LAST-ERROR (op-code 20)

This operation returns the last error code returned by a call to C$REGEXP. Its usage is:

```
CALL "C$REGEXP" USING AREGEXP-LAST-ERROR GIVING return-value
```

The error value is returned in return-value. The possible error values (described in “acucobol.def”) have the following meanings:

AREGEXP-ERROR-NO-ERROR	0	No error
AREGEXP-ERROR-NO-MEMORY	1	Insufficient memory to handle the request
AREGEXP-ERROR-BRACE-EXPECTED	2	A closing brace is missing
AREGEXP-ERROR-PAREN-EXPECTED	3	A closing parenthesis is missing
AREGEXP-ERROR-BRACKET-EXPECTED	4	A closing bracket is missing
AREGEXP-ERROR-UNEXPECTED	5	An unknown error occurred
AREGEXP-ERROR-EMPTY-RANGE	6	An empty range was given
AREGEXP-ERROR-INVALID-GROUP	7	The group provided was invalid
AREGEXP-ERROR-INVALID-RANGE	8	An invalid range was given
AREGEXP-ERROR-EMPTY-REPEATOP	9	A repeat operator was given on an empty subexpression
AREGEXP-ERROR-INVALID-INPUT	10	The input was invalid
AREGEXP-ERROR-INVALID-HANDLE	11	The handle is not a regular expression handle or a match handle

AREGEXP-ERROR-INVALID-ARGUMENT	12	One of the arguments given is invalid
AREGEXP-ERROR-INVALID-CALL-SEQ	13	The order of C$REGEXP operations is an invalid sequence.
AREGEXP-ERROR-NO-MATCH	14	The regular expression did not find a match in the given string.

Note: If the error code returned does not match a value in the list, it may be that the value is coming from the host's regular expression library. Refer to the documentation for the host's regular expression library.

C$RERR

This routine returns extended file status information.

Usage

```
CALL "C$RERR"
    USING EXTEND-STAT, TEXT-MESSAGE, STATUS-TYPE
```

Parameters

EXTEND-STAT (returned) PIC X(5) or larger; PIC X(7) or larger under VMS

TEXT-MESSAGE (returned) PIC X(n)

STATUS-TYPE (input) Numeric parameter

Comments

C$RERR must be passed a USING argument that should be PICTURE X(5) or larger. This argument is filled in with either the extended *file status* caused by the last file I/O, or the extended *transaction status* caused by the last transaction operation. The value of STATUS-TYPE determines which status will be filled in.

A text message is available for some 9D errors, which are host system errors. When a text message is available, it's moved into the data item TEXT-MESSAGE, and returned as the second parameter. You can use TEXT-MESSAGE to display text to the user, so the user can decide what action to take. This parameter is ignored for transaction status.

The third parameter, STATUS-TYPE, controls whether file status or transaction status information is returned by the routine. If STATUS-TYPE is set to "1" or the parameter is omitted, file status is returned. If STATUS-TYPE is set to "2", transaction status is returned.

The first two characters of the extended file status are identical to the normal FILE STATUS value returned by ACUCOBOL-GT for a file operation. The last two characters further clarify the reason for the particular FILE STATUS value. The values used here are listed in the file status table found in Appendix E. If the file status (first two characters) is "30", the remainder of the information is the operating system's status code explaining what caused the error.

On some systems, the operating system requires more than two digits for its status codes. That is why the C$RERR routine may be passed a field that is larger than four characters. Whenever an error "30" occurs, the operating system's status value is returned in this extended field. The number returned is a left-justified decimal value. If the receiving field is too small, the right-most digits are returned. If the receiving field is too large, the excess characters are filled with spaces.

The first two characters of the extended transaction status are identical to the contents of the TRANSACTION-STATUS register. The last two characters further clarify the reason for the particular transaction status value. The values used here are listed in the transaction status table found in Appendix E, **Section E.4**.

In a few cases, there may also be a tertiary status code. If there is one, a comma will be placed immediately after the secondary status code, and the tertiary code will then appear left-justified.

C$RERRNAME

This routine returns the name of the last file used in an I/O statement. It can be used in conjunction with **C$RERR** to determine file errors.

Usage

```
CALL "C$RERRNAME"
    USING FILE-NAME
```

Parameter

FILE-NAME PIC X(n)

This field is filled in with the name of the last file that was involved in an I/O statement. If the last statement was a transaction statement, then this routine returns spaces. The file name returned is the one found in the file's ASSIGN clause when it was opened. Any name translations specified in the configuration file are reflected in the returned name, but the FILE-PREFIX and FILE-SUFFIX used are not.

C$RESOURCE

Use C$RESOURCE to load a resource file and get a resource handle or to destroy the resource handle and free any memory associated with it. The AcuBench Screen Designer saves changes made to an ActiveX control in a "state" resource file. The runtime uses the information in this resource file when displaying the ActiveX control.

After an ActiveX control is displayed you may destroy the resource handle used to initialize it. The ActiveX control only references this resource handle during the first DISPLAY or MODIFY which sets the initial-state property (usually the first display). In general, you should destroy resource handles to free memory after all ActiveX controls that use them have been displayed.

Usage

```
CALL "C$RESOURCE"
    USING OP-CODE, parameters
    GIVING RESOURCE-HANDLE
```

Parameters

OP-CODE Numeric parameter

Op-code indicates the desired operation. The file “activex.def” contains level 78 symbolic names for these operations.

Parameters Vary depending on the op-code chosen. These are described in the Comments section below.

RESOURCE-HANDLE PIC 9(9)

RESOURCE-HANDLE holds the return value of C$RESOURCE. Values less than or equal to zero indicate errors. This is only used by the CRESOURCE-LOAD operation.

Comments

There are two operations available in C$RESOURCE:

1. load a resource file and return a resource handle
2. destroy a resource handle and free its associated memory

To use C$RESOURCE, you pass an op-code as the first parameter, followed by one additional parameter. The op-code determines which operation is performed and the meaning of the additional parameter.

Currently, there are two operations: CRESOURCE-LOAD and CRESOURCE-DESTROY:

CRESOURCE-LOAD This operation loads a resource file from disk and returns a resource handle. It takes one additional parameter.

NAME	This is an alphanumeric literal or data item that is the file name of the resource file. This must be a resource file generated by AcuBench. The file usually ends with ".res".

Note: When you are running in a thin client environment, and a file name beginning with "@[DISPLAY]" is passed to this routine, it will attempt to access the file in the display host's file system. It does not download the file from the server. For more information, refer to section 7.2, "Using Library Routines and DLLs in Thin Client." of the *AcuConnect User's Guide*.

CRESOURCE-DESTROY This operation destroys a resource handle and releases its memory. It takes one additional parameter.

RES-HANDLE	This is a resource handle returned by CRESOURCE-LOAD.

Note: The behavior of this library routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the documentation on FILENAME_SPACES in **Appendix H, "Appendix H: Configuration Variables,"** for information about the terminating character for path names.

Example

```
CALL "C$RESOURCE" USING CRESOURCE-LOAD, "PROGRAM1.RES"
   GIVING RES-HANDLE.
IF RES-HANDLE > 0 THEN
   DISPLAY MSCHART LINE 10 COLUMN 10 LINE 5 SIZE 40
      INITIAL-STATE = (RES-HANDLE, "MSCHART-1-INITIAL-STATE")
      HANDLE IN MSCHART-1
   CALL "C$RESOURCE" USING CRESOURCE-DESTROY, RES-HANDLE
END-IF
```

C$RUN

ACUCOBOL-GT for Windows (including Thin Client) supports an alternate method for running other programs. This is through the library routine C$RUN. This library routine works identically to the SYSTEM library routine, except that the calling program does not wait for the called program to finish. Instead, both programs run in parallel.

Usage

```
CALL "C$RUN"
    USING COMMAND-LINE,
    GIVING STATUS-VAL
```

Parameters

COMMAND-LINE PIC X(n)

Contains the operating system command line to execute.

STATUS-VAL Any numeric data item

Returns “0” if successful or “-1” if not.

Description

C$RUN sets STATUS-VAL to “-1” if the call fails or to “0” if it succeeds.

C$RUN is implemented only under the Windows and Windows NT versions of ACUCOBOL-GT. On other systems, it always returns “-1”.

C$RUN is supported in Thin Client environments. To execute a program on the display host in a thin client environment, add the prefix "@[DISPLAY]:" to the name of any program that resides on the client machine. For example:

```
C$RUN "@[DISPLAY]:C:\notepad myfile.txt
```

C$SETERRORFILE

This routine sets the name of the runtime error file. Format specifiers may be used in the name. If the resulting file name differs from the current runtime error file name, the runtime closes the current error file, attempts to rename it to the new name, and opens the new file in extend mode. Subsequent error and trace messages are written to the new error file. Note that the rename operation overwrites an existing file. Also, the rename operation may fail if the original error file is in use by other runtime processes. In this case the original file will be left alone and the new file will start empty.

Usage

```
CALL "C$SETERRORFILE"
    USING ERROR-FILE-NAME
```

Parameters

ERROR-FILE-NAME PIC X(n)

Contains the name of the error file. The runtime error file format specifiers may be used.

Comments

This routine provides a means for applications to embed identifying information in the runtime error file name. For example, after a user logs in, an application can change the name of the runtime error file so that it includes the user name.

If the runtime fails and the user calls customer support, the support analyst can search the directory containing runtime error files and quickly identify the one with the user's name to help resolve the issue. This is especially useful in situations where the runtime was launched from AcuConnect using an entry in the AcuConnect alias file. Since one alias is used by many users, there is potentially only one error file. In this case, error messages tied to failures experienced by a particular user may be lost or difficult to read because multiple runtime processes are writing to the same error file. C$SETERRORFILE helps by allowing you to set the error file name from COBOL for a particular runtime instance.

C$SETEVENTDATA

When an ActiveX control or COM object event procedure exits, it can send information back to the control through the event parameters. These parameters are stored in the control and can be set by calling C$SETEVENTDATA before the control's event procedure exits.

Usage

```
CALL "C$SETEVENTDATA"
    USING EVENT-CONTROL-HANDLE, SRC-ITEM-1, [SRC-ITEM-2, ...]
    GIVING RESULT-CODE
```

Parameters

EVENT-CONTROL-HANDLE USAGE HANDLE

Handle to the control that triggered the event.

SRC-ITEM-1 Any COBOL data type

The first source data item.

SRC-ITEM-2, . . . Any COBOL data type (optional)

Any number of source items.

RESULT-CODE Signed numeric value

Receives the result-code for the operation. This will be 0 to indicate success or a negative value to indicate failure. (Microsoft defines many standard "result codes" in their documentation. Note that these are usually in hexadecimal notation.)

Comments

C$SETEVENTDATA converts the COBOL-type data in the source items to the corresponding event parameter types.

You are responsible for specifying compatible types. For example, if the source item you specify is alphabetic, and the event parameter you are setting is a signed integer, C$SETEVENTDATA will try to read a number from the alphabetic item and move it to the event parameter. This is not a programming error and neither the compiler nor runtime will warn you about it.

Example

Suppose you have displayed an ActiveX control that triggers an event called AxEventOne which has three parameters. You would use the following COBOL syntax to get the event parameters, add two to each one and set the event parameters to their new values:

```
WHEN AxEventOne
   CALL "C$GETEVENTDATA" USING EVENT-CONTROL-HANDLE,
      PARAM-1, PARAM-2, PARAM-3
   ADD 2 TO PARAM-1
   ADD 2 TO PARAM-2
   ADD 2 TO PARAM-3
   CALL "C$SETEVENTDATA" USING EVENT-CONTROL-HANDLE,
      PARAM-1, PARAM-2, PARAM-3
```

For more examples of how to set event parameters for ActiveX events, refer to section 4.4 in *A Guide to Interoperating with ACUCOBOL-GT.*

C$SETEVENTPARAM

C$SETEVENTPARAM is an alternate way to set event parameters for ActiveX controls. Use it to set a single event parameter when there are several for an event. To use this routine you must know the actual name of the parameter. You may determine these names by reading the ActiveX control's documentation or by looking at the definitions in the copy book for the ActiveX control.

It is common for an ActiveX event to receive many parameters. C$SETEVENTPARAM allows you to set the values of only the parameters you care about.

Please note that C$SETEVENTPARAM cannot be used to set event parameters for COM objects. You must use C$SETEVENTDATA for COM objects.

Usage

```
CALL "C$SETEVENTPARAM"
    USING EVENT-CONTROL-HANDLE, PARAM-NAME, PARAM-VALUE
    GIVING RESULT-CODE
```

Parameters

EVENT-CONTROL-HANDLE USAGE HANDLE

Handle to the control that generated the event.

PARAM-NAME PIC X(n)

The symbolic name of the event parameter.

PARAM-VALUE Any COBOL data type

Source item containing the event parameter's value.

RESULT-CODE Signed numeric value

Receives the result-code for the operation. This will be 0 to indicate success or a negative value to indicate failure. (Microsoft defines many standard "result codes" in their documentation. Note that these are usually in hexadecimal notation.)

Comments

C$SETEVENTPARAM converts the COBOL-type data in the source item to the named event parameter's type. Using this routine instead of C$SETEVENTDATA will make your code more readable. The object code will be a little larger and calls to this routine will take a little longer than calls to C$SETEVENTDATA. However, these differences will probably be unnoticeable and the benefits of readable code outweigh the performance and size considerations.

You are responsible for specifying a compatible types. For example, if the source item you specify is alphabetic, and the event parameter you are setting is a signed integer, C$SETEVENTPARAM tries to read a number from the alphabetic item and move it to the event parameter. This is not a programming error and neither the compiler nor runtime warns you about it.

Example

Suppose you have displayed an ActiveX control that triggers an event called AxEventOne which has three parameters. Then suppose that PARAM-1 and PARAM-2 contain information about the event and that only PARAM-3 is meant to be set by the event procedure. Since PARAM-3 is the third parameter, to set it you would have to pass two “dummy” parameters using C$SETEVENTDATA. For example:

```
CALL "C$SETEVENTDATA"
   USING EVENT-CONTROL-HANDLE, 0, 0, PARAM-3.
```

However, If you determined that the name of PARAM-3 in the ActiveX control was”Param3”. You could then use C$SETEVENTPARAM to accomplish this task in a more elegant and readable way. For example,

```
CALL "C$SETEVENTPARAM"
   USING EVENT-CONTROL-HANDLE, "Param3", PARAM-3.
```

For more examples of how to set event parameters for ActiveX events, refer to section 4.4 in *A Guide to Interoperating with ACUCOBOL-GT.*

C$SETVARIANT

This routine sets data items referenced by Variant handles. Note that there is also a **C$GETVARIANT** routine that retrieves data referenced by a Variant handle.

C$SETVARIANT converts COBOL type data to Variant type data. Programs that call ACUCOBOL-GT using the ACUCOBOL-GT Automation Server or runtime DLL pass their parameters (by reference) as Variant types. The COBOL program receives handles to the Variant data. C$SETVARIANT sets the data associated with a particular handle to the value of a COBOL data item. The data is automatically converted to the proper Variant format.

Usage

```
CALL "C$SETVARIANT"
    USING SRC-ITEM, H-VARIANT
    GIVING RESULT-CODE
```

Parameters

SRC-ITEM Any COBOL data type

The source data item.

H-VARIANT USAGE HANDLE

Handle to Variant type data. Data passed in from a program calling ACUCOBOL-GT using the ACUCOBOL-GT Automation Server or runtime DLL is in the form of handles to Variant type data.

RESULT-CODE Signed numeric value

Receives the result code for the operation. This will be 0 or a positive value to indicate success or a negative value to indicate failure.

Under Microsoft Windows this is a code of type HRESULT that can be looked up in Microsoft documentation to determine the reason for the failure or additional information about the success.

Comments

The COBOL data item SRC-ITEM is converted to Variant type data, and is stored in the Variant item associated with H-VARIANT.

C$SETVARIANT creates a new variant to store the initial value of the handle item passed to it is LOW-VALUES or SPACES. (It is your responsibility to free this variant using the DESTROY verb.)

The C$SETVARIANT library routine and OLE SAFEARRAY data type are supported in thin client environments, regardless of the kind of operating system on the server. However, be careful when using C$SETVARIANT in thin client environments, because it generates network traffic and can affect

performance. When using this library routine in a thin client environment, you should pass only small amounts of data. Note that **C$GETVARIANT** is not supported in thin client environments at this time.

C$SLEEP

This routine causes the program to pause in a machine efficient fashion.

Usage

```
CALL "C$SLEEP"
    USING NUM-SEC
```

Parameter

NUM-SEC Numeric or alphanumeric parameter

The number of seconds to *sleep*.

This parameter is a an unsigned fixed-point numeric parameter, or an alphanumeric data item containing an unsigned fixed-point number.

Description

This routine can be used to impose slight delays in loops. For example, you might want to introduce a delay in a loop that is waiting for a record to become unlocked. Calling C$SLEEP will allow the machine to execute other programs while you wait.

The C$SLEEP routine is passed one argument. This argument is the number of seconds you want to pause. For example, to pause the program for five and a half seconds, you could use either of the following:

```
CALL "C$SLEEP" USING 5.5
CALL "C$SLEEP" USING "5.5"
```

The amount of time paused is only approximate. Depending on the granularity of the system clock and the current load on the machine, the time paused may actually be shorter or longer than the time requested. Typically, the time paused will be within one second or one-tenth of a second of the amount requested (unless the machine is excessively loaded).

If the sleep duration is zero, this function does nothing. If the sleep duration is signed, this function generates a runtime error.

C$SOCKET

This routine can be used to communicate with other processes on remote or local hosts. This provides an interface to inter-process communication via sockets.

Usage

```
CALL "C$SOCKET"
    USING OP-CODE, parameters
```

Parameters

OP-CODE Numeric parameter

Specifies the operation to perform. These are defined in the description below.

Parameters Various types defined in “socket.def”.

The remaining parameters vary depending on the operation selected. They provide information and hold results for the operations specified.

Description

All parameters passed to C$SOCKET are passed BY REFERENCE. The C$SOCKET routine provides any necessary conversions. Numeric arguments passed to this routine must be declared as COMP 5.

Note: If a COBOL thread calls one of these operations, all threads are blocked until the operation is finished and control is returned to the COBOL program.

AGS-CREATE-SERVER (op-code 1)

This operation creates a server-side socket. It must be called once at the beginning of any service you create. If the call is successful, the value in RETURN-CODE should be moved to a data item that is USAGE HANDLE. This data item is then passed as the socket handle to AGS-ACCEPT or AGS-NEXT-READ. This socket handle is not available for read or write operations. You can use the GIVING phrase instead of MOVE to store the value in a data item. If the call fails, RETURN-CODE is NULL. This operation has a single parameter:

port-number	a numeric value specifying the port on which the socket is created. All clients must use the same port number to connect to this server.

AGS-ACCEPT (op-code 2)

This operation waits for a connection from a client. It blocks other calls while waiting, and returns only after a client has attempted to connect. If a client successfully connects, the value in RETURN-CODE is a socket handle that may be used to communicate with the client using AGS-WRITE and AGS-READ. This operation is called once if the server is a single-client server. In that case, once the client has connected it is safe to close the original server socket (created in AGS-CREATE-SERVER) using AGS-CLOSE, and use only the socket handle returned by this operation. This operation has a single parameter:

socket-handle	this is returned by a call from AGS-CREATE-SERVER.

AGS-CREATE-CLIENT (op-code 3)

This operation attempts to connect to a server. It waits only a short time before giving up, so the server should be running before the client makes this call. (The length of time is dependent on the underlying socket layer.) If it is successful, the value in RETURN-CODE is a socket handle that can be used to communicate with the server using AGS-WRITE and AGS-READ. The socket handle should be moved to a data item of USAGE HANDLE. This operation takes two parameters:

port-number a numeric value specifying the port on which the socket is created.

server-name a PIC X(n) data item that holds the machine name of the server.

AGS-CLOSE (op-code 4)

This operation closes a socket handle. After closing a socket handle, it should no longer be referenced. This operation takes a single parameter:

socket-handle this indicates which socket to close.

AGS-WRITE (op-code 5)

This operation writes data to a socket, either from the client to the server, or from the server to the client. With this operation, data is actually written:

- when AGS-READ is attempted on that socket,
- when you are querying for any available sockets to read using AGS-READ or AGS-NEXT-READ, or
- when you call AGS-FLUSH.

The value in RETURN-CODE is the number of bytes written. If this is different than the length parameter, an error has occurred.

Note: It is up to you to make sure that the server is writing when the client is reading, and vice versa. If both attempt to read at the same time, a deadlock will result.

This operation takes three parameters:

socket-handle a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.

buffer indicates a buffer to write. It can be of any format, including a group item.

Note: Be careful when sending numeric data across the network because some machines use different byte ordering than others and native numeric data can appear swapped on different machines. COMP-4 data is in the order that most network servers expect for binary data. If you are communicating with a non-COBOL client or server, you should use COMP-4 data of the correct size for the machine in question. If your client and server are both COBOL, you can use standard COBOL types.

length — the number of bytes to write. If the buffer passed is smaller than the value of this parameter, an error will result.

AGS-READ (op-code 6)

This operation reads data from a socket. It blocks other calls until all the data requested is actually read, or an error occurs. The value returned in RETURN-CODE is the number of bytes actually read. If this is different than the length parameter, an error occurred. If the socket is closed before the entire buffer is filled, C$SOCKET will return the number of bytes read to that point, which will be less than the amount requested. The next time AGS-READ is called, C$SOCKET will return -1 to signify that the socket is closed. This operation takes four parameters:

socket-handle — a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.

buffer — indicates a buffer to read. It can be of any format, including a group item. (The same cautions about byte order described in AGS-WRITE apply here.)

length — the number of bytes to read. If the buffer passed is smaller than the value of this parameter, an error will result. If length = 0, then the return value is the number of bytes available to be read on the socket. In other words, after calling AGS-READ with a length of “0”, you can call AGS-READ again with a length equal to the previous return value and be guaranteed not to block. If length is negative, the data is moved to the buffer, but it is also left in the socket so that a future

call to AGS-READ can read it again. In this case, the number of bytes transferred to the buffer is the absolute value of length.

timeval — a number, measured in milliseconds, that determines how long to execute this operation. If the specified time-out period passes, *buffer* will contain as much data as is available, and the return value will be the number of bytes read. This will probably be less than the number of bytes desired. This allows COBOL programs to wait for data that may not come.

AGS-FLUSH (op-code 7)

This operation flushes any data in the socket, sending any data that has been written, and checking for data to be read. The data to be read is stored in an internal buffer, awaiting a call to AGS-READ. This operation returns no value. This operation takes a single parameter:

socket-handle — this indicates which socket handle to flush.

AGS-EMPTY (op-code 8)

This operation is similar to AGS-READ, except that the number of bytes is thrown away, rather than being stored. This operation takes two parameters:

socket-handle — a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.

length — the number of bytes to throw away. Setting this parameter means that AGS-EMPTY will not complete until that many bytes are available on the socket to throw away.

AGS-GETHOSTNAME (op-code 9)

This operation allows the COBOL program to get the name of the host machine on which the COBOL program is executing. The return value is “0” on success, and “-1” on error. This operation takes a single parameter:

hostname this parameter should be a PIC X(n) parameter. The name of the host machine is stored in this data item.

AGS-LAST-ERROR (op-code 10)

This operation allows the COBOL program to determine the last error on a socket. This information is only meaningful when an error has occurred. It is useful if one of the operations that returns a socket handle returns an error instead. The value stored in RETURN-CODE is the error number. The error numbers are listed by name in "socket.def". (To interpret these error codes, refer to third-party documentation about sockets.) This operation takes a single parameter:

socket-handle a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ, or a NULL socket handle.

AGS-NEXT-READ (op-code 11)

This operation allows you to create multi-client servers. It waits until data is ready to be read from one of the sockets your server has created. Note that this operation only returns information about sockets created as children of the server socket passed (meaning that it only waits for sockets that were ACCEPTed from that socket) and ignores all other sockets. It automatically accepts new client connections from AGS-CREATE-CLIENT, and returns the corresponding socket handle as one that can be read. The value in RETURN-CODE is a socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ, "0" to signify that the timeval has elapsed, or "-1" to signify an error. (The socket handle returned from this operation may be from a new client which has connected and sent data. If your program has not yet recognized this as a valid socket, the value may be unfamiliar to you.) This operation takes two parameters:

server-socket-handle the socket handle returned by AGS-CREATE-SERVER.

timeval a number, measured in microseconds, that determines how long to execute this operation. The first time you call AGS-NEXT-READ, *timeval* determines the

length of time the operation executes. All subsequent invocations of this operation are handled in one of two ways:

1) If there is a socket handle that has already been determined to have data available, that socket handle is returned immediately regardless of the value of *timeval*.

2) If all sockets have been processed, then the setting of timeval determines how long the routine will execute, just as in the initial call. Any sockets that receive data during that time will store the data until AGS-READ or AGS-EMPTY is called with that socket as an argument. Any future calls to AGS-NEXT-READ will return those socket handles.

If *timeval* is set to "0", all the sockets are checked and the operation returns immediately. If no sockets have data, the routine returns "0". Otherwise the return value will be a socket that has data. If *timeval* is set to the default of "-1", the operation waits until a socket which can be read is available (potentially forever). If all your server does is service clients, then you should always pass the value of "-1" as the *timeval*. The only reason to pass a *timeval* that is not "-1" is if your server wants to perform other work while it is not busy servicing clients.

AGS-REMOTE-NAME (op-code 12)

This operation returns the name of a remote machine. It takes two parameters:

socket-handle	a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.
remote-name	a PIC X(n) data item that is filled with the name of the remote machine.

AGS-REMOTE-ADDR (op-code 13)

This operation returns the IP address of a remote machine. It takes two parameters:

socket-handle — a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.

remote-addr — a PIC X(n) data item that is filled with the IP address of the remote machine.

AGS-READ-LINE (op-code 14)

This operation reads a line of data from a socket. A line is defined as a block of text terminated by either a NewLine or Carriage-Return / NewLine (NL or CRNL). The NL or CRNL is stripped from the data before it is returned, and the return value is the number of bytes read, not counting the NL or CRNL. This operation blocks other calls until all the data requested is actually read, or an error occurs. This operation takes four parameters:

socket-handle — a valid socket handle returned from AGS-ACCEPT, AGS-CREATE-CLIENT, or AGS-NEXT-READ.

buffer — indicates a buffer to read. It can be of any format, including a group item. (The same cautions about byte order described in AGS-WRITE apply here.)

length — the number of bytes to read. If the buffer passed is smaller than the value of this parameter, an error results. If length = 0, then the return value is the number of bytes available to be read on the socket. In other words, after calling AGS-READ-LINE with a length of 0, you can call AGS-READ-LINE again with a length equal to the previous return value and be guaranteed not to block. If length is negative, the data is moved to the buffer, but it is also left in the socket so that a future call to AGS-READ-LINE can read it again. In this case, the number of bytes transferred to the buffer is the absolute value of length.

timeval a number, measured in milliseconds, that determines how long to execute this operation. If the specified time-out period passes, *buffer* will contain as much data as is available, and the return value will be the number of bytes read. This will probably be less than the number of bytes desired. This allows COBOL programs to wait for data that may not come.

AGS-GETHOSTADDR (op-code 15)

This operation code takes a single argument:

host-address A pic x(15) or larger item. This is the address of the local host in dotted notation (192.15.4.32) when C$SOCKET returns.

AGS-GETSOCKETPORT (op-code 16)

This operation code takes a single argument:

socket-handle This is a socket handle. The return value is the port being used by that socket, or "-1" on error. This is useful when passing "0" as the port number when creating a socket, since a socket is created on some unknown port in that case. This function can then be used to determine the actual port being used.

Examples

Three sample programs are provided to illustrate the use of the C$SOCKET routine.

- "sockcli.cbl" is a client that connects to a server and sends it data. The server returns the data modified.
- "socksrv1.cbl" is a single-client server. This means that it can accommodate a single client. When that client shuts down, the server also halts. If another client subsequently tries to connect to the server, it is ignored.

- "socksrvm.cbl" is a multi-client server. Any number of clients can attach to this server. When one client disconnects, the rest continue to be serviced.

Because of limitations in the "socksrvm.cbl" sample program, the only way to halt the server is to kill it at the operating system level. This is not a general requirement of multi-client servers.

C$SYSLOG

This library routine can be used to open, write to, and close the system log. Using this routine, you can write to the system log or event notification system (on Windows) in the event of a serious error that administrators need to know about.

Usage

```
CALL "C$SYSLOG"
    USING OP-CODE, parameters
```

Parameters

OP-CODE Constant

Indicates which C$SYSLOG operation to perform. The operations are described below in the Description section.

parameters vary depending on the op-code chosen

Provide information and hold results for the operation specified. These parameters are discussed with their corresponding op-codes in the Description section below.

Description

C$SYSLOG sends messages to syslog on UNIX systems that have the syslog function. On other UNIX systems, and on other non-Windows operating systems, this routine sends messages to the console. On Windows systems, the routine sends messages to the event log, which can be viewed using the Event Viewer applet available on Windows.

The runtime also sends messages to the system log when it detects broken files. This function allows COBOL programmers to notify system administrators automatically when a broken file is detected, instead of relying on individual users to report such errors. The runtime sends such notifications only if the COBOL program has opened the system log using the C$SYSLOG routine as described below.

Note: Because the various implementations of system logging don't report errors, the C$SYSLOG routine does not report errors either, because it never receives any.

See your UNIX system documentation for information about the syslog facility. Refer to your Windows documentation for information on the Event Viewer applet.

The following constants used by the C$SYSLOG library routine are defined in “acucobol.def”:

CSYSLOG-OPEN (op-code 0) opens the system log. This operation takes the following two parameters and has no return value:

DOMAIN Alphanumeric item

The UNC name of a Windows machine to which to send events. This parameter is used only for Windows machines. It is ignored on other operating systems.

If this parameter is set to any of the following items, the local machine executing the runtime receives the event messages:

- the NULL keyword
- an empty string literal (this setting generates a compiler warning)
- a string literal that contains spaces
- a data item with a value of spaces or low-values

APPNAME Alphanumeric item

The name of the application under which to log the notifications. See the Event Viewer or syslog documentation for information about what this parameter actually does and how to filter notifications on this name.

CSYSLOG-WRITE (op-code 1) allows you to write to the system log. This operation takes the following two parameters:

PRIORITY Numeric item

This parameter may have one of the following values:

CSYSLOG-PRIORITY-SUCCESS (value 0)

CSYSLOG-PRIORITY-INFORMATION (value 1)

CSYSLOG-PRIORITY-WARNING (value 2)

CSYSLOG-PRIORITY-ERROR (value 3)

See the Event Viewer or syslog documentation that comes with your operating system for information about these values and how to filter based on them.

MESSAGE Alphanumeric item

The message sent to the system log. This parameter has no maximum length; it is sent to the system log as is. The system administrator sees this message in the system log.

CSYSLOG-CLOSE (op-code 2) closes the system log. This operation takes no parameters and returns no value. After the system log is closed and before it is opened again, all writes to the system log (including those done internally by the runtime for reporting broken files) fail with no warning or error message.

C$SYSTEM

This routine combines the functionality of “SYSTEM” and “C$RUN”. It allows you to run other programs from inside a COBOL application in a variety of ways.

C$SYSTEM adds the following capabilities to the original capabilities of SYSTEM and C$RUN:

1. Uniform programming interface for all options
2. Asynchronous operation (C$RUN) added to UNIX hosts
3. Windows hosts can specify minimized, maximized, or hidden windows
4. Smart shell selection for Windows and Windows NT

Usage

```
CALL "C$SYSTEM"
    USING CMD-LINE, FLAGS
    GIVING EXIT-STATUS
```

C$SYSTEM can be used to execute a program in Thin Client environments. To execute a program on the display host in a thin client environment, add the prefix "@[DISPLAY]:" to the name of any program that resides on the client machine. For Example:

```
C$SYSTEM "@[DISPLAY]:C:\notepad myfile.txt"
```

Parameters

CMD-LINE PIC X(n)

Contains the operating system command line to execute.

FLAGS Numeric unsigned (optional)

Supplies the options for the operation. If omitted, acts as if “0” was specified. You can find possible values of FLAGS in “acucobol.def”.

EXIT-STATUS Any numeric data item

Returns the called program’s exit status.

FLAGS

The FLAGS field specifies various options about how the command should be run. Determine the value of the FLAGS field by adding together the values corresponding to the following options:

CSYS-ASYNC (value 1): This option causes the command to run independently of the COBOL program. After starting the command, the COBOL program continues. When this option is specified, EXIT-STATUS returns undefined results. When this flag is not used, the COBOL program waits for the command executed to finish before the COBOL program continues. CSYS-ASYNC is functional only on Windows and UNIX systems.

Note: On UNIX machines, specifying CSYS-ASYNC with a program that tries to do input or output to the terminal is not supported.

CSYS-NO-IO (value 2): For character-based systems, the runtime normally sets the terminal to its default state prior to running the command, and resets it back to the state needed by the runtime when the command finishes. This option ensures that the called application runs correctly if the application uses the screen. However, CSYS-NO-IO also causes the runtime to "forget" the contents of the screen. This happens because the command executed may display information on the screen that ACUCOBOL-GT is not aware of. Because of this, windows created after a call to C$SYSTEM may not correctly restore the screen contents when these windows are closed. You can avoid this problem by re-initializing the screen after C$SYSTEM returns. You can do this by erasing the screen or closing a floating or pop-up window that covers the entire screen (the window must have been created by the C$SYSTEM call).

If the command to be executed will not perform any screen I/O, then you can request that C$SYSTEM retain ACUCOBOL-GT's memory of the original screen by using the CSYS-NO-IO option. This will avoid the problem described above. The option has no effect in Windows, where the command runs in its own window.

CSYS-MAXIMIZED (value 4): This option causes the command to run in a maximized window. This is functional only when you are running under Windows.

CSYS-MINIMIZED (value 8): This option causes the command to run in a minimized window. In addition, the COBOL program remains the active program retaining the keyboard focus and keeping the active appearance. This is functional only when you are running under Windows.

CSYS-COMPATIBILITY (value 16): This option causes the command to run in a window that is compatible with the way the SYSTEM library routine works. Use this option if you want to modify a call to SYSTEM and change this call to C$SYSTEM. There are very few differences between the default behavior of SYSTEM and C$SYSTEM, so this option is rarely needed. The only known difference involves the Microsoft Word application. If you use SYSTEM to start Microsoft Word, it always starts in a "normal" sized window, that is, the window size suggested by Windows. If you use C$SYSTEM to start Microsoft Word (with no FLAGS specified), then Word adopts the last window size it previously used. Supplying a flag of CSYS-COMPATIBILITY causes C$SYSTEM to behave the same as SYSTEM. Of course, if you prefer the behavior of C$SYSTEM, the flag should not be used. In comparison with SYSTEM, C$SYSTEM generally conforms more closely to the way Windows itself launches programs. The CSYS-COMPATIBILITY flag is recommended only if you change a SYSTEM call to a C$SYSTEM call and you observe a difference you do not like.

CSYS-HIDDEN (value 32): This option runs the command in a hidden window. Note that some applications, particularly those that routinely interact with the user, may get confused if you "hide" the command. This works well, however, for executing system tasks that do not have a user interface, such as executing a batch file that renames a series of files. This option is functional only when you are running under Windows.

CSYS-SHELL (value 64): When this option is specified, C$SYSTEM uses the host's command-line processor (the host's shell) to execute the command. Otherwise, the command may be executed without the command-line processor. This option affects only Windows (non-Windows versions always use the host's shell). For Windows applications that create their own windows, you should avoid using the shell - the application will not receive the initial window size request specified in FLAGS. For ".COM" and ".BAT" programs, and other built-in shell commands such as COPY and DIR, you *must* use the shell or the command may not execute.

The effect of this option is to prefix the command with the value of the COMSPEC environment variable and "/C". Under Windows, this will usually result in a prefix like "C:\COMMAND.COM /C". Under Windows NT, the prefix will typically be "CMD.EXE /C".

CSYS-DESKTOP (value 128): This option is for applications running in the thin client environment. It indicates that the application wants to run the command on the client system rather than the application server. When the command executes, unless the CSYS-ASYNC option is also specified, the thin client appears to "hang" while the application waits for the command's termination status. This behavior can be avoided with the CSYS-ASYNC flag. The CSYS-ASYNC flag causes the command to be run asynchronously.

If CSYS-DESKTOP is specified but the calling program is not running under thin client, the flag is ignored and the command is run on the same machine as the calling application.

CSYS-INHERIT-HANDLES (value 256): This option causes the new process to inherit each inheritable handle owned by the calling process. This includes "stdin", "stdout", "stderr", and other file handles that the calling process has open.

Note that because the called process inherits many open files, it is vulnerable to running out of file handles.

This option is needed when an Alternate Terminal Manager runtime calls C$SYSTEM to run a batch program which in turn calls another Alternate Terminal Manager runtime. Without this option, the called program will not display any output to the screen.

Comments

The "C$SYSTEM" routine submits CMD-LINE to the host operating system as if it were a command keyed in from the terminal. The maximum allowable length for the command line is 1024 bytes.

Note: Applications that run under Windows but that do not create their own windows should use the CSYS-SHELL flag to execute ".COM", ".BAT", and built-in shell commands such as COPY and DIR. See the description of **CSYS-SHELL**flag.

You should specify only one window size flag (CSYS-MAXIMIZED, CSYS-MINIMIZED, CSYS-COMPATIBILITY, or CSYS-HIDDEN). In the absence of any window size flag, the command runs in a "normal" window whose size is determined by the operating system. Windows programs can set their own window size. This will override the window size suggested by FLAGS. Essentially, the value of FLAGS is only a "suggestion" to the application.

Options that are not meaningful to the host system are ignored. Meaningful options in the same FLAGS setting are still applied.

The status of a call to C$SYSTEM is placed in EXIT-STATUS. This is usually the exit status of the executed program, or is "-1" if C$SYSTEM failed. Note that Windows will return "-1" from commands that are built into COMMAND.COM because COMMAND.COM does not return an exit status for built-in functions.

C$TOUPPER and C$TOLOWER

These routines translate text to upper- or lower-case.

Usage

```
CALL "C$TOUPPER"
    USING TEXT-DATA, VALUE TEXT-LEN

CALL "C$TOLOWER"
    USING TEXT-DATA, VALUE TEXT-LEN
```

Parameters

TEXT-DATA PIC X(n)

Contains the data to translate to upper- or lower-case.

TEXT-LEN USAGE UNSIGNED-INT, or a numeric literal

Contains the number of characters to translate.

Description

C$TOUPPER translates the first TEXT-LEN characters in TEXT-DATA to upper-case. C$TOLOWER translates them to lower-case. No size checking is done on TEXT-DATA, so you must ensure that TEXT-LEN has a valid value. VALUE must be included in the calling statement. If it is omitted, the program will very likely encounter memory errors. These routines only translate characters with a numeric value of 0-128. Anything above that (such as é, with a value of 130) must be mapped to its associated upper- or lower-case character using the configuration variable UPPER-LOWER-MAP.

Note: You can also translate character strings using the intrinsic functions UPPER-CASE and LOWER-CASE, the CONVERTING option of the INSPECT statement, or the UPPER or LOWER options of the ACCEPT statement.

C$XML

This routine lets you retrieve and parse precise information from an XML document. It also lets you add, modify, or delete data in an XML document. Refer to the *Guide to Interoperating with ACUCOBOL-GT*, Chapter 11, section 11.2.6 for additional user instructions, examples, and help with XML terminology like *element*, *attribute*, *parent*, *child*, and *sibling*.

Usage

```
CALL "C$XML"
    USING OP-CODE, parameters...
```

Parameters

OP-CODE Numeric parameter

Specifies the operation to perform. These are defined in the description below. There are constants defined for the op-codes in “acucobol.def”.

The parameters vary depending on the operation selected. They provide information and hold results for the operations specified. These are described below.

Description

Unless otherwise noted, the return code is “0” if an error has occurred. The return code is positive if everything went correctly. Possible operations include:

CXML-PARSE-FILE (op-code 1)

Parses the specified file, returning a parser handle as the return-code. This operation takes one parameter:

filename name of the XML document to parse

To read a file from the Internet and parse it, you can pass the filename with URL syntax. For example, you could pass the following filename:

```
http://myserver.mycomp.com/xmldata/bookfile.xml
```

Use this op-code when the file is fairly small, and you want to read and parse the whole file with a single call and store it in memory.

CXML-RELEASE-PARSER (op-code 2)

Releases memory allocated by parsing. The passed handle is zeroed by this call. This operation takes one parameter:

handle the parser handle returned by CXML-PARSE-FILE, CXML-OPEN-FILE, CXML-PARSE-STRING, or CXML-NEW-PARSER

Caution: Failing to call with this op-code results in a memory leak.

CXML-GET-FIRST-CHILD (op-code 3)

Retrieves the handle of the first child element of the handle passed. The return-code is a handle of the first child element, or "0" if there are no children. (The case of no children is not considered an error, even though CXML-GET-LAST-ERROR will return "10".) This operation takes one parameter:

handle	a parser handle or an element handle. If a parser handle, returns the first child of the top-level element.

CXML-GET-NEXT-SIBLING (op-code 4)

Retrieves the handle of the next sibling element of the handle passed. The return-code is a handle of the next sibling element, or "0" if no more siblings. (The case of no next sibling is not considered an error, even though CXML-GET-LAST-ERROR will return "11".) This operation takes one parameter:

handle	an element handle

CXML-GET-PARENT (op-code 5)

Retrieves the handle of the parent element of the handle passed. This enables you to go through the XML tree without keeping track of all the handles received from C$XML. The return-code contains the handle of the parent element of that element, or "0" if this is the top-level element or some other error. This operation takes one parameter:

handle	an element handle

CXML-GET-DATA (op-code 6)

Retrieves the name and value of that element. This operation takes four parameters, the fourth parameter is optional:

handle	is an element handle or parser handle. If handle is a parser handle, it is first set to the handle of the top-level element of the parsed file.
item-name	is returned as the name of the element

item-value is returned as the CDATA of that element (the data outside of the XML tags)

item-value-length (optional) is set to the length of the data returned in item-value.

CXML-GET-ATTRIBUTE-COUNT (op-code 7)

Retrieves the number of attributes in that element. This operation takes one parameter.

handle an element handle or parser handle. If it is a parser handle, it is first set to the handle of the top-level element of the parsed file.

CXML-GET-ATTRIBUTE (op-code 8)

Retrieves the name and value of the attributes in that element. This operation takes five parameters, the fifth parameter is optional:

handle an element handle or parser handle. If it is a parser handle, it is first set to the handle of the top-level element of the parsed file.

attr-num the attribute to get (starting at 1)

attr-name the name of the attribute

attr-value the value of the attribute

attr-value-len (optional) the length of the attribute value (a numeric item)

If the return-code is “0”, it can mean that there are no elements, or that the passed handle is not a valid handle. Query on the last error with op-code 9 to determine which is the case.

CXML-GET-LAST-ERROR (op-code 9)

Any C$XML call that fails will generate an error code, both a numeric value and a string value that describes the error. If return-code from any other function is “0” or “-1”, call this operation to get the error. This function returns the numeric value of the last error. This function takes one parameter:

text-val the text value of the error code

These errors are listed as a level 78 data item in “acucobol.def”. The possible errors are:

Numeric value	**Text Value**	**Description**
1	CXML-NO-MEMORY	Unable to create parser due to low memory
2	CXML-EXPAT-ERROR	Unable to create parser - expat error
3	CXML-FILE-OPEN-ERROR	Unable to open named file
4	CXML-PARSE-ERROR	Invalid XML file or other parsing error
5	CXML-INVALID-PARSER-HANDLE	The passed handle is not a valid parser handle
6	CXML-INVALID-ELEMENT-HANDLE	The passed handle is not a valid element handle
7	CXML-INVALID-ATTRIBUTE-NUMBER	Invalid attribute number
8	CXML-URL-ERROR	The URL given could not be accessed
9	CXML-NOT-AVAILABLE	The XML parser is not available on this machine
10	CXML-NO-CHILDREN	The specified element has no children
11	CXML-NO-SIBLINGS	The specified element has no siblings
12	CXML-NO-PARENT	The specified element is a top-level element

Numeric value	Text Value	Description
13	CXML-NO-VALUE	The specified element has no value
14	CXML-NO-ATTRIBUTES	The specified element has no attributes
15	CXML-REGEXP-ERROR	The regular expression given caused an error
16	CXML-TOP-LEVEL	The specified parser already has a top-level element
17	CXML-INVALID-PROC-INSTR-NUMBER	The idx given for CXML-GET-PROC-INSTR is outside of the range of available processing instructions (i.e., is greater than the value returned by CXML-GET-PROC-INSTR-COUNT).
18	CXML-NO-PROCESSING-INSTRUCTIONS	There are no processing instructions for CXML-GET-PROC-INSTR-COUNT or CXML-GET-PROC-INSTR, or target was not used or is blank in CXML-SET-PROC-INSTR (i.e., you are removing a processing instruction).

CXML-OPEN-FILE (op-code 10)

Opens the named file with an XML parser, positioning it at the top-level element. (Use CXML-PARSE-NEXT-RECORD to get more data). The returned handle is a parser handle. This operation takes one parameter:

filename the name of the file to open with the parser

CXML-PARSE-STRING (op-code 11)

Parses the specified string as XML, returning a parser handle. You can parse strings passed by reference. Note that strings passed by reference MUST be terminated with low-values so that the runtime can determine the length of the string passed. Not terminating with low-values will result in undefined behavior.

This operation takes one parameter:

string the string or string reference to be parsed as XML

CXML-PARSE-NEXT-RECORD (op-code 12)

Parses the next record in the element of the specified handle. This operation takes one parameter:

handle is a parser handle returned by CXML-OPEN-FILE

No data is returned, but now more data is available for retrieval by other op-codes. Return-code is “0” on error (including end-of-file) or non-zero if everything worked.

CXML-GET-PREV-SIBLING (op-code 13)

Retrieves the previous sibling element of the handle passed. The return-code is a handle of the previous sibling element, or “0” if no previous siblings. (The case of no previous sibling is not considered an error, even though CXML-GET-LAST-ERROR will return “11”.) This operation takes one parameter:

handle an element handle

CXML-NEW-PARSER (op-code 14)

Creates an empty XML document in memory and returns the parser handle. You must pass this to CXML-RELEASE-PARSER to free the data. This operation takes no parameters.

CXML-GET-ATTRIBUTE-BY-NAME (op-code 15)

Retrieves the named attribute from an element. This operation takes seven parameters. The last three, *attr-name*, *attr-value*, and *attr-len* are all optional.

handle	is an element handle
attr-regex-name	is a regular expression matching the name of the attribute returned. This regular expression must be null-terminated, or fewer than 1024 bytes.
attr-regex-flags	is a collection of flags to apply to the regular expression. The valid flags are listed in "acucobol.def" and described under **C$REGEXP op-code 2**. They can be added together to get more than one flag.
attr-idx	is one less than the starting index of attributes to search (use "0" to start at the first attribute). Is also the index of the attribute actually returned if successful. This returned attribute number can be used in the next call to GET-ATTRIBUTE-BY-NAME to continue searching where you left off.
attr-name	(optional) is the name of the actual attribute returned
attr-value	(optional) is the value of that attribute
attr-len	(optional) is the length of the attribute value

If desired, *attr-name*, *attr-value*, and *attr-len* can be retrieved by calling CXML-GET-ATTRIBUTE (op-code 8) with the index returned from this call. Not getting these values can be useful if all you want to do is test for the existence of a particular named attribute.

CXML-GET-CHILD-BY-NAME (op-code 16)

Retrieves the named child element. This operation takes three parameters:

handle	is an element handle
regex	is a regular expression of the child to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.
flags	is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-CHILD-BY-CDATA (op-code 17)

Retrieves the child element whose data matches the regular expression given. This operation takes three parameters:

handle	is an element handle
regex	is a regular expression of the child to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.
flags	is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-CHILD-BY-ATTR-NAME (op-code 18)

Retrieves the child element that has an attribute name that matches the regular expression given. This operation takes three parameters:

handle	is an element handle
regex	is a regular expression of the child to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-CHILD-BY-ATTR-VALUE (op-code 19)

Retrieves the child element that has an attribute value that matches the regular expression given. This operation takes three parameters:

handle is an element handle

regex is a regular expression of the child to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-SIBLING-BY-NAME (op-code 20)

Retrieves the named sibling element. This operation takes three parameters:

handle is an element handle

regex is a regular expression of the sibling to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-SIBLING-BY-CDATA (op-code 21)

Retrieves the sibling element whose data matches the regular expression given. This operation takes three parameters:

handle is an element handle

regex is a regular expression of the sibling to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-SIBLING-BY-ATTR-NAME (op-code 22)

Retrieves the sibling element that has an attribute name that matches the regular expression given. This operation takes three parameters:

handle is an element handle

regex is a regular expression of the sibling to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-SIBLING-BY-ATTR-VALUE (op-code 23)

Retrieves the sibling element that has an attribute value that matches the regular expression given. This operation takes three parameters:

handle is an element handle

regex is a regular expression of the sibling to locate. This regular expression must be null-terminated, or fewer than 1024 bytes.

flags is flags for the regular expression. See **C$REGEXP op-code 2** for a list of valid flags.

Return-code is the handle of the returned child, or "0" if no such child exists.

CXML-GET-COMMENT (op-code 24)

Retrieves the comment for the specified handle. This operation takes three parameters. The third is optional.

handle	is an element handle or a parser handle
data	is returned as the comment for the handle
len	(if provided) is the length of the comment

When comments are returned to the COBOL program, they are separated by a single byte of low-values as they are in the XML file. For example:

```
<!--  this is a multi-line comment
      for a single XML file.  -->
```

is returned to the COBOL program as a single string. But this comment:

```
<!--  this is two separate comments  -->
<!--  this is the second line of the separate comment -->
```

is returned to the COBOL program with a single byte of LOW-VALUES separating the separate lines and comment. The final line is terminated by two bytes of LOW-VALUES, so you know how many lines of comments are in the XML file.

If you pass the third optional *len* parameter to C$XML when getting comments, the length in bytes is returned in that parameter.

CXML-SET-DATA (op-code 25)

Sets data in the specified element. This operation takes three parameters. The *len* parameter is optional.

handle	a handle to the element where the data is to be set
data	is the new value of that element
len	is the length of data (the variable). If omitted, it defaults to the size of the variable.

Return code is “0” on error and “1” on success.

CXML-MODIFY-ATTRIBUTE-VALUE (op-code 26)

Modifies the specified attribute value of an element. This operation takes four parameters. The last one is optional.

handle	is an element handle
attr-num	is the attribute number whose value you want to modify
new-attr-value	is the new value (of length *new-attr-len*) of that attribute
new-attr-len	(optional) is the length of the new attribute value

If *new-attr-len* is omitted, it defaults to the length of the data element given for *new-attr-value*.

CXML-ADD-CHILD (op-code 27)

Adds a new child element to the list of children already in the specified element. The new element will be the last in the list. This operation takes four parameters:

handle	is an element handle
element-name	is the name of the new child element
element-data	(optional) is the value of the new child element. By default, this is blank.
data-len	(optional) is the length of the new child element. If omitted, it defaults to the size of the data item.

Return code is the new element handle, or “0” on error.

CXML-ADD-SIBLING (op-code 28)

Adds a new sibling element to the specified element. The new element will be the next element of this sibling. This operation takes four parameters:

handle	is an element handle

element-name	is the name of the new sibling element
element-data	(optional) is the value of the new sibling element. By default, this is blank.
data-len	(optional) is the length of the new sibling element. If omitted, it defaults to the size of the data element.

Return code is the new element handle, or "0" on error.

CXML-ADD-ATTRIBUTE (op-code 29)

Adds a new attribute to the specified element. This operation takes four parameters. The last one is optional:

handle	is an element handle
attr-name	is the new attribute name
attr-value	is the new attribute value, of length *attr-value-len* (if given) or the length of the data element given as the *attr-name*.
attr-value-len	(optional) length of the new attribute value

Return code is the index of the new attribute, or "0" on error.

CXML-ADD-COMMENT (op-code 30)

Adds a new comment to the specified element handle or parser handle. This operation takes three parameters. The last one is optional.

handle	is an element handle or a parser handle
data	is the comment to add to the handle
len	(if provided) is the length of the comment

CXML-APPEND-COMMENT (op-code 31)

Appends a comment to the specified element handle or parser handle. This operation takes three parameters. The last one is optional.

handle is an element handle or a parser handle

data is the comment to append to the handle

len (if provided) is the length of the comment

Note that comments are associated with the element that follows them, as in:

```
<!-- comment assoc with elem-1 -->
<elem-1>
data for elem1
</elem-1>
```

CXML-DELETE-ATTRIBUTE (op-code 32)

Deletes the specified attribute from an element. This operation takes two parameters:

handle is an element handle

attr-num is the index of the attribute to delete (starting at "1").

CXML-DELETE-ELEMENT (op-code 33)

Deletes the specified element. This operation takes one parameter:

handle is the handle of the element to delete

If this element has any children, all children are also deleted.

CXML-DELETE-COMMENT (op-code 34)

Deletes the comment from the specified element handle or parser handle. This operation takes one parameter:

handle is an element handle or a parser handle

There is no return code (errors are silent).

CXML-WRITE-FILE (op-code 35)

Writes the specified file to the data tree. This operation takes two parameters:

handle	is a parser handle
filename	is the file to which to write the data tree

CXML-GET-PROC-INSTR-COUNT (OP-CODE 36)

Sets the number of processing instructions to get, if any. This op-code takes a single parameter, a valid parser handle. The return-code is "0" on error (including that there are no processing instructions), or the number of processing instructions on success.

CXML-GET-PROC-INSTR (OP-CODE 37)

Retrieves the stylesheet and other processing instructions from the XML file. This op-code takes four parameters:

handle	is a valid parser handle
idx (pic 9(n))	is the number of the processing instruction to get (starting at "1")
target (pic x(n))	is where the processing instruction target is returned. This data is space-filled from what is read from the file.
data (pic x(n))	is where the processing instruction data is returned. This data is space-filled from what is read from the file.

Processing instructions are composed of a target and data, and there can be multiple instructions in a given XML file. The target is the first word of the instruction line, and the data is the rest of the line. For example, given the following XML snippet:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/css" href="svg_default.css"?>
<!-- testit.xml - generated by ACUCOBOL-GT v8.0.0 -->
<svg
```

there is a single processing instruction (the stylesheet line). The target is the first word (xml-stylesheet), and the data is the rest of the information, not including the question marks (?): type="text/css" href="svg_default.css"

CXML-SET-PROC-INSTR (OP-CODE 38)

Sets the processing instructions in the XML file being created. This op-code takes two or four parameters:

handle is a valid parser handle

idx (pic 9(n)) is the number of the processing instruction to set (starting at "1")

target (pic x(n)) is the processing instruction target. This variable will have trailing spaces removed before being written to the XML file.

data (pic x(n)) is the processing instruction data. This variable will have trailing spaces removed before being written to the XML file. If *target* is all spaces, or if *target* and *data* are omitted, then the processing instruction is deleted. In this case, any later processing instructions are moved. (If instruction 3 is deleted, then instruction 4 becomes instruction 3, and instruction 5 becomes instruction 4, and so on.)

CXML-GET-VERSION (OP-CODE 39)

Retrieves the version attribute from the XML file. This op-code takes two parameters:

handle is a valid parser handle

dest (pic x(n) for n >= 3) is the destination for the version value

CXML-SET-VERSION (OP-CODE 40)

Sets the version attribute in the XML file. This op-code takes two parameters:

handle is a valid parser handle

new-version (pic x(n)) is the new version value. Note that making this an invalid version usually makes the resulting XML file invalid. It is your responsibility to make sure *new-version* is valid. At this time, the only valid value is "1.0".

CXML-GET-ENCODING (OP-CODE 41)

Retrieves the encoding attribute from the XML file. This op-code takes two parameters:

handle is a valid parser handle

dest (pic x(n)) is the destination for the encoding value

CXML-SET-ENCODING (OP-CODE 42)

Sets the encoding attribute in the XML file. This op-code takes two parameters:

handle is a valid parser handle

*new-encoding (pic x(n))*is the new encoding value

While the ISO-8859-1 encoding is supported by C$XML, it always gives strings in UTF-8, regardless of the encoding of the XML file.

Starting with version 8.1.2, the data returned to the COBOL program will be in one of three (instead of one of two) encodings. If the encoding in the XML file is given as either US-ASCII, UTF-8, or ISO-8859-1, the data returned to the COBOL program will be encoded in that character set. The only other encoding supported by C$XML is UTF-16. If an XML file is parsed that uses UTF-16, the data will be returned to the COBOL program as UTF-8.

Note that modifying the encoding will most likely cause errors, unless you translate all the data of the XML file into the new encoding before writing it. In other words, changing the encoding attribute does NOT cause the entire XML file to be translated into the new encoding automatically. Caution should be used in this area.

CXML-GET-STANDALONE (OP-CODE 43)

Retrieves the stand-alone attribute from the XML file. This op-code takes two parameters:

handle is a valid parser handle

dest (pic x(n) for n >= 3) is the destination for the stand-alone value

This is returned as either “no”, “yes”, or all blanks (if the stand-alone attribute is not specified in the XML file).

CXML-SET-STANDALONE (OP-CODE 44)

Sets the stand-alone attribute in the XML file. This op-code takes two parameters:

handle is a valid parser handle

new-standalone (pic x(n)) is the new stand-alone value

Return-code can either be “no” or “yes”, which puts the corresponding value into the XML file. Any other value causes the stand-alone attribute to be omitted from the XML file.

CXML-GET-RAW-DOCTYPE-LEN (OP-CODE 45)

Retrieves the raw doctype attribute length from the XML file. This op-code takes a single parameter:

handle is a parser handle

Return-code is the length of the raw doctype, or “-1” on error.

CXML-GET-RAW-DOCTYPE (OP-CODE 46)

Retrieves the raw doctype attribute from the XML file. This op-code takes two parameters, and an optional parameter:

handle is a parser handle

doctype (pic x(n) OR usage pointer) is a data item to be filled in with the current raw doctype value. If it is shorter than the length of the current *doctype*, it will be truncated. If *doctype* is of type POINTER, then you should include *len* (a numeric data item) as well and set it to the number of bytes allocated for *doctype*.

Return-code is "0" on error, or "1" on success.

CXML-SET-RAW-DOCTYPE (OP-CODE 47)

Sets the raw doctype attribute in the XML file. This op-code takes two parameters:

handle is a parser handle

doctype (pic x(n)) is the new doctype. The old doctype (if any exists) is discarded. Trailing spaces are removed.

Comments

This utility gives you low-level control over the parsing of XML data. If you do not require such precise control or if you would prefer to use a transparent interface to map XML to COBOL, you can use AcuXML to parse the XML data instead. AcuXML is a file system interface that is designed to map XML data to COBOL based on eXtended File Descriptors (XFDs) created at compile time. (See the *Guide to Interoperating with ACUCOBOL-GT*, Chapter 11, section 11.2, for more information.)

The C$XML routine is useful for parsing all XML files, including non-records-based XML files such as XFDs or configuration files that you have formatted in XML. AcuXML is not able to parse XML documents such as these, because it quits after the close of the first top-level element, considering that to be the end of the file. AcuXML is useful strictly for records-based XML files.

DISPLAY_REG_*

The routines named DISPLAY_REG_* parallel the routines named REG_*. These routines allow you to access and work with the Windows registry. For more information, see the section titled "**Routines to Handle Dynamic Memory**" in this Appendix.

Error and Exit Procedures

The error and exit procedure facilities allow you to cause one or more error or exit procedures to be called automatically when a program generates a particular type of error or terminates normally. Support of error and exit procedures in ACUCOBOL-GT helps facilitate the porting of Micro Focus COBOL applications containing this feature to ACUCOBOL-GT.

You can specify one or more error procedures to be called automatically when a run unit terminates normally or generates any of certain runtime errors. See the **CBL_ERROR_PROC** Routine in this appendix for details.

You can specify one or more exit procedures to be called automatically when a run unit terminates normally. See the **CBL_EXIT_PROC** Routine in this appendix for details.

An exit procedure can retrieve information about the termination that induced it with the **CBL_GET_EXIT_INFO** routine. See the entry in this appendix for details.

The error and exit procedures discussed in this appendix are:

- **CBL_ERROR_PROC**
- **CBL_EXIT_PROC**
- **CBL_GET_EXIT_INFO**

HEX2ASCII

HEX2ASCII converts hexadecimal data from the non-displayable hexadecimal format to its ASCII equivalent. This routine is the inverse of the ASCII2HEX routine.

Usage

```
CALL "HEX2ASCII"
    USING ASCII-VALUE, HEX-VALUE
```

Parameters

ASCII-VALUE PIC X(2)

The ASCII representation of a unit of data.

HEX-VALUE PIC X(4)

The hexadecimal value.

When you are defining the parameters, use the exact field sizes specified in the calling conventions above, otherwise the runtime may terminate abnormally.

Any characters that are not valid hexadecimal digits are treated as the digit "0". Trailing spaces are removed from the input value before conversion.

I$IO

The I$IO routine provides an interface to the file handler. An operation code and some number of additional parameters (depending on the operation called) are passed to the routine. The return code is set automatically after the call. The external variable "F-ERRNO" is set according to any errors found. "F-ERRNO" may not be reset on entry to I$IO, and should be checked only if I$IO returns an error condition.

Usage

```
CALL "I$IO"
```

```
USING OP-CODE, parameters
```

Parameters

OP-CODE Numeric parameter

Specifies the file handling routine to be performed. This table shows which operation corresponds to each operation code. The operations are detailed in the description below:

Code	Operation
1	OPEN-FUNCTION
2	CLOSE-FUNCTION
3	MAKE-FUNCTION
4	INFO-FUNCTION
5	READ-FUNCTION
6	NEXT-FUNCTION
7	PREVIOUS-FUNCTION
8	START-FUNCTION
9	WRITE-FUNCTION
10	REWRITE-FUNCTION
11	DELETE-FUNCTION
12	UNLOCK-FUNCTION
13	REMOVE-FUNCTION
14	SYNC-FUNCTION
15	EXECUTE-FUNCTION
16	BEGIN-FUNCTION
17	COMMIT-FUNCTION
18	ROLLBACK-FUNCTION
19	RECOVER-FUNCTION
21	IN-TRANSACTION-FUNCTION

parameters Vary depending on the op-code chosen

The remaining parameters vary depending on the operation selected. They provide information and hold results for the operations specified. All parameters are passed *by reference*. Parameters may be omitted from those operations that do not require them. These parameters are detailed in the "Description" below.

Description

All parameters passed to I$IO are passed *by reference*. This applies even to parameters that are integer values in the corresponding file handling routines. All numeric parameters should be passed to I$IO as SIGNED-SHORT values. The I$IO routine provides any necessary addressing conversions. Note that a parameter must be in the correct format for its type. Parameters that are PIC X must be terminated by a LOW-VALUES character.

Except for the MAKE function, I$IO will automatically terminate any PIC X parameters with a LOW-VALUES byte for you. Also, you do not have to specify SYNC for level 01 or level 77 parameters because they are automatically synchronized by ACUCOBOL-GT.

The file "filesys.def" is a COBOL COPY file that contains many useful definitions for use with I$IO. It contains definitions for the I$IO codes along with the "F-ERRNO" error values and many useful pre-declared variables that are of the proper type and usage.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

Note: The runtime configuration variable FILE_PREFIX is ignored by the I$IO routine.

OP-CODES and PARAMETERS

OPEN-FUNCTION (op-code 1)

This routine opens an existing indexed file. If it is successful, the value in RETURN-CODE should be moved to a data item that is USAGE POINTER. This data item is passed as the open file handle to the other file handling routines. If it fails, RETURN-CODE is set to a NULL value. After the file is opened, the primary key is set as the current key of reference and is positioned at the beginning of the file.

The OPEN routine has three parameters, *name*, *mode* and *l_parms*.

Name is the name of the file to open. It must be null-terminated.

Mode is one of the following values (defined in "filesys.def"):

Finput	open for input only
Foutput	open for output only
Fio	open for input and output
Fextend	same as Foutput

This routine only opens already existing files. If the file does not exist, the routine fails, even when opening with mode "Foutput".

"Foutput" does *not* delete the current file (this is different from the OPEN OUTPUT verb in COBOL).

Mode may furthermore have one of the following flags added to it to indicate file locking options (defined in "filesys.def"):

Fread_lock	locks file against other updaters
Fwrite_lock	locks file against all others
Fmass_update	same as Fwrite_lock
Ftrans	extended locking rules for transaction management

Some file systems cannot support the "Fread_lock" option correctly. For these systems, the setting of the external variable "F_UPGRADE_RLOCK" determines what happens. If this variable is set to the default, then the

"Fread_lock" setting is treated as a normal open (no file locking). If this variable is non-zero, then the "Fread_lock" setting is treated as "Fwrite_lock" instead.

A few file systems do not support any form of file locking. If locking is requested on one of these file systems, the open proceeds as if file locking were not specified, but the external variable "F-ERRNO" is set to W_NO_SUPPORT. This is also returned for file systems that cannot support multiple record locks when "Fmulti_lock" is specified.

If "Fmass_update" is used, then the file system is also requested to emphasize speed of updates over file security.

Additionally, "Fmulti_lock" may be also added to *mode* to request that more than one record lock be maintained in the file by this process. If this option is not specified, then any I/O operation on the file will first release any currently locked record. This results in only one record lock being set in the file at any time. When this option is used, locked records are released only when the file is closed or when the UNLOCK routine is called.

L_parms points to a null-terminated string that describes the key structure for the file. The *l_parms* parameter is the same as the *l_parms* parameter passed when using the MAKE op-code. This parameter is a string that contains three comma-separated numbers. These values are (in order):

1 the maximum record size

2 the minimum record size

3 the number of keys for the file.

If the maximum record size does not match the minimum record size, then variable sized records are implied. This parameter is not used by all file systems, but is supplied for those file systems that cannot determine these values on their own.

Note: The *l_parms* parameter is always required, but it is ignored by the default file systems shipped with ACUCOBOL-GT (RMS for VAX/VMS machines and Vision for all others). It is required by other file systems, however, including Btrieve and C-ISAM. If you ever intend on using any of these file systems, you should ensure that the values passed in *l_params* are correct.

CLOSE-FUNCTION (op-code 2)

This routine closes an open file. It also removes currently held locks on the file. The CLOSE routine has only one parameter, *f*, a file handle returned by OPEN. For some file systems, it is possible that CLOSE will write additional records that had been previously buffered by the system. For this reason, it is possible that a "disk full" condition can occur.

MAKE-FUNCTION (op-code 3)

This routine is used to create a new indexed file. It will overwrite any existing file. This routine will not overwrite a file that is currently in use. If the file is in use, the error E_FILE_LOCKED will be returned. The MAKE routine has six parameters, *name*, *comment*, *p_parms*, *l_parms*, *keys*, and *trans*.

This routine does not automatically terminate its parameters with LOW-VALUES, you must insure that the parameters are correctly terminated yourself. If you do not wish to supply an alternate collating sequence in the "trans" parameter to MAKE, then simply omit the parameter. The I$IO routine does not allow you to specify a "NULL" parameter which is what is expected by the MAKE routine. Instead, by omitting the parameter, the I$IO routine will construct a "NULL" parameter for you.

The host file system may ignore any or all of the values specified by *comment*, *p_parms* and *trans*. If the host system cannot perform the requested operation, it is ignored (however, if the *trans* value is ignored, "F-ERRNO" is set to W_NO_SUPPORT). If the host system cannot support one of the values specified in *l_parms* or *keys*, then MAKE fails and RETURN-CODE is set to the error condition E_NO_SUPPORT. If any of the parameters are ill-formed or contain values outside of the legal range of values, E_PARAM_ERR is returned.

Name points to the name of the file. This must be null-terminated.

Comment may be NULL or may point to comment string that describes the file. This comment is stored in the file, but has no other functional use. The comment may be up to 30 characters long and must be null-terminated. Many host file systems cannot store comments. On these systems, this parameter is ignored.

P_parms points to a string that describes various physical characteristics of the file. This string consists of five numeric fields separated by commas and must be null-terminated. Each of these fields advises the file system of desired treatment for the physical storage of the file. Individual file systems may ignore some or all of these fields. Furthermore, *p_parms* may be NULL to imply "0" for each of the fields.

The "filesys.def" COPY file has a data item containing these fields.

The fields are as follows:

1. Blocking factor. This value is the number of disk sectors to store in a single file block. It may be zero to request the default blocking factor for the file system.

2. Pre-allocate amount. This is the number of file blocks (the size of which depends on the blocking factor above) to initially allocate to the file. The purpose of this is to allocate contiguous disk space for the file. It does not limit the file's total size. This value may be zero to request the default initial allocation amount.

3. Extension factor. This is the number of file blocks to add to the file when it needs to grow. The purpose of this is to help reduce disk fragmentation. This value may be set to zero to request the default extension amount.

4. Compression factor. This is a value between zero and 100. It represents the amount of file compression desired. A value of zero indicates no compression and a value of 100 indicates maximum compression. Values in between indicate varying degrees of compression. The exact meaning of this depends on the host file system.

5. Encryption flag. If this value is “1”, then encryption is desired for the disk file. If it is zero, then no encryption is desired. The results of encryption depend on the host file system. Since no key is supplied, the results of encryption are usually only moderately secure.

l_parms points to a string that describes various logical characteristics of the file. The string consists of three numeric fields separated by commas. The string must be null-terminated.

The “filesys.def” COPY file has a data item containing these fields.

The fields are as follows:

1. Maximum record size. This is the size of the largest record to be placed in the file. For Vision 5 files, this may range from 1 to 67,108,864. For Vision 2, 3, and 4 files, this may range from 1 to 32,767.

2. Minimum record size. This is the size of the smallest record to be placed in the file. This may range from 1 to the maximum record size. If this field is the same as the maximum record size, then fixed-length records are implied.

3. Number of keys. This is the number of keys in the file, including the primary key. This may range from 1 to MAX_KEYS.

keys -- points to a null-terminated string that describes the key structure for the file. *keys* is a string of numbers separated by commas. The first key described is the primary key. It may not allow duplicate values. The primary key is called key “0”. The next key described is key “1” and so on. There should be as many keys described as the “number of keys” field of *l_parms* indicates.

The “filesys.def” COPY file has a data item containing these fields.

Each key description contains the following information:

1. Number of segments. This is the number of segments in this key. A segment is a contiguous region of bytes in the record. You may describe a split key by specifying more than one segment. With Vision Version 3 files, this may range from 1 to V3_MAX_SEGS. With Vision Version 4 and 5 files, this may range from 1 to MAX_SEGS.

2. Duplicates flag. If this value is “1”, then duplicate keys are allowed. If “0”, then duplicate values are not allowed. This must be “0” for the primary key.

3. Segment size. This is the number of bytes in the first segment. This may be between 1 and MAX_KEY_SIZE.

4. Segment offset. This is the offset from the beginning of the record to the first byte of the segment. A segment that starts at the beginning of the record has an offset of “0”.

5. Remaining segments. The segment size and segment offset fields are repeated for each additional segment in the key. The sum of the segment sizes may not exceed MAX_KEY_SIZE.

For example, a file with two keys, the first one having two segments (offset zero, length 10 and offset 50, length 5) and the second one with one segment (offset 20, length 15) and allowing duplicates would be written:

```
2,0,10,0,5,50,1,1,15,20
```

trans -- This parameter specifies an alternate collating sequence for the keys. If this parameter is NULL, then keys are ordered in ascending sequence based on their native unsigned value. If *trans* is not NULL, it must point to a 256 byte region of memory. Unlike other strings, this need not be null-terminated and is likely to contain nulls within it. This 256 byte region is used as a translation table on the bytes of each key to arrive at a new key-ordering. Each byte is used as an index into this table, and the resulting value is used to order the keys. Some host file systems cannot do key translations. On these systems, this parameter is ignored.

INFO-FUNCTION (op-code 4)

This routine returns a variety of information about the open indexed file *f*, depending on the value of *mode*. The information is returned in the area pointed to by *result*. Most of the *modes* return one or more numeric values. These values are represented in *result* as fixed size fields with the numbers represented as strings. When more than one value is returned in *result*, the values are separated by commas.

The “filesys.def” COPY file contains layouts for each kind of information that can be retrieved with this routine.

The INFO routine has three parameters, *f*, *mode*, and *result*.

F is a file handle returned by OPEN.

Mode determines what ***result*** is returned with a series of comma-separated numbers that define the format of *result*. (Note that in the following descriptions, the first value is number "1", the second is number "2" and so on. The number of times the field number is repeated represents the size of the field. For example "111,22" indicates that two values are returned, the first one is three digits long and the second one is two digits long.)

When a particular *mode* cannot be determined by the file system, then the error value E_NO_SUPPORT is set. If a particular value of mode "-1" or mode "-2" cannot be determined, it is set to zero.

The following *modes* are supported:

-1 This returns the same information as the *l_parms* parameter of the MAKE function. *Result* is in the format of "11111,22222,333" where:

1 = maximum record size

2 = minimum record size

3 = number of keys

-2 Returns the same information as the *p_parms* parameter of the MAKE function. *Result* is in the format of "11,22222,33,444,5" where:

1 = blocking factor

2 = pre-allocation amount

3 = extension factor

4 = compression factor

5 = encrypted flag

-3 Returns the comment specified to the MAKE function that made the file. The format is a null-terminated string of up to thirty characters.

-4 Returns the number of records in the file. This is returned as a 10-digit number.

-5 Returns the 256-byte key translation table specified to MAKE when the file was originally made. If no key translation table was specified, then the E_NO_SUPPORT error is set. In this case, this should be simply taken to mean that the native key ordering was used.

-6 Returns the number of currently locked records for the file handle passed to I$IO.

-7 For Vision Version 4 and 5 files, returns the number of data and index segments in the form: “11111,22222” where

1 = number of data segments

2 = number of index segments

For Vision Version 2 and 3 files, this mode always returns “00000,00000”.

-8 Returns the name and size of a Vision Version 4 or 5 file segment.

This mode is different from the other modes in that it uses the third argument as both INPUT and OUTPUT. Set FS-TYPE to FS-DATA (255) or FS-INDEX (254) and FILE-SEGMENT-NUMBER to the number of the segment you want information about. Upon return, name will contain the filename of the segment and FS-SIZE will contain the size of the segment. The FILE-SEGMENT-INFO data item in filesys.def describes this structure.

This operation can be called with a Vision Version 2 or 3 file, also. In this case, type is ignored, but seg must be 1. The name and size returned will be those of the single Version 2 or 3 file.

-9 Returns the sum of the sizes of all the segments that make up a Vision Version 4 or 5 file. The return value is 15 digits long. If called with a Vision Version 2 or 3 file, this operation returns the size of the single Version 2 or 3 file.

-10 Returns the version number of the Vision file in a three character string. For example, if the file is Vision Version 5, “005” is returned.

0+ A *mode* of zero or greater indicates that information about a particular key is desired. That key information is returned as “11,2,333,44444” where

1 = number of segments in key

2 = “1” if duplicates are allowed

3 = size of first segment

4 = byte offset of first segment

The third and fourth fields are repeated for each additional segment in the key.

Note: Versions of INFO prior to the version used by the 3.2 runtime returned only one digit for the number of segments in the key.

READ-FUNCTION (op-code 5)

This routine reads a record out of an indexed file. The READ routine has three parameters, *f, record*, and *keynum*.

F must be a valid file handle returned by OPEN.

Record points to the area to hold the record read.

Keynum is the key number to read from. Key "0" is the primary key, key "1" is the first alternate and so on. The bytes in *record* corresponding to the key *keynum* must contain the key value of the record to be read. If this routine succeeds, RETURN-CODE is set to the size of the record read. RETURN-CODE is set to zero on failure.

Records read by a file open for input only are not locked. Furthermore, most file systems do not block the reading of locked records by a file open for input (Note that this feature depends on the host file system - not all can support it). Records read from a file open for I/O are automatically locked unless the external variable "f-no-lock" is set to a non-zero value first in which case they are treated in the same manner as files open for input.

If the key *keynum* allows duplicates and the next record in the file contains the same key value as the record read, the variable "F-ERRNO" is set to W-DUP-OK. This feature is not supported by many host file systems.

A successful READ causes the current key of reference to be set to *keynum* and the file position is set to the record read. This is used by the NEXT and PREVIOUS routines. If READ is unsuccessful, then the current key of reference is set to an undefined state.

NEXT-FUNCTION (op-code 6)

This routine reads the next record in the sequence of records specified by the current key of reference. When a file is first opened, its key of reference is the primary key and the file is positioned so that the NEXT record is the first record in the file. The READ and START routines can change the current key of reference. The NEXT routine has two parameters, *f*, and *record*. *F* must be a valid file handle returned by OPEN. *Record* points to the area to hold the record read.

If this routine succeeds, RETURN-CODE is set to the size of the record read. If it fails, RETURN-CODE is set to zero and sets the current key of reference to the "undefined" state. However, if it fails due to the record being locked, the current key of reference is unchanged and the file pointer is set to the locked record (some file systems do not support this rule). Also, if it fails because it reached the end of the file, then the current key of reference is left unchanged and the file pointer is positioned so that a call to PREVIOUS returns the last record in the file.

The NEXT routine follows the same record locking rules as the READ routine.

If the record following the one read contains the same key value (in the current key of reference), then "F-ERRNO" is set to W-DUP-OK.

PREVIOUS-FUNCTION (op-code 7)

This routine behaves just like the NEXT routine with two exceptions. The first is that RETURN-CODE is set to the preceding record in the file instead of the next one. The second is that it never sets the W_DUP_OK error value. The PREVIOUS routine has two parameters, *f*, and *record*. *F* must be a valid file handle returned by OPEN. *Record* points to the area to hold the record read.

Some file systems do not support the ability to read a file backwards. For these systems, this routine sets the error value E_NO_SUPPORT.

This routine treats the beginning of the file in a manner that is analogous to the way NEXT treats the end of the file. If an PREVIOUS call reaches the beginning of the file, a call to NEXT returns the first record of the file.

START-FUNCTION (op-code 8)

This routine selects the current key of reference and positions the file pointer for the next NEXT or PREVIOUS routine. The START routine has five parameters, *f*, *record*, *keynum*, *keysize* and *mode*.

F must be a valid file handle returned by OPEN.

Record points to the area to hold the record read.

Keynum selects which key to use. The corresponding key area in *record* must contain the key value that will be used to position the file.

Keysize indicates the size of the key. If *keysize* is zero, the entire key is used. Otherwise, only the first *keysize* bytes of the key will be used.

Mode selects how the file is to be positioned with respect to the key value defined in *record*. It can be one of the following values:

F_EQUALS	The file is positioned at the record that matches the key value
F_NOT_LESS	The file is positioned at the record that matches the key value, or the next greater one if no one matches
F_GREATER	The file is positioned at the first record greater than the key value specified
F_LESS	The file is positioned at the last record smaller than the key value specified
F_NOT_GREATER	The file is positioned at the record that matches the key value, or the last record smaller than the key value if no one matches

The F_EQUALS mode is usually used to test for the existence of a record or to position a file when the key value is known. The F_NOT_LESS and F_GREATER modes are used to position the file for a series of NEXT calls and the F_LESS and F_NOT_GREATER modes are used to prepare for a series of PREVIOUS calls.

After a successful START, the current key of reference will set to *keynum*. The next NEXT or PREVIOUS call will return the record selected by the START routine. Note that in this case, NEXT and PREVIOUS both return the same record.

If the START routine fails, then the current key of reference is placed in the "undefined" state.

Some file systems cannot perform the F_LESS or F_NOT_GREATER modes. On these file systems, specifying these modes causes START to return an error and set the E_NO_SUPPORT condition.

WRITE-FUNCTION (op-code 9)

This routine adds a new record to the passed file. The WRITE routine has three parameters, *f*, *record*, and *size*.

F must be a valid file handle returned by OPEN.

Record points to the record to add.

Size is the size of the record. If *size* is zero, then the maximum record size for the file is used.

If the file has keys that allow duplicate values, and one or more of those keys in *record* match keys that already exist in the file, then "F-ERRNO" is set to W-DUP-OK. Several file systems cannot detect this condition and in this case, "F-ERRNO" is set to zero instead.

For keys that allow duplicates, the new record is added such that its keys are placed at the end of the sequence of those keys with the same value. Many file systems do not support this rule, in which case the ordering is defined by the file system.

The WRITE routine does not change the current file position or affect the current key of reference.

REWRITE-FUNCTION (op-code 10)

This routine replaces an existing record in the file. The REWRITE routine has three parameters, *f*, *record*, and *size*.

F must be a valid file handle returned by OPEN.

Record points to the new record to place in the file.

Size may be zero to indicate the maximum record size for the file. The record replaced is specified by the primary key value found in *record*. The size of the new record need not match the size of the existing record.

For keys that are unchanged between the new record and the old record, the ordering of those keys is unchanged. For keys that have changed, the new keys are placed at the end of the sequence of keys that have the same value. If any of the changed keys match keys that already exist in the file (and these keys allow duplicates), then "F-ERRNO" is set to W-DUP-OK. Note that these rules depend on the host system and may be different for some file systems.

The REWRITE routine does not affect the file position or the current key of reference.

DELETE-FUNCTION (op-code 11)

This routine deletes the record identified by the value found in the primary key area of *record*. It does not affect the current file position or key of reference. The DELETE routine has two parameters, *f*, and *record*. *F* must be a valid file handle returned by OPEN. *Record* points to the area to hold the record read.

UNLOCK-FUNCTION (op-code 12)

This routine unlocks any locked records held by the current process in the passed file. It is not an error to call this routine when there are no locked records. This routine does not affect the current file position or key of reference. This routine will not unlock any records if it is called during a transaction. "Commit" (see below) should be used instead. The "unlock" routine has only one parameter, *f*. *F* must be a valid file handle returned by OPEN.

REMOVE-FUNCTION (op-code 13)

This routine should remove from disk the indexed file indicated by *name*. This routine is specialized for indexed files because some host systems implement indexed files in more than one physical file. This routine should be called only when the file to be removed is closed. It has only one parameter, *name*. *Name* points to the name of the file. It must be NULL terminated.

SYNC-FUNCTION (op-code 14)

This routine causes all file buffers to be flushed to disk. The SYNC routine has only one parameter, *all_files*, which is a bit field. If

```
(all_files & FA-MASS-UPDATE)
```

is non-zero, then MASS-UPDATE files should be synced. If

```
(all_files & FA-REMOTE)
```

is non-zero, then remote files should be synced.

The constants FA-MASS-UPDATE and FA-REMOTE are defined in ”filesys.def”.

The exact effect of this routine depends on the host file system. It is entirely possible that this routine will do nothing under a particular system. Because of the highly host-dependent nature of this routine, no error reporting is done.

EXECUTE-FUNCTION (op-code 15)

This routine executes a file-system specific command. The EXECUTE routine has two parameters, *system*, and *command*.

System points to the name of the file system (e.g. “vision”, “btrieve”, etc.).

Command contains a command to be executed by that file system. The list of available commands is completely file-system specific. Many file systems do not have any commands that can be executed in this manner. The routine returns 0 if successful, otherwise, RETURN-CODE is set to a file-system specific error value. For file systems that do not support any commands, “-1” is always returned.

Note: Different file systems process this routine differently. For example, the interface for Microsoft SQL Server has some unique parameters. See the documentation for the specific file system interface for exceptions to this operation.

BEGIN-FUNCTION (op-code 16)

The BEGIN routine initiates a transaction. On indexed file systems this usually opens the log file for appending the first time it is called (if the log file doesn't exist, it is created). This routine has no parameters.

COMMIT-FUNCTION (op-code 17)

This routine commits all changes and releases all locks. It also ends a transaction. This routine has no parameters.

ROLLBACK-FUNCTION (op-code 18)

This routine rolls back all files affected to the state that they were in after the last completed transaction. This routine has no parameters.

RECOVER-FUNCTION (op-code 19)

This routine rolls forward all files affected to the state that they were in after the last completed transaction. This routine has no parameters.

IN-TRANSACTION-FUNCTION (op-code 21)

This routine returns a value indicating whether or not the program is currently in an unfinished transaction. The return value is "1" if there is current and unfinished transaction, "0" otherwise. This routine has no parameters.

Example

The following program opens a file, determines the number of records in the file, and then reads each of those records, printing out the size of each record found.

This example makes heavy use of variables found in “filesys.def”.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  EXAMPLE.

DATA DIVISION.

WORKING-STORAGE SECTION.

* Assume that we already know the maximum record size, minimum
* record size and the number of keys in the file.

78 MAX-SIZE    VALUE 1000.
78 MIN-SIZE    VALUE 100.
78 KEY-COUNT   VALUE 1.

COPY "filesys.def".

77  FILE-HANDLE               USAGE POINTER.
01  RECORD-AREA.
    03 OCCURS MAX-SIZE TIMES  PIC X.

PROCEDURE DIVISION.
MAIN-LOGIC.
* Open the file
    SET OPEN-FUNCTION TO TRUE.
    MOVE MAX-SIZE TO MAX-REC-SIZE.
    MOVE MIN-SIZE TO MIN-REC-SIZE.
    MOVE KEY-COUNT TO NUM-KEYS.
    MOVE Finput TO OPEN-MODE.
    CALL "I$IO" USING IO-FUNCTION, "MYFILE", OPEN-MODE,
                LOGICAL-INFO.
    IF RETURN-CODE = ZERO
        DISPLAY "Could not open file, error code = ", F-ERRNO,
                CONVERT
        STOP RUN.
    MOVE RETURN-CODE TO FILE-HANDLE.
* Now get the record count
    SET INFO-FUNCTION TO TRUE.
    SET GET-RECORD-COUNT TO TRUE.
    CALL "I$IO" USING IO-FUNCTION, FILE-HANDLE, INFO-MODE,
                RECORD-COUNT-INFO.
    IF E-NO-SUPPORT
        DISPLAY "File system cannot determine record count"
        PERFORM CLOSE-FILE
```

```
        STOP RUN.
* Read each record
    SET NEXT-FUNCTION TO TRUE
    PERFORM NUMBER-OF-RECORDS TIMES
       CALL "I$IO" USING IO-FUNCTION, FILE-HANDLE, RECORD-AREA
        IF RETURN-CODE = ZERO
        DISPLAY "Error reading record, code = ", F-ERRNO,
                 CONVERT
        PERFORM CLOSE-FILE
        STOP RUN
        ELSE
        DISPLAY "Record size = ", RETURN-CODE, CONVERT, LEFT
        END-IF
    END-PERFORM.
* All done
    PERFORM CLOSE-FILE.
    STOP RUN.

CLOSE-FILE.
    SET CLOSE-FUNCTION TO TRUE.
    CALL "I$IO" USING IO-FUNCTION, FILE-HANDLE.
```

LIB$GET_SYMBOL

The LIB$GET_SYMBOL routine retrieves a symbol's current value. *This routine is VMS-specific and should never be used on other machines.*

Usage

```
CALL "LIB$GET_SYMBOL"
    USING SYM-NAME, SYM-VALUE, SYM-SIZE, SYM-LOCATION
```

Parameters

SYM-NAME PIC X(n)

Contains the name of the symbol to retrieve. The local symbol table is searched first, followed by the global table.

SYM-VALUE PIC X(n)

If a value is found, it is returned in the second parameter.

SYM-SIZE PIC 9(n) USAGE COMP-1 (optional)

The third parameter is optional. It is filled in with the number of characters contained in the returned value.

SYM-LOCATION PIC X or PIC 9 (optional)

The final parameter is also optional. It is filled in with a "1" if the value is found in the local symbol table, a "2" if found in the global table.

This routine does not report any error conditions. See the VMS *System Routines* manual for details on VMS-specific routines.

LIB$SET_SYMBOL

This routine sets the value of a symbol belonging to the current command processor. *This library routine is VMS-specific and should never be used on other machines.*

Usage

```
CALL "LIB$SET_SYMBOL"
    USING SYM-NAME, SYM-VALUE, SYM-LOCATION
```

Parameters

SYM-NAME PIC X(n) USAGE DISPLAY

The first parameter is the symbol's name. This name has trailing spaces removed and is converted to upper case.

SYM-VALUE PIC X(n) or PIC 9(n) USAGE DISPLAY

The second parameter is the value to assign to the symbol. This is not converted to upper case, but does have trailing spaces removed.

SYM-LOCATION PIC X or PIC 9 USAGE DISPLAY (optional)

The third parameter is optional. It should be a "1" to set the symbol in the local symbol table. It should be a "2" to use the global symbol table. If this parameter is omitted, the local table is used.

This routine does not report any error conditions. See the VMS *System Routines* manual for details on VMS-specific routines.

Comments

It is possible to set a symbol's value from ACUCOBOL-GT using the SYSTEM library routine, but when you do this, the symbol's value immediately resets to its previous value as soon as the SYSTEM call is complete. This is because the SYSTEM routine starts another command processor which maintains its own set of symbols.

Routines to Handle Dynamic Memory

The next six routines, and the C$MEMCPY routine, allow you to create and manage dynamically allocated memory. This can be useful when you have to manipulate a number of items, but don't know how many items beforehand. Dynamic memory is allocated using the same mechanism that the runtime uses to allocate memory to programs. In the debugger, you can see how much dynamic memory is currently allocated to the program by running the program in the debugger. To display all memory allocated to the program select the Memory Usage menu item from the File menu (or press "U" on the keyboard). The amount of dynamic memory is the number that appears in the fifth position of the value displayed (see section 3.1.3 in Book 1, *ACUCOBOL-GT User's Guide*).

M$ALLOC (Dynamic Memory Routine)

M$ALLOC allocates a new area of dynamic memory.

Usage

```
CALL "M$ALLOC"
    USING ITEM-SIZE, MEM-ADDRESS
```

Parameters

ITEM-SIZE Numeric parameter

This indicates the number of bytes to allocate. This must be greater than zero.

MEM-ADDRESS USAGE POINTER

This holds the return value, either the address of the allocated memory or NULL if the allocation fails.

Comments

The maximum amount of memory you may allocate in one call depends on the host machine, but is at least 65260 bytes for all machines (providing that much memory is available). M$ALLOC adds some overhead to each memory block allocated. This ranges between 4 and 16 bytes depending on the machine architecture. Also, each operating system will typically add its own overhead. The debugger's "U" command reports the amount of memory you have currently allocated via M$ALLOC. The overhead added by M$ALLOC is included in the total shown, but the operating system's overhead is not. Memory allocated by M$ALLOC is initialized to binary zeros (LOW VALUES).

If you try to allocate more memory than the environment can give you, M$ALLOC will return NULL, and no memory will be allocated.

M$COPY (Dynamic Memory Routine)

Copies a region of memory from one location to another.

Usage

```
CALL "M$COPY"
    USING DEST-PTR, SRC-PTR, NUM-BYTES
```

Parameters

DEST-PTR USAGE POINTER

Contains the address of the first byte of the destination region.

SRC-PTR USAGE POINTER

Contains the address of the first byte of the source region.

NUM-BYTES Numeric parameter

Indicates the size of the memory region to be copied.

Description

This routine copies NUM-BYTES from the address contained in SRC-PTR to the address contained in DEST-PTR. Note that this routine is relatively dangerous to use. No boundary checking is performed to ensure that the address range is valid, so memory access violations may result if you pass it incorrect data.

This routine is functionally similar to the C$MEMCOPY routine except that parameters are passed by reference instead of by value. For example, you can copy 10 bytes to DEST-PTR from the memory address contained in SRC-PTR with:

```
CALL "M$COPY"
   USING DEST-PTR, SRC-PTR, 10
```

M$FILL (Dynamic Memory Routine)

This routine is used to set a region of memory to a constant value.

Usage

```
CALL "M$FILL"
    USING DEST-PTR, BYTE-VALUE, NUM-BYTES
```

Parameters

DEST-PTR USAGE POINTER

Contains the address of the first byte of the region to be filled.

BYTE-VALUE Alpha-numeric parameter

Contains the value with which to fill the memory region.

NUM-BYTES Numeric parameter

Indicates the size of the memory region.

Description

This routine fills NUM-BYTES with BYTE-VALUE starting at address DEST-PTR. The parameters are passed BY REFERENCE. This routine does not do any boundary checking to make sure that the address range is valid.

M$FREE (Dynamic Memory Routine)

Frees a previously allocated piece of memory.

Usage

```
CALL "M$FREE"
    USING MEM-ADDRESS
```

Parameter

MEM-ADDRESS USAGE POINTER

Must point to a memory area previously allocated by M$ALLOC.

Comments

Use M$FREE to release a memory block allocated by M$ALLOC. This memory is returned to the pool of memory available for use by the runtime. On most operating systems, this memory is still associated with the runtime's

process, so it cannot be used by any other processes. On a few systems, this memory may be made available to the operating system for re-use by other processes.

It is an error to attempt to use a block of memory once it has been freed. It is also an error to free a block of memory more than once or to free a memory address that has never been allocated. Any of these errors can lead to memory access violations. The runtime attempts to detect these errors and avoid them, but it cannot detect all such errors.

M$GET (Dynamic Memory Routine)

Retrieves data from an allocated memory block.

Usage

```
CALL "M$GET"
    USING MEM-ADDRESS, DATA-ITEM, DATA-SIZE, DATA-OFFSET
```

Parameters

MEM-ADDRESS USAGE POINTER

Must point to a memory area previously allocated by M$ALLOC.

DATA-ITEM Any data item

Data from the memory block will be stored in this item.

DATA-SIZE Numeric parameter (optional)

The number of bytes to move from the memory block. If omitted, then the number of bytes is set to the size of the memory block (excluding overhead bytes).

DATA-OFFSET Numeric parameter (optional)

The location within the memory block from which to start the move. The first location is position “1”. If omitted, this value defaults to “1”.

Description

This routine retrieves data from the memory block at MEM-ADDRESS and stores it in DATA-ITEM. Regardless of the value of DATA-SIZE, no bytes are copied from past the end of the memory block. Note that the size of DATA-ITEM is not checked.

M$PUT (Dynamic Memory Routine)

Stores data in an allocated memory block.

Usage

```
CALL "M$PUT"
    USING MEM-ADDRESS, DATA-ITEM, DATA-SIZE, DATA-OFFSET
```

Parameters

MEM-ADDRESS USAGE POINTER

Must point to a memory area previously allocated by M$ALLOC.

DATA-ITEM Any data item

This is the data that will be stored in the memory block.

DATA-SIZE Numeric parameter (optional)

The number of bytes to move to the memory block. If omitted, then the number of bytes is set to the size of the memory block (excluding overhead bytes).

DATA-OFFSET PIC 9(n), USAGE DISPLAY or COMP-4 (optional)

The location within the memory block from which to start the move. The first location is position “1”. If omitted, this value defaults to “1”.

Description

This routine copies DATA-ITEM into the memory pointed to by MEM-ADDRESS for DATA-SIZE bytes. Regardless of the value of DATA-SIZE, no bytes are copied that exceed the size of the memory block at MEM-ADDRESS.

OCTAL2ASCII

OCTAL2ASCII converts data presented in octal format to its ASCII equivalent. This routine is the inverse of the **ASCII2OCTAL** routine.

Usage

```
CALL "OCTAL2ASCII"
    USING ASCII-VALUE, OCTAL-VALUE
```

Parameters

ASCII-VALUE PIC X(2)

The ASCII representation of a unit of data.

OCTAL-VALUE PIC X(8)

The octal value.

When you are defining the parameters, use the exact field sizes specified in the calling conventions above, otherwise the runtime may terminate abnormally.

Any characters that are not valid octal digits are treated as the digit “0”. Trailing spaces are removed from the input value.

Routines to Handle the Windows Registry

Caution: When you are changing registry settings, Windows does not validate any of the values you write to it—any operation is allowed. Therefore, you must be very careful of what changes you make to the registry because no error messages will appear if you make a mistake. You should create a backup of your Windows registry before making any changes. Even better, back up your entire configuration, so that it could be restored in full in the event of a crash.

The library routines that follow enable you to create and query Windows Registry keys. Each routine can be called by either of two names. Use the DISPLAY_REG_* name when you want to work with the registry on the display host (this is the local host when the application is run with a standard runtime, and the *thin client* when the application is run with the thin client). Use the REG_* name when you want to work with the registry on the server host (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

If the routines are called from a non-Windows platform, they will return an error status code of "-1".

The Windows Registry contains the hardware, software, and user preferences that determine the configuration of a particular PC. When a user makes changes to the desktop, file structure, system configuration or software, the changes are reflected in the registry. This information used to be contained in files like config.sys, autoexec.bat, win.ini, system.ini and control.ini, but the registry provides a single source for accessing all of a computer's settings.

Each setting in the registry has a handle or *Key.* Only two registry keys are actually stored on the hard disk: HKEY_LOCAL_MACHINE and HKEY_USERS. All other registry keys are aliases or branches of these two principal keys, or are created dynamically by Windows. For example, HKEY_CLASSES_ROOT is an alias of the branch HKEY_LOCAL_MACHINE\Software\Classes. In order to interact with these registry keys, the Windows Application Programmer's Interface (API)

provides a series of functions that call the registry. We have created the library routines below to operate registry functions from within your COBOL program.

For security reasons, in thin client deployments several of the functions require user authorization before the action is performed. These functions include:

DISPLAY_REG_CREATE_KEY
DISPLAY_REG_CREATE_KEY_EX
DISPLAY_REG_DELETE_KEY
DISPLAY_REG_DELETE_VALUE
DISPLAY_REG_SET_VALUE
DISPLAY_REG_SET_VALUE_EX

When one of these functions is called, a Security Information dialog is displayed and the user must select a Yes or No button to indicate whether the action may proceed.

REG_CLOSE_KEY, DISPLAY_REG_CLOSE_KEY

Closes a specified registry key by providing access to the Windows registry routine RegCloseKey.

Use DISPLAY_REG_CLOSE_KEY when you want the action to be performed on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_CLOSE_KEY when you want the action to be performed on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_CLOSE_KEY"
    USING OPEN-KEY-HANDLE
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of an open key to close.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft's "RegCloseKey" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

Registry keys should not be left open any longer than necessary.

REG_CREATE_KEY, DISPLAY_REG_CREATE_KEY

Creates a specified registry key by providing access to the Windows registry routine RegCreateKey. If the key already exists, this function opens it (same functionality as REG_OPEN_KEY). If the key does not exist, it is created and then opened.

Use DISPLAY_REG_CREATE_KEY when you want the action to be performed on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_CREATE_KEY when you want the action to be performed on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_CREATE_KEY"
    USING OPEN-KEY-HANDLE, SUBKEY-TO-BE-CREATED,
        SUBKEY-HANDLE
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in “acugui.def”):

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

The key created or opened by REG_CREATE_KEY is a subkey of the key identified by OPEN-KEY-HANDLE.

SUBKEY-TO-BE-CREATED PIC X(n)

Name of the subkey to create or open.

SUBKEY-HANDLE Usage unsigned-long

Data item to receive the handle of the newly created or opened key.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft’s “RegCreateKey” function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

REG_CREATE_KEY may be used to create several keys at once.

For example, by setting SUBKEY-TO-BE-CREATED to

```
"key1\key2\key3"
```

you could create three keys, where “key3” is a subkey of “key2”, “key2” is a subkey of “key1”, and “key1” is a subkey of the key specified by OPEN-KEY-HANDLE. The lowest-level key (“key3” in this example) is the one that is opened and has its handle returned in SUBKEY-HANDLE.

REG_CREATE_KEY_EX, DISPLAY_REG_CREATE_KEY_EX

Creates a specified registry key by providing access to the Windows registry routine RegCreateKeyEx. If the key already exists, this function opens it (same functionality as REG_OPEN_KEY_EX). If the key does not exist, it is created and then opened.

Use DISPLAY_REG_CREATE_KEY_EX when you want the action to be performed on the display host’s registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_CREATE_KEY_EX when you want the action to be performed on the server host’s registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_CREATE_KEY_EX"
    USING OPEN-KEY-HANDLE, SUBKEY-TO-BE-CREATED,CLASS-NAME,
        OPTIONS, SAM-DESIRED, SUBKEY-HANDLE, DISPOSITION
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in “acugui.def”):

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

The key created or opened by REG_CREATE_KEY_EX is a subkey of the key identified by OPEN-KEY-HANDLE.

SUBKEY-TO-BE-CREATED PIC X(n)

Name of the subkey to create or open.

CLASS-NAME PIC X(n)

Specifies the class (object type) of the key to be created. This parameter is ignored if the key already exists.

OPTIONS Usage unsigned-long

Specifies special options for the key. This parameter must be one of the following values (defined in “acugui.def”):

Value	**Meaning**
REG_OPTION_VOLATILE	The value of this key varies depending on the Windows operating system used: **Windows 98**: This value is ignored. That is, even if REG_OPTION_VOLATILE is specified, the RegCreateKeyEx function creates a nonvolatile key and returns ERROR_SUCCESS. **Windows NT 4.0, Windows 2000**: This key is volatile; the information is stored in memory and is not preserved when the system is restarted.
REG_OPTION_NON_VOLATILE	This key is not volatile; the information is stored in a file and is preserved when the system is restarted.

By default, keys are not volatile. This option is ignored if the key already exists.

SAM-DESIRED Usage unsigned-long

Specifies a security access mask (SAM) that describes the desired security access for the new key. This parameter can be a combination of the following values (defined in “acugui.def”):

Value	Meaning
KEY_ALL_ACCESS	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, EY_CREATE_SUB_KEY, KEY_CREATE_LINK, and KEY_SET_VALUE access.
KEY_CREATE_LINK	Permission to create a symbolic link.
KEY_CREATE_SUB_KEY	Permission to create subkeys.
KEY_ENUMERATE_SUB_KEYS	Permission to enumerate subkeys.
KEY_EXECUTE	Permission for read access.
KEY_NOTIFY	Permission for change notification.
KEY_QUERY_VALUE	Permission to query subkey data.
KEY_READ	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY access.
KEY_SET_VALUE	Permission to set subkey data.
KEY_WRITE	Combination of KEY_SET_VALUE and KEY_CREATE_SUB_KEY access.

The above values may be combined in COBOL applications by using the “CBL_OR” library routine.

SUBKEY-HANDLE Usage unsigned-long

Data item to receive the handle of the newly created or opened key.

DISPOSITION Usage unsigned-long

Points to a variable that receives one of the following disposition values (defined in “acugui.def”):

Value	Meaning
REG_CREATED_NEW_KEY	The key did not exist and was created.
REG_OPENED_EXISTING_KEY	The key existed and was simply opened without being changed.

STATUS-CODE Numeric data item

Receives the return status of the call to Microsoft’s “RegCreateKeyEx” function. A return status of zero indicates success; non-zero indicates that an error occurred.

REG_DELETE_KEY, DISPLAY_REG_DELETE_KEY

Deletes a specified registry key by providing access to the Windows registry routine RegDeleteKey. When a key is deleted, its value and all of its subkeys are also deleted.

Note: On Windows NT systems only, this routine cannot be used to delete keys that have subkeys.

Use DISPLAY_REG_DELETE_KEY when you want the action to be performed on the display host’s registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_DELETE_KEY when you want the action to be performed on the server host’s registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_DELETE_KEY"
    USING OPEN-KEY-HANDLE, SUBKEY-TO-BE-DELETED,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in “acugui.def”):

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

The key deleted by REG_DELETE_KEY is a subkey of the key identified by OPEN-KEY-HANDLE.

SUBKEY-TO-BE-DELETED PIC X(n)

Name of the key to delete. Must be a subkey of the key identified by OPEN-KEY-HANDLE. If SUBKEY-TO-BE-DELETED is set to NULL or to a string consisting of all spaces, the routine does nothing and returns with STATUS-CODE set to ‘13’.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft’s “RegDeleteKey” function. A return status of zero indicates success; non-zero indicates that an error occurred. On 32-bit Windows systems, a return status of ‘5’ indicates that either the application does not have delete privileges for the specified key, or another application has the key opened.

REG_DELETE_VALUE, DISPLAY_REG_DELETE_VALUE

Deletes a named value from a specified registry key.

Use DISPLAY_REG_DELETE_VALUE when you want the action to be performed on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_DELETE_VALUE when you want the action to be performed on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_DELETE_VALUE"
    USING OPEN-KEY-HANDLE, VALUE-NAME,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

VALUE-NAME PIC X(n)

Name of the value to delete. If NULL or an empty string, the value set by REG_SET_VALUE will be deleted.

STATUS-CODE Numeric data item.

Receives the return status of call to Microsoft's "RegDeleteValue" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

The key identified by OPEN-KEY-HANDLE must have been opened with KEY_SET_VALUE access (KEY_WRITE access includes KEY_SET_VALUE access).

REG_ENUM_KEY, DISPLAY_REG_ENUM_KEY

Enumerates subkeys of a specified open registry key. REG_ENUM_KEY retrieves the name of one subkey each time it is called.

Use DISPLAY_REG_ENUM_KEY to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_ENUM_KEY to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_ENUM_KEY"
    USING OPEN-KEY-HANDLE, NDX, SUBKEY-NAME, NAME-SIZE,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

The keys enumerated by REG_ENUM_KEY are subkeys of the key identified by OPEN-KEY-HANDLE.

NDX Numeric data item

Specifies the index of the subkey to retrieve. NDX should be set to '1' for the first call to REG_ENUM_KEY and then incremented for subsequent calls. The subkeys are not returned in any particular order.

SUBKEY-NAME PIC X(n)

Receives the name of the subkey. REG_ENUM_KEY copies only the name of the subkey, not the full key hierarchy, to the SUBKEY-NAME buffer.

NAME-SIZE Numeric data item

Specifies the size, in characters, of the SUBKEY-NAME buffer.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft's "RegEnumKey" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

To enumerate subkeys, your application should initialize the NDX parameter to '1' and call REG_ENUM_KEY repeatedly, incrementing NDX each time, until there are no more subkeys. You can tell that there are no more subkeys when the function returns a non-zero STATUS-CODE ('259' for 32-bit Windows).

While an application is using REG_ENUM_KEY, it should not make calls to any registry routines that might change the key being queried.

If the subkey name exceeds the size of the SUBKEY-NAME buffer (as specified by the NAME-SIZE parameter), the result depends on the operating system. Under 32-bit Windows, a STATUS-CODE of '234' is returned.

REG_ENUM_VALUE, DISPLAY_REG_ENUM_VALUE

Enumerates the values of a specified registry key. REG_ENUM_VALUE retrieves the name of one value each time it is called.

Use DISPLAY_REG_ENUM_VALUE to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_ENUM_VALUE to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_ENUM_VALUE"
    USING OPEN-KEY-HANDLE, NDX, VALUE-NAME, NAME-SIZE,
        DATA-TYPE, VALUE-DATA, DATA-SIZE,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in “acugui.def”):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

NDX Numeric data item

Specifies the index of the value to retrieve. NDX should be set to ‘1’ for the first call to REG_ENUM_VALUE and then incremented for subsequent calls. The values are not returned in any particular order.

VALUE-NAME PIC X(n)

Receives the name of the value.

NAME-SIZE Usage unsigned-long

Specifies the size, in characters, of the VALUE-NAME buffer. When the function returns, NAME-SIZE contains the size of the string copied to VALUE-NAME.

DATA-TYPE Usage unsigned-long

Points to a variable that receives the type code for the value entry. The type code can be one of the following values (defined in “acugui.def”):

Value	**Meaning**
REG_BINARY	Binary data in any form.
REG_DWORD	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN	A 32-bit number in little-endian format (same as REG_DWORD). In little-endian format, the most significant byte of a word is the high-order word. This is the most common format for computers running Windows and Windows NT.
REG_DWORD_BIG_ENDIAN	A 32-bit number in big-endian format. In big-endian format, the most significant byte of a word is the low-order word.
REG_EXPAND_SZ	A null-terminated string that contains unexpanded references to environment variables (for example, “%PATH%”). It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI functions.
REG_LINK	A Unicode symbolic link.
REG_MULTI_SZ	An array of null-terminated strings, terminated by two null characters.
REG_NONE	No defined value type.
REG_RESOURCE_LIST	A device-driver resource list.
REG_SZ	A null-terminated string. It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.

VALUE-DATA Variable parameter

Buffer to receive the data for the value entry. If you know what type of data is being returned, you may specify this parameter accordingly. If the type of data returned is unknown, you may specify a group item structured as follows:

```
01 VALUE-DATA
   02 VALUE-DATA-ORIG                                PIC X (n).
   02 VALUE-DWORD redefines VALUE-DATA-ORIG         Usage signed-long.
   02 VALUE-LIT-ENDIAN redefines VALUE-DATA-ORIG    Usage signed-long.
   02 VALUE-BIG-ENDIAN redefines VALUE-DATA-ORIG     S9 (9) COMP-4.
   02 VALUE-EXPAND-SZ redefines VALUE-DATA-ORIG      PIC X (n).
   02 VALUE-LINK redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-MULTI-SZ redefines VALUE-DATA-ORIG       PIC X (n).
   02 VALUE-NONE redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-RESOURCE-LIST redefines VALUE-DATA-ORIG  PIC X (n).
   02 VALUE-SZ redefines VALUE-DATA-ORIG             PIC X (n).
```

DATA-SIZE Usage unsigned-long

Specifies the size, in bytes, of the VALUE-DATA buffer. When the function returns, DATA-SIZE contains the number of bytes copied to VALUE-DATA. This parameter can be NULL only if VALUE-DATA is NULL.

STATUS-CODE Numeric data item

Receives the return status of call to Microsoft's "RegEnumValue" function. A return status of zero indicates success; non-zero indicates that an error occurred. A value of '259' indicates that there are no more values for the specified key.

Comments

To enumerate values, an application should initialize the NDX parameter to '1' and call REG_ENUM_VALUE repeatedly, incrementing NDX each time, until there are no more values (that is, until the function returns with STATUS-CODE set to '259').

While an application is using REG_ENUM_VALUE, it should not make calls to any registry routines that might change the values being queried.

The key identified by the OPEN-KEY-HANDLE parameter must have been opened with KEY_QUERY_VALUE access.

REG_OPEN_KEY, DISPLAY_REG_OPEN_KEY

Opens a specified registry key by accessing the Windows registry routine RegOpenKey.

Use DISPLAY_REG_OPEN_KEY to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_OPEN_KEY to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_OPEN_KEY"
   USING OPEN-KEY-HANDLE, SUBKEY-TO-BE-OPENED, SUBKEY-HANDLE,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

SUBKEY-TO-BE-OPENED PIC X(n)

String containing the name of the key to open. This key must be a subkey of the key identified by OPEN-KEY-HANDLE.

SUBKEY-HANDLE Usage unsigned-long

Data item to receive the handle of the opened subkey.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft's "RegOpenKey" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

You can use REG_OPEN_KEY to open a key several layers deep in the registry tree structure. For example, if you set SUBKEY-TO-BE-OPENED to a string of the form:

```
"level_1\level_2\level_3"
```

this would open the key "level_3", where "level_3" is a subkey of "level_2", "level_2" is a subkey of "level_1", and "level_1" is a subkey of the key specified by OPEN-KEY-HANDLE.

REG_OPEN_KEY_EX, DISPLAY_REG_OPEN_KEY_EX

Opens a specified registry key by accessing the Windows registry routine "RegOpenKeyEX".

Use DISPLAY_REG_OPEN_KEY_EX to perform the action on the display host's registry (the local host when the application is run with the thin client). Use REG_OPEN_KEY_EX to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_OPEN_KEY_EX"
    USING OPEN-KEY-HANDLE, SUBKEY-TO-BE-OPENED, SAM-DESIRED,
        SUBKEY-HANDLE,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

SUBKEY-TO-BE-OPENED PIC X(n)

String containing the name of the key to open. This key must be a subkey of the key identified by OPEN-KEY-HANDLE.

SAM-DESIRED Usage unsigned-long

Specifies a security access mask (SAM) that describes the desired security access for the new key. This parameter can be a combination of the following values (defined in “acugui.def”):

Value	**Meaning**
KEY_ALL_ACCESS	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, KEY_CREATE_SUB_KEY, KEY_CREATE_LINK, and KEY_SET_VALUE access.
KEY_CREATE_LINK	Permission to create a symbolic link.
KEY_CREATE_SUB_KEY	Permission to create subkeys.
KEY_ENUMERATE_SUB_KEYS	Permission to enumerate subkeys.
KEY_EXECUTE	Permission for read access.
KEY_NOTIFY	Permission for change notification.
KEY_QUERY_VALUE	Permission to query subkey data.
KEY_READ	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY access.
KEY_SET_VALUE	Permission to set subkey data.
KEY_WRITE	Combination of KEY_SET_VALUE and KEY_CREATE_SUB_KEY access.

You may combine the above in COBOL by using the “CBL_OR” library routine.

SUBKEY-HANDLE Usage unsigned-long

Data item to receive the handle of the opened key.

STATUS-CODE Numeric data item.

Receives the return status of call to Microsoft’s “RegOpenKeyEx” function. A return status of zero indicates success; non-zero indicates that an error occurred.

REG_QUERY_VALUE, DISPLAY_REG_QUERY_VALUE

Retrieves the value associated with a specified registry key by accessing the Windows registry routine RegQueryValue.

Use DISPLAY_REG_QUERY_VALUE to perform the action on the display host’s registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_QUERY_VALUE to perform the action on the server host’s registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_QUERY_VALUE"
    USING OPEN-KEY-HANDLE, RETURN-VALUE, RETURN-SIZE,
    GIVING STATUS-CODE
```

or

```
CALL "REG_QUERY_VALUE"
    USING OPEN-KEY-HANDLE, RETURN-VALUE, RETURN-SIZE,
       SUBKEY-NAME,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

RETURN-VALUE PIC X(n)

Receives the value associated with the specified key.

RETURN-SIZE usage unsigned-long

Specifies the size, in characters, of the RETURN-VALUE buffer. When the function returns, RETURN-SIZE contains the size of the string copied to RETURN-VALUE.

SUBKEY-NAME PIC X(n) (optional)

Name of a subkey of OPEN-KEY-HANDLE for which a value is to be retrieved. If SUBKEY-NAME is omitted or contains an empty string, REG_QUERY_VALUE retrieves the value associated with the key identified by OPEN-KEY-HANDLE.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft's "RegQueryValue" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

When REG_QUERY_VALUE is called repeatedly in a loop, be sure to reset the value of RETURN-SIZE between calls.

If the length of the value string exceeds the size of the RETURN-VALUE buffer (the buffer size is indicated in the input value of RETURN-SIZE), then the behavior of the routine depends on the operating system. On 32-bit Windows systems, a value string that exceeds the size of the

RETURN-VALUE buffer causes RETURN-SIZE to be set to the full length of the value string. In this case, the value string is not copied into RETURN-VALUE, and STATUS-CODE is set to '234'.

REG_QUERY_VALUE_EX, DISPLAY_REG_QUERY_VALUE_EX

Retrieves the type and data for a specified value name associated with an open registry key.

Use DISPLAY_REG_QUERY_VALUE_EX to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_QUERY_VALUE_EX to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_QUERY_VALUE_EX"
    USING OPEN-KEY-HANDLE, VALUE-NAME, DATA-TYPE, VALUE-DATA,
        DATA-SIZE,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

VALUE-NAME PIC X(n)

Name of the value to be queried.

DATA-TYPE Usage unsigned-long

Points to a variable that receives the type code for the value entry. The type code can be one of the following values (defined in "acugui.def"):

Value	**Meaning**
REG_BINARY	Binary data in any form.
REG_DWORD	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN	A 32-bit number in little-endian format (same as REG_DWORD). In little-endian format, the most significant byte of a word is the high-order word. This is the most common format for computers running Windows and Windows NT.
REG_DWORD_BIG_ENDIAN	A 32-bit number in big-endian format. In big-endian format, the most significant byte of a word is the low-order word.
REG_EXPAND_SZ	A null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI functions.
REG_LINK	A Unicode symbolic link.
REG_MULTI_SZ	An array of null-terminated strings, terminated by two null characters.
REG_NONE	No defined value type.
REG_RESOURCE_LIST	A device-driver resource list.
REG_SZ	A null-terminated string. It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.

VALUE-DATA Variable parameter

Buffer to receive the data for the value entry. If you know what type of data is being returned, you may specify this parameter accordingly. If the type of data returned is unknown, you may specify a group item structured as follows:

```
01 VALUE-DATA
   02 VALUE-DATA-ORIG                                PIC X (n).
   02 VALUE-BINARY redefines VALUE-DATA-ORIG         PIC X (n).
   02 VALUE-DWORD redefines VALUE-DATA-ORIG         Usage signed-long.
   02 VALUE-LIT-ENDIAN redefines VALUE-DATA-ORIG    Usage signed-long.
   02 VALUE-BIG-ENDIAN redefines VALUE-DATA-ORIG     S9 (9) COMP-4.
   02 VALUE-EXPAND-SZ redefines VALUE-DATA-ORIG      PIC X (n).
   02 VALUE-LINK redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-MULTI-SZ redefines VALUE-DATA-ORIG       PIC X (n).
   02 VALUE-NONE redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-RESOURCE-LIST redefines VALUE-DATA-ORIG  PIC X (n).
   02 VALUE-SZ redefines VALUE-DATA-ORIG             PIC X (n).
```

DATA-SIZE Usage unsigned-long

Prior to the call, must specify the size, in bytes, of the VALUE-DATA buffer. When the function returns, DATA-SIZE contains the number of bytes copied to VALUE-DATA.

STATUS-CODE Numeric data item

Receives the return status of call to Microsoft's "RegQueryValueEx" function. A return status of zero indicates success; non-zero indicates that an error occurred. A return status of '234' indicates that the size of the data to be returned in VALUE-DATA exceeds the buffer size specified by the DATA-SIZE parameter.

Comments

The key identified by OPEN-KEY-HANDLE must have been opened with KEY_QUERY_VALUE access.

This function does not expand the environment-variable names in the value data when the value type is REG_EXPAND_SZ.

When calling the REG_QUERY_VALUE_EX function with OPEN-KEY-HANDLE set to the HKEY_PERFORMANCE_DATA handle and a value string of a specified object, the returned data structure sometimes

has unrequested objects. Don't be surprised; this is normal behavior. When calling the REG_QUERY_VALUE_EX function, you should always expect to 'walk' the returned data structure to look for the requested object.

REG_SET_VALUE, DISPLAY_REG_SET_VALUE

Associates a value with a specified registry key by accessing the Windows registry routine RegSetValue.

Use DISPLAY_REG_SET_VALUE to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_SET_VALUE to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_SET_VALUE"
    USING OPEN-KEY-HANDLE, VALUE,
    GIVING STATUS-CODE
```

or

```
CALL "REG_SET_VALUE"
    USING OPEN-KEY-HANDLE, VALUE, SUBKEY-NAME,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in "acugui.def"):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

VALUE PIC X(n)

String containing the value to set for the specified key.

SUBKEY-NAME PIC X(n) (optional)

Name of a subkey of OPEN-KEY-HANDLE with which to associate a value. If there is no existing subkey matching the name specified, REG_SET_VALUE will first create it. If SUBKEY-NAME is omitted or contains an empty string, the specified value will be associated with the key identified by OPEN-KEY-HANDLE.

STATUS-CODE Numeric data item.

Receives the return status of the call to Microsoft's "RegSetValue" function. A return status of zero indicates success; non-zero indicates that an error occurred.

REG_SET_VALUE_EX, DISPLAY_REG_SET_VALUE_EX

Sets value and type information for a specified open registry key.

Use DISPLAY_REG_SET_VALUE_EX to perform the action on the display host's registry (the local host when the application is run with a standard runtime; the *thin client* when the application is run with the thin client). Use REG_SET_VALUE_EX to perform the action on the server host's registry (the local host when the application is run with a standard runtime, and the application host when the application is run with the thin client).

Usage

```
CALL "REG_SET_VALUE_EX"
    USING OPEN-KEY-HANDLE, DATA-TYPE, VALUE-DATA, DATA-SIZE,
    GIVING STATUS-CODE
```

or:

```
CALL "REG_SET_VALUE_EX"
    USING OPEN-KEY-HANDLE, DATA-TYPE, VALUE-DATA, DATA-SIZE,
        VALUE-NAME,
    GIVING STATUS-CODE
```

Parameters

OPEN-KEY-HANDLE Usage unsigned-long

Handle of a currently open key or one of the following predefined handles of keys that are always open (defined in “acugui.def”):

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

DATA-TYPE Usage unsigned-long

Specifies the type code for the value entry. The type code can be one of the following values (defined in “acugui.def”):

Value	**Meaning**
REG_BINARY	Binary data in any form.
REG_DWORD	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN	A 32-bit number in little-endian format (same as REG_DWORD). In little-endian format, the most significant byte of a word is the high-order word. This is the most common format for computers running Windows and Windows NT.
REG_DWORD_BIG_ENDIAN	A 32-bit number in big-endian format. In big-endian format, the most significant byte of a word is the low-order word.
REG_EXPAND_SZ	A null-terminated string that contains unexpanded references to environment variables (for example, “%PATH%”). It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI functions.
REG_LINK	A Unicode symbolic link.
REG_MULTI_SZ	An array of null-terminated strings, terminated by two null characters.
REG_NONE	No defined value type.

Value	Meaning
REG_RESOURCE_LIST	A device-driver resource list.
REG_SZ	A null-terminated string. It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.

VALUE-DATA Variable parameter

Buffer to set the data for the value entry. If you know what type of data is being returned, you may specify this parameter accordingly. If the type of data returned is unknown, you may specify a group item structured as follows:

```
01 VALUE-DATA
   02 VALUE-DATA-ORIG                                PIC X (n).
   02 VALUE-BINARY redefines VALUE-DATA-ORIG         PIC X (n).
   02 VALUE-DWORD redefines VALUE-DATA-ORIG         Usage signed-long.
   02 VALUE-LIT-ENDIAN redefines VALUE-DATA-ORIG    Usage signed-long.
   02 VALUE-BIG-ENDIAN redefines VALUE-DATA-ORIG     S9 (9) COMP-4.
   02 VALUE-EXPAND-SZ redefines VALUE-DATA-ORIG      PIC X (n).
   02 VALUE-LINK redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-MULTI-SZ redefines VALUE-DATA-ORIG       PIC X (n).
   02 VALUE-NONE redefines VALUE-DATA-ORIG           PIC X (n).
   02 VALUE-RESOURCE-LIST redefines VALUE-DATA-ORIG  PIC X (n).
   02 VALUE-SZ redefines VALUE-DATA-ORIG             PIC X (n).
```

DATA-SIZE Usage unsigned-long

Specifies the size, in bytes, of the information in the VALUE-DATA buffer. If the data is of type REG_SZ, REG_EXPAND_SZ, or REG_MULTI_SZ, DATA-SIZE must include the size of the terminating null character.

VALUE-NAME PIC X(n)

Name of the value to set. If a value with this name is not already present in the key, the function adds it to the key.

STATUS-CODE Numeric data item.

Receives the return status of call to Microsoft's "RegSetValueEx" function. A return status of zero indicates success; non-zero indicates that an error occurred.

Comments

Value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as files with the filenames stored in the registry. This helps the registry perform efficiently. Application elements such as icons, bitmaps, and executable files should be stored as files and not be placed in the registry.

The key identified by the OPEN-KEY-HANDLE parameter must have been opened with KEY_SET_VALUE access.

RENAME

Renames the existing file. RENAME takes two, three, or four USING parameters.

Usage

```
CALL "RENAME"
    USING SOURCE-FILE, DEST-FILE, RENAME-STATUS, FILE-TYPE
```

Parameters

SOURCE-FILE PIC X(n)

Contains the name of the file to rename.

DEST-FILE PIC X(n)

Contains the new name for the file. The RENAME routine will remove the destination file if necessary.

RENAME-STATUS PIC 9(9) COMP-4

This parameter will be set to ZERO if the routine is successful, otherwise it will be set to the operating system's error number.

FILE-TYPE PIC X

Indicates the file type. If the FILE-TYPE parameter is used, it must specify either an "S" for a sequential file, an "R" for a relative file, or an "I" for an indexed file. If FILE-TYPE is not used, it is assumed to be "S". This can be useful in cases where the file is held in more than one physical disk file (for example, C-ISAM indexed files are physically held in two separate files). If the FILE-TYPE parameter is omitted, then only the single physical file named in SOURCE-FILE is renamed.

Comments

The file being renamed should be *closed*. Note that the RENAME routine cannot rename a file across different drives under Windows, but it can change a file's directory. Under UNIX systems, the RENAME procedure will attempt to copy the file if the target name is on a different device. This may take some time if the file is large. If the copy is successful, the original file is removed.

Note: The behavior of this library routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the documentation on **FILENAME_SPACES** in Appendix H for information about the terminating character for path names.

R$IO

The R$IO routine provides an interface to the relative file handler. Calls to the routine require an operation code and a variable number of parameters, depending on the operation called. The return code is set automatically after the call. The external variable "F-ERRNO" is set according to any errors found. "F-ERRNO" may not be reset on entry to R$IO, and should be checked only if R$IO returns an error condition.

This section includes the following topics:

Usage

```
CALL "R$IO"
    USING OP-CODE, PARAMETERS
```

Parameters

OP-CODE Numeric parameter

Specifies the file handling routine to be performed. The following table lists each op-code and its corresponding operation. The operations are detailed below:

Code	Operation
1	R-OPEN-FUNCTION
2	R-CLOSE-FUNCTION
3	R-MAKE-FUNCTION
4	R-READ-FUNCTION
5	R-NEXT-FUNCTION
6	R-PREVIOUS-FUNCTION
7	R-START-FUNCTION
8	R-WRITE-FUNCTION
9	R-REWRITE-FUNCTION
10	R-DELETE-FUNCTION
11	R-UNLOCK-FUNCTION

PARAMETERS vary depending on op-code

The remaining parameters vary depending on the operation selected. They provide information and hold results for the operations specified. Parameters may be omitted from those operations that do not require them. The parameters are detailed below.

Description

All parameters passed to R$IO are passed by reference. This applies even to parameters that are integer values in the corresponding file handling routines.

R$IO automatically terminates PIC X filename parameters with a LOW-VALUES byte.

You do not have to specify SYNC for level 01 or level 77 parameters because they are automatically synchronized by ACUCOBOL-GT.

The "filesys.def" COPY file contains many useful definitions for use with R$IO. It contains definitions for the R$IO op-codes as well as the "F-ERRNO" error values. It also includes many useful pre-declared variables that are of the proper type and usage.

Note: The behavior of this routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the entry for **FILENAME_SPACES** in Appendix H, for information about the terminating character for path names.

The runtime configuration variables FILE_PREFIX and FILE_SUFFIX are ignored by the R$IO routine.

OP-CODES and PARAMETERS

R-OPEN-FUNCTION (op-code 1)

This routine opens an existing relative file. If it is successful, the value in RETURN-CODE should be moved to a data item that is USAGE HANDLE. This data item is passed as the open file handle to the other file handling routines. If it fails, RETURN-CODE is set to a NULL value.

The R-OPEN-FUNCTION routine takes four required parameters, and one optional parameter: *filename*, *mode*, *maxsize*, *minsize*, and *blocks*.

Filename is the name of the file to open. It does not need to be null-terminated.

Mode is one of the following values (defined in “filesys.def”):

Finput	open for input only
Foutput	open for output only
Fio	open for input and output
Fextend	open for output only (same as Foutput)

“Foutput” does not delete the current file (this behavior differs from the OPEN OUTPUT statement in COBOL).

This routine only opens files that already exist. If the file does not exist, the routine fails, even when opening with mode “Foutput”.

Mode may furthermore have one of the following flags added to it to indicate file locking options (defined in “filesys.def”):

Fread_lock	locks file against other updaters
Fwrite_lock	locks file against all others
Fmass_update	locks file against all others (same as Fwrite_lock)

If “Fmass_update” is used, the file system is directed to emphasize speed of updates over file security.

Additionally, “Fmulti_lock” may be added to mode to request that more than one record lock be maintained in the file by this process. If this option is not specified, then any I/O operation on the file will first release any currently locked record. This results in only one record lock being set in the file at any time. When this option is used, locked records are released only when the file is closed or when the UNLOCK routine is called.

Maxsize is the maximum record size.

Minsize is the minimum record size. If maxsize is not equal to minsize, the records are considered variable length.

Blocks is the size of a block in bytes. This parameter is optional and defaults to zero (“0”), meaning that a block is the size of a record.

R-CLOSE-FUNCTION (op-code 2)

This routine closes an open file. It also removes currently held locks on the file. R-CLOSE-FUNCTION has only one parameter, *file_handle*, the file handle of the file to close. The file handle is a handle returned by the R-OPEN-FUNCTION. For some file systems, it is possible that R-CLOSE-FUNCTION will write additional records that had been previously buffered by the system. For this reason, it is possible that a "disk full" condition can occur.

R-MAKE-FUNCTION (op-code 3)

This routine creates a new relative file. It will overwrite any existing file of the same name. However, it will not overwrite a file that is currently in use. If the file is in use, the error E_FILE_LOCKED is returned.

The R-MAKE-FUNCTION routine has two required parameters and one optional parameter: *filename*, *l_params*, *blocks*.

Filename is the name of the file to create. The name need not be null-terminated.

L_parms is a string that describes logical characteristics of the file. The string consists of two numeric fields separated by a comma. The string must be null-terminated. The fields are as follows:

1. Maximum record size. This is the size in bytes of the largest record to be placed in the file. This can range from 1 to 67,108,864.

2. Minimum record size.

Blocks is the size of a block of records in bytes. The default is zero ("0").

R-READ-FUNCTION (op-code 4)

This routine reads a record out of the relative file. The R-READ-FUNCTION routine has three parameters: *file_handle*, *record_area*, and *keyval*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Record_area points to the area to hold the record read. It must be at least MAXSIZE bytes in length.

Keyval is the record number of the record to read. It is a long value.

If R-READ-FUNCTION succeeds, RETURN-CODE is set to the size of the record read, plus one. If it fails, RETURN-CODE is set to zero. However, if the function fails because the record is locked, the file pointer is set to the locked record.

Records read in a file open for input only are not locked. Furthermore, most file systems do not block the reading of locked records in a file open for input (this feature depends on the host file system - not all support it). Records read from a file open for I/O are automatically locked unless the external variable "f-no-lock" is set to a non-zero value, in which case they are treated in the same manner as files open for input.

R-NEXT-FUNCTION (op-code 5)

This routine reads the next record in the sequence of records in a relative file. The R-NEXT-FUNCTION routine has two parameters: *file_handle* and *record_area*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Record_area points to the area to hold the record read. It must be at least MAXSIZE bytes in length.

If R-NEXT-FUNCTION succeeds, RETURN-CODE is set to the size of the record read, plus one. If it fails, RETURN-CODE is set to zero. However, if the function fails because the record is locked, the file pointer is set to the locked record.

Records read in a file open for input only are not locked. Furthermore, most file systems do not block the reading of locked records in a file open for input (this feature depends on the host file system - not all support it). Records read from a file open for I/O are automatically locked unless the external variable "f-no-lock" is set to a non-zero value, in which case they are treated in the same manner as files open for input.

R-PREVIOUS-FUNCTION (op-code 6)

This routine reads the previous record in the sequence of records in a relative file. The R-PREVIOUS-FUNCTION routine has two parameters: *file_handle* and *record_area*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Record_area points to the area to hold the record read. It must be at least MAXSIZE bytes in length.

If R-PREVIOUS-FUNCTION succeeds, RETURN-CODE is set to the size of the record read, plus one. If it fails, RETURN-CODE is set to zero. However, if the function fails because the record is locked, the file pointer is set to the locked record.

Records read in a file open for input only are not locked. Furthermore, most file systems do not block the reading of locked records in a file open for input (this feature depends on the host file system - not all support it). Records read from a file open for I/O are automatically locked unless the external variable "f-no-lock" is set to a non-zero value, in which case they are treated in the same manner as files open for input.

R-START-FUNCTION (op-code 7)

This routine positions the file pointer for the next R-NEXT-FUNCTION or R-PREVIOUS-FUNCTION. The R-START-FUNCTION routine has three parameters: *file_handle*, *keyval*, and *mode*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Keyval is the record position at which to start. It is a long value.

Mode is a direction. Valid values are as defined in "filesys.def". They include:

F-EQUALS (0)	start at the specified key
F-NOT-LESS (1)	start at the specified key, or the one after
F-GREATER (2)	start at the record beyond the specified key

F-LESS (3)	start at the record before the specified key
F-NOT-GREATER (4)	start at the specified key, or the one before

The F_EQUALS mode is usually used to test for the existence of a record or to position a file when the key value is known. The F_NOT_LESS and F_GREATER modes are used to position the file for a series of NEXT calls and the F_LESS and F_NOT_GREATER modes are used to prepare for a series of PREVIOUS calls.

After a successful START, the current key of reference is set to keyval. The next READ, NEXT or PREVIOUS call returns the record selected by the START routine. Note that in this case, READ, NEXT and PREVIOUS will all return the same record.

If the START routine fails, the current key of reference is placed in the "undefined" state.

Some file systems cannot perform the F_LESS or F_NOT_GREATER modes. On these file systems, specifying these modes causes START to return an error and set the E_NO_SUPPORT condition.

R-WRITE-FUNCTION (op-code 8)

This routine adds a new record to the file. It has four parameters: *file_handle*, *record*, *length*, and *keyval*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Record is the record data to write to the file.

Length is the number of bytes to write (for variable-length files).

Keyval is the record number to write. A keyval of "-1" means to write at the end of the file.

The R-WRITE-FUNCTION routine does not change the current file position.

R-REWRITE-FUNCTION (op-code 9)

This routine replaces an existing record in the file. It has four parameters: *file_handle*, *record*, *length*, and *keyval*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Record is the record data to write to the file.

Length is the number of bytes to write (for variable-length files).

Keyval is the record number to write. A value of "-1" is invalid with the REWRITE function.

The R-WRITE-FUNCTION routine does not change the current file position.

R-DELETE-FUNCTION (op-code 10)

This routine deletes the specified record. It does not affect the current file position. The DELETE routine has two parameters: *file_handle* and *keyval*.

File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

Keyval specifies the position of the record to delete.

R-UNLOCK-FUNCTION (op-code 11)

This routine unlocks any locked records held by the current process in the specified file. It is not an error to call this routine when there are no locked records. This routine does not affect the current file position. This routine will not unlock any records if it is called during a transaction. "Commit" should be used instead.

R-UNLOCK-FUNCTION has only one parameter: *file_handle*. File_handle must be a valid file handle returned by R-OPEN-FUNCTION.

SYSTEM

The SYSTEM library routine provides a method of executing an operating system command.

Usage

```
CALL "SYSTEM"
    USING MY-COMMAND-LINE,
    GIVING EXIT-STATUS
```

Parameters

MY-COMMAND-LINE PIC X(n)

Contains the operating system command line to execute.

EXIT-STATUS Any numeric data item

Returns the called program's exit status.

Comments

The SYSTEM routine takes a parameter, which is submitted to the host operating system as if it were a command typed in from a terminal.

Some operating systems place limits on the length of a command-line string. Under Windows, the limit is 128 bytes. When you issue a SYSTEM call using a variable, make sure that the length of the variable doesn't exceed the operating system's limit.

The user's terminal is set to its default operating state before this command is run and is reset after it's complete. The runtime system waits for the command to complete.

Note: On Windows systems, if you append an ampersand (&) character to the command line, the program will run asynchronously. This should not be done for programs providing input files, but is often useful for programs processing output files.

The status of a call to SYSTEM is placed into EXIT-STATUS. This is usually the exit status of the executed program, or is "-1" if the SYSTEM routine failed.

Here's an example of a call to SYSTEM. On a UNIX machine, you could display a directory listing of the "/usr" directory with the following command:

```
CALL "SYSTEM" USING "ls /usr"
```

If your machine is running Windows and you want to execute MS-DOS operating system commands via SYSTEM, you must pass the name COMMAND.COM, as well as the operating system command. Use the syntax shown in this example that executes the DIR command:

```
CALL "SYSTEM" USING
     "COMMAND.COM /C DIR"
```

When CALL "SYSTEM" is used to initiate a program, it looks only for files with a ".EXE" extension. If you want to call a ".COM" or ".BAT" file, you must explicitly add that extension in your code. For example:

```
CALL "SYSTEM" USING
     "COMMAND.COM /C MYBATCH.BAT"
```

The SYSTEM routine is provided in source form as a sample of a C subroutine.

Note: This routine causes ACUCOBOL-GT to *forget* the contents of the user's screen. This is done because the command executed may display information on the screen that ACUCOBOL-GT is not aware of. Because of this, *pop-up* windows made after a call to the SYSTEM routine may not correctly restore the screen contents when they are closed. You can avoid this problem by re-initializing the screen after you call the SYSTEM routine. You can do this by erasing the screen or by closing a pop-up window that covers the entire screen.

If the command to be executed will not perform any screen I/O, then you can request the SYSTEM routine to retain ACUCOBOL-GT's memory of the user's screen. This will avoid the problem mentioned in the preceding

paragraph. To do this, simply pass a second argument to the SYSTEM routine. This may be any parameter you choose. For clarity, we suggest that the second argument be the literal “NO I-O”.

S$IO

The S$IO routine provides an interface to the sequential file handler. Calls to the routine require an operation code and a variable number of parameters, depending on the operation called. The return code is set automatically after the call. The external variable “F-ERRNO” is set according to any errors found. “F-ERRNO” may not be reset on entry to S$IO, and should be checked only if S$IO returns an error condition.

Note: File locking should be applied whenever SEEK operations are used.

Usage

```
CALL "S$IO"
    USING OP-CODE, PARAMETERS
```

Parameters

OP-CODE Numeric parameter

Specifies the file handling routine to be performed. The following table lists each op-code and its corresponding operation. The operations are detailed below:

Code	Operation
1	S-OPEN-FUNCTION
2	S-CLOSE-FUNCTION
3	S-MAKE-FUNCTION
4	S-READ-FUNCTION
5	S-WRITE-FUNCTION
6	S-REWRITE-FUNCTION
7	S-SEEK-FUNCTION

PARAMETERS variable depending on op-code

The remaining parameters vary depending on the operation selected. They provide information and hold results for the operations specified. Parameters may be omitted from those operations that do not require them. The parameters are detailed in the section titled “Description” below.

Description

All parameters passed to S$IO are passed by reference. This applies even to parameters that are integer values in the corresponding file handling routines.

Except for the MAKE function, S$IO automatically terminates PIC X parameters with a LOW-VALUES byte.

You do not have to specify SYNC for level 01 or level 77 parameters because they are automatically synchronized by ACUCOBOL-GT.

The “filesys.def” COPY file contains many useful definitions for use with S$IO. It contains definitions for the S$IO op-codes as well as the “F-ERRNO” error values. It also incudes many useful pre-declared variables that are of the proper type and usage.

Note: The behavior of this routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the documentation on **FILENAME_SPACES** in Appendix H for information about the terminating character for path names.

The runtime configuration variables FILE_PREFIX and FILE_SUFFIX are ignored by the S$IO routine.

OP-CODES and PARAMETERS

S-OPEN-FUNCTION (op-code 1)

This routine opens an existing sequential file. If it is successful, the value in RETURN-CODE should be moved to a data item that is USAGE HANDLE. This data item is passed as the open file handle to the other file handling routines. If it fails, RETURN-CODE is set to a NULL value.

The S-OPEN-FUNCTION routine has four required parameters, and three optional parameters: *name*, *mode*, *recsize*, *type*, *blocking*, *padding*, *pipe_name*.

Name is the name of the file to open. It need not be null-terminated.

Mode is one of the following values (defined in "filesys.def"):

Finput	open for input only
Foutput	open for output only
Fio	open for input and output
Fextend	open for output only (same as Foutput)

"Foutput" does not delete the current file (this behavior differs from the OPEN OUTPUT statement in COBOL).

Note: This routine only opens files that already exist. If the file does not exist, the routine fails, even when opening with mode "Foutput".

Mode may furthermore have one of the following flags added to it to indicate file locking options (defined in "filesys.def"):

Fread_lock	locks file against other updaters
Fwrite_lock	locks file against all others
Fmass_update	locks file against all others (same as Fwrite_lock)

If "Fmass_update" is used, the file system is directed to emphasize speed of updates over file security.

Additionally, "Fmulti_lock" may be added to *mode* to request that more than one record lock be maintained in the file by this process. If this option is not specified, then any I/O operation on the file will first release any currently locked record. This results in only one record lock being set in the file at any time. When this option is used, locked records are released only when the file is closed or when the UNLOCK routine is called.

Recsize is the maximum size of each record. This must be known at open time.

Type is the type of sequential file, and is one of the following values (defined in "filesys.def"):

S-FIXED	fixed record binary sequential file
S-VAR-COUNT	variable record length binary sequential file
S-LINE	line sequential file
S-PRINT	line sequential file

Blocking is the size of a block, in bytes. This parameter is optional and defaults to 0, meaning that a block is the size of a record.

Padding is the value of the pad character for filling short blocks. This parameter is optional and defaults to 0, meaning that any short blocks are padded with a binary 0 value.

Pipe-name is the name of the pipe to open instead of a file. This parameter is optional. It only has an effect on UNIX machines.

S-CLOSE-FUNCTION (op-code 2)

This routine closes an open file. It also removes currently held locks on the file. S-CLOSE-FUNCTION has only one parameter, *f*, a file handle returned by the S-OPEN-FUNCTION. For some file systems, it is possible that S-CLOSE-FUNCTION will write additional records that had been previously buffered by the system. For this reason, it is possible that a "disk full" condition can occur.

S-MAKE-FUNCTION (op-code 3)

This routine creates a new sequential file. It will overwrite any existing file of the same name. However, it will not overwrite a file that is currently in use. If the file is in use, the error E_FILE_LOCKED is returned.

This routine does not automatically terminate its parameters with LOW-VALUES. You must ensure that its parameters are correctly terminated.

The S-MAKE-FUNCTION routine takes two parameters: *name* and *l_parms*.

Name contains the name of the file. The name need not be null-terminated.

L-parms is a string that describes various logical characteristics of the file. The string consists of three numeric fields separated by commas. The string must be null-terminated. This parameter is optional. If it is not specified, the values are not known to the runtime. The fields are as follows:

1. *Maximum record size*. This is the size in bytes of the largest record to be placed in the file. This can range from 1 to 67,108,864.
2. The *file type* (accepted values are the same as described in S-OPEN-FUNCTION). This must be a single byte containing a binary value that indicates the type of the file.
3. *Block size*. This is the size of a block of records.

S-READ-FUNCTION (op-code 4)

This routine reads the next record in the sequence of records. The S-READ-FUNCTION routine has two parameters, *f* and *record*.

F must be a valid file handle returned by S-OPEN-FUNCTION.

Record points to the area to hold the record read.

If S-READ-FUNCTION succeeds, RETURN-CODE is set to the size of the record read, plus one. If it fails, RETURN-CODE is set to zero. However, if the function fails due to the record being locked, the file pointer is set to the locked record.

Records read in a file open for input only are not locked. Furthermore, most file systems do not block the reading of locked records in a file open for input (this feature depends on the host file system - not all support it). Records read from a file open for I/O are automatically locked unless the external variable “f-no-lock” is set to a non-zero value, in which case they are treated in the same manner as files open for input.

S-WRITE-FUNCTION (op-code 5)

This routine adds a new record to the named file. It has four parameters: *f*, *record*, *size*, and *cr_cntrl*.

F must be a valid file handle returned by S-OPEN-FUNCTION.

Record points to the record to add.

Size is the size of the record. If size is zero, then the maximum record size for the file is used.

Cr_cntrl is the number of lines to advance before writing the new record. This is only valid for print files.

The S-WRITE-FUNCTION routine does not change the current file position.

S-REWRITE-FUNCTION (op-code 6)

This routine replaces an existing record in the file. It has three parameters: *f*, *record*, and *size*.

F must be a valid file handle returned by S-OPEN-FUNCTION.

Record points to the new record to place in the file.

Size is the size of the record. It may be zero to indicate the maximum record size for the file. The size of the new record need not match the size of the existing record.

The S-REWRITE-FUNCTION routine does not affect the file position.

S-SEEK-FUNCTION (op-code 7)

This routine changes the current position of the file for subsequent READs.

The only time the runtime supports a seek in a sequential file is 1) to find the last record in order to read the file backwards, and 2) after each READ to seek to the previous record. *These are the only uses of SEEK supported by this library.* The file must be locked to ensure that other users of the file don't add to it.

S-SEEK-FUNCTION has three parameters: *f, offset* and *mode*.

F must be a valid file handle returned by OPEN.

Offset is the number of bytes to move from the current file position. Offset is limited to a 32-bit value.

Mode is a flag that specifies the position from which the offset is measured. A value of zero sets the position at the beginning of the file, regardless of the current position. A value of one sets the position to the current position. A value of two sets the position to the end of the file. Note that offset can be negative value.

W$BITMAP

This routine is a collection of related operations that handle bitmapped (BMP and JPEG) images. Only Windows machines can actually display bitmaps. On all other machines, this routine returns an error code.

Usage

```
CALL "W$BITMAP"
    USING OP-CODE, parameters,
    GIVING BITMAP-HANDLE
```

Parameters

OP-CODE Numeric parameter

Selects the W$BITMAP operation to perform. The file “acugui.def” contains level 78 symbolic names for these operations. Unless otherwise noted, these operations can be used in a thin client environment. The specific operations are described below.

parameters Vary depending on the op-code chosen.

BITMAP-HANDLE PIC S9(9) COMP-4 (or COMP-5)

BITMAP-HANDLE holds the return value of W$BITMAP. Values less than or equal to zero indicate errors. If you are loading or destroying ImageLists, this should be a COMP-5 field.

Description

W$BITMAP can be used to display a bitmapped image, load a bitmapped image into memory, or remove a bitmapped image and free its memory. You can use this routine to load a bitmapped image into memory as a Windows API data type called an "ImageList", which treats the bitmap file as a series of fixed-width images. This provides a simplified way to load and destroy ImageLists when using the thin client. This routine can be used to capture screen shots of an active window or desktop. This routine can also be used to load IPictureDisp objects.

When it is trying to locate a bitmap file, W$BITMAP will search first for a *resource* with the specified name, and then for a disk file. Resources are named in a fashion similar to disk files, but without any directory information. (See the COPY RESOURCE statement in section 2.4.1 of Book 3, *ACUCOBOL-GT Reference Manual* for more information about including resources.)

Note: When you are running in a thin client environment, and a file name beginning with "@[DISPLAY]" is passed to this routine, it will attempt to access the file in the display host's file system. It does not download the file from the server. For more information, refer to section 7.2, "Using Library Routines and DLLs in Thin Client," in the *AcuConnect User's Guide*.

W$BITMAP will examine files to determine the type of image format. If the file suffix is ".jpg", ".jpe" or ".jpeg", W$BITMAP assumes the file is a JPEG image. Otherwise, it assumes the files are in BMP format. In order to read JPG files, you must have the file "ajpg32.dll" installed in the runtime directory. Only 32-bit runtimes support JPEG format images. If you need JPEG support on 64-bit Windows, run the 32-bit runtime or the Thin Client. You can also run the Thin Client with the 64-bit runtime.

W$BITMAP loads 24-bit color or 8-bit grayscale images in the JPEG File Interchange Format (JFIF). It reads baseline and extended DCT sequential and progressive files that are Huffman encoded (file types SOF0, SOF1, SOF2). The JPG lossless mode format is not supported.

Note: The behavior of this library routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the documentation on **FILENAME_SPACES** in Appendix H.

OP-CODES and PARAMETERS

WBITMAP-DISPLAY (op-code 1)

This operation retrieves a bitmapped image from disk and displays it on the screen. This op-code takes four additional parameters:

name This is an alphanumeric literal or data item that is the file name of the bitmap to display. This must be a file that contains a device-independent bitmap (usually called ".bmp" files). Create this file with a bitmap editor. For example, the bitmap contained in the sample program "tour.cbl" was created using the Paint program that comes with Windows. The file name is not interpreted--it should be the exact file name of the image.

row This is a numeric parameter. This value is the row where you want to place the upper-left corner of the image. Row values are treated just as they are in a DISPLAY statement. This means that row "1" is the top row of the current ACUCOBOL-GT window. You may also refer to fractional row positions. For example, row "1.5" is halfway between the top of row "1" and the top of row "2".

column This is a numeric parameter. This identifies the column where the upper-left corner of the image is to be placed. COLUMN is processed in the same manner as ROW.

flags This is a numeric parameter that contains display options. Currently, the only option is WBITMAP-NO-FILL. When this is set, it inhibits the background-fill operation described below. This parameter can be omitted, in which case the fill option behaves as described.

The bitmapped image contained in the named file is loaded into memory, converted to a device-dependent bitmap and displayed at the position indicated by ROW and COLUMN. The entire image in the file is displayed, except that it is truncated to fit in the current ACUCOBOL-GT window.

Note: You should generally try to produce small bitmaps, because full-screen bitmaps occupy significant amounts of memory and can take noticeable time to process. If you use Windows Paint to create the image, you should first use the "Image Attributes" option to set the size of the image (the default is a full page; you'll want to make it smaller).

Frequently, the edges of the image will not exactly match a row and column boundary. ACUCOBOL-GT fills the tiny area between the edge of the bitmap and the next character cell with the current window's background color. If you use the WBITMAP-NO-FILL parameter, then the runtime leaves this area alone. Usually, this means that it will show arbitrary data as the program progresses. This can be useful, however, if you are placing several bitmaps very close together and do not want the fill area of one bitmap to overwrite another bitmapped image.

When W$BITMAP returns, it sets BITMAP-HANDLE to a positive (non-zero) value that is the bitmap's *handle*. This should be saved in a variable declared as PIC 9(9). Use the handle when you make future reference to the displayed image (for example, you need the handle in order to remove the bitmap and free the memory that it occupies). If BITMAP-HANDLE is less than or equal to zero, then an error has occurred. (See the section on Error Handling below.)

You may display data and pop-up windows over bitmapped images. However, the current implementation may cause the image to *flash through* the data displayed on top when the runtime is repainting the screen. The final image should be correct, but the flash can be annoying in some cases. To

avoid this, either do not display anything over the image, or display over the entire image (the runtime will not try to display the image if it is entirely hidden).

The effect of displaying overlapping bitmapped images is undefined.

WBITMAP-DESTROY (op-code 2)

This operation removes a bitmapped image from the screen and frees the memory used by that bitmap. It takes only one parameter, the handle of the bitmap returned by WBITMAP-DISPLAY. This should be either USAGE COMP-4 or unsigned DISPLAY.

When an image is removed from the screen, it's replaced by spaces using the current window's background color. Only those parts of the screen that are currently showing the image are updated.

You can effectively remove an image from the screen by displaying over it. However, this does *not* free the memory used by the image. The runtime also spends some time whenever it updates the screen determining whether or not the image is visible. For these reasons, you should destroy images when you are done with them.

The runtime automatically destroys all remaining images when it shuts down.

WBITMAP-LOAD (op-code 3)

This operation loads bitmapped images from disk into memory so that they can be displayed with bitmapped buttons.

Note: You may not use remote name notation with this operation.

This op-code takes three additional parameters. They are, in order:

name A literal or data item containing the name of the file to load. The length of the *name* (including path) should not exceed 90 characters. This limit may vary on different operating systems.

bitmap-handle A PIC S9(9) COMP-4 data item that stores a handle to the bitmapped image loaded into memory.

flags A numeric parameter that contains loading options. This parameter can be omitted.

Currently the only option is WBITMAP-NO-DOWNLOAD. This option is intended for programs that are deployed with the ACUCOBOL-GT Thin Client. When the option is set, the server does not download the specified bitmap to the client. The flag indicates to the server that the bitmap is already in the client's cache directory. To properly use the option, the program must keep track of whether the bitmap is already in the client's cache.

If the operation is successful, *bitmap-handle* holds a positive value. If *bitmap-handle* is "0" or negative, an error occurred.

If you have bitmapped images in more than one file, you need to load each file before using the images they contain. Be sure to store separate handles for each file loaded.

When you are done with an image and have destroyed all the buttons that reference that image, you can remove it from memory with the WBITMAP-DESTROY operation. Do not destroy an image that is referenced by an active control or print job, because the results are unpredictable.

WBITMAP-LOAD-IMAGELIST (op-code 5)

This operation works in the same way as the WBITMAP-LOAD operation, except that the returned handle is a Win32 API ImageList handle instead of a bitmap handle. This handle can be directly used by ActiveX controls which use ImageLists, and should be USAGE COMP-5. When running under the thin client, the resulting ImageList resides on the desktop machine. The ImageList handle is only meaningful to software components on that machine (such as a displayed ActiveX control). Unlike images loaded with WBITMAP-LOAD, the runtime does not track ImageLists. For this reason, you should destroy an ImageList once you no longer need it. Error values

returned by this operation are identical to those returned by the WBITMAP-LOAD operation. This operation takes up to three additional parameters:

name This required alphanumeric literal is the name of the file you wish to load. It should be a BMP or JPG file. The file is handled in exactly the same fashion as it is for the WBITMAP-LOAD operation.

width This numeric parameter is optional. The bitmap file is treated as a series of distinct images arranged horizontally. The height of each image is determined by the height of the bitmap. The width of each image is defined by this parameter, expressed in pixels. Each image must be the same width. If omitted, a default width of 16 pixels (a common width for toolbar images) is used.

transparent-color This optional numeric literal is set to the value of a color that you designate as a "transparent" color. Pixels of this color are not displayed when the bitmap is drawn, allowing the background to show through. Color values are expressed as integers composed of red, green and blue components. The mathematical expression for the composite value is:

(blue * 65536) + (green * 256) + (red)

where red, green, and blue are values between 0 and 255. You can use a hexadecimal numeric literal to express this value since each component of the color occupies one byte in a binary value. For example, "x#FF8000" expresses an orange color with red at 255, green at 128, and blue at 0. The first byte (x 'FF') is the red value, the second byte (x'80') is the green value, and the last byte (x'00') is the blue value.

Note: The value "x#C0C0C0" is often used for the transparent color since this is the medium gray shade used in default Windows color schemes. Images drawn using this as a "background" color adapt well when the user changes from the default color scheme.

If this parameter is omitted, all colors are treated as usual and the image is drawn without modification.

WBITMAP-DESTROY-IMAGELIST (op-code 6)

This operation destroys an ImageList. Normally this will be an ImageList created by the WBITMAP-LOAD-IMAGELIST operation, but it may be any ImageList that resides on the desktop machine. When running under the thin client, the ImageList is destroyed on the desktop machine. This operation takes a single parameter, the handle of the ImageList to destroy. This should be USAGE COMP-5. Because the runtime cannot verify this parameter, you should take extra care to ensure that it is valid.

WBITMAP-CAPTURE-IMAGE (op-code 7)

This operation captures a screen shot of a window, and stores it as a BMP file. (If you are using this operation in a thin client environment, refer to section 7.2.5 of the *AcuConnect User's Guide* for special considerations.) This operation takes four additional parameters:

name This required alphanumeric literal is the name of the file in which the image is saved. The *name* may include embedded spaces. This parameter is not affected by the FILE_PREFIX configuration variable. If a file of the specified name already exists, it is overwritten unless it is read-only. If this parameter is omitted or set to spaces, the image is saved to the Windows clipboard.

window-handle This optional PIC S9(9) COMP-4 data item is the Windows system handle of the image to capture. If this parameter is omitted or set to "0", the currently active window is captured. (Active means the window that responds to a keystroke.) If no *window-handle* is available or there is no currently active window, an image of the desktop is captured.

Note: The Windows system handle is not the same as the ACUCOBOL-GT window handle. You may determine the value of *window-handle* using the following code:

```
DISPLAY WINDOW HANDLE IN acu-handle.
```

or

```
DISPLAY ENTRY-FIELD HANDLE IN acu-handle.
INQUIRE acu-handle SYSTEM HANDLE IN window-handle.
```

where *acu-handle* is the handle of a window or control, and *window-handle* is a PIC 9(9) COMP-5 or PC X(4) COMP-N data item.

client This optional numeric data item is used to specify whether to capture the entire window, or just the interior of the window. If you specify this parameter, you must also specify *window-handle*. This parameter may have one of two values:

- 0 (default) The entire window is captured, including the title bar, window frame, menu, etc.
- 1 Only the interior of the window specified by *window-handle* is captured. This can be useful if you are displaying a scanned image, and want to capture only that image, not the window frame.

colordepth This optional numeric data item sets the number of color bits per pixel used when capturing the image. If this value is not specified, the screen default is used. The default color density of a screen is a property of the video adapter driver. On most modern PCs this is set to 24-bits per pixel. This parameter is not used when images are saved to the clipboard. This parameter may be set to one of the following values:

- 1 Monochrome
- 4 4-bits per pixel
- 8 8-bits per pixel (256 colors)
- 16 16-bits per pixel
- 24 24-bits per pixel (True color)
- 32 32-bits per pixel (True color)

When making copies of all or part of the screen, do not use higher color density than necessary. Color is memory intensive and a higher color density does not improve image quality as much as it results in a larger file size. Consider what you will use the image for and set *colordepth* accordingly. Many tools do not support bitmaps with more than 256 colors. A color density setting of "8" is adequate for most situations.

WBITMAP-CAPTURE-DESKTOP (op-code 8)

This operation captures a screen shot of the entire desktop and stores it as a BMP file. (If you are using this operation in a thin client environment, refer to section 7.2.5 of the *AcuConnect User's Guide* for special considerations.) This operation takes two additional parameters:

name This required alphanumeric literal is the name of the file in which the image is saved. The *name* may include embedded spaces. The length of *name* (including path) should not exceed 90 characters. This limit may vary on different operating systems. This parameter is not affected by the FILE_PREFIX configuration variable. If a file of the same name already exists, it is overwritten unless it is read-only. If this parameter is omitted or set to spaces, the image is saved to the Windows clipboard.

If you plan to use this operation in a thin client environment, refer to section 7.2.5 of the *AcuConnect User's Guide* for special considerations.

colordepth This optional numeric data item sets the number of color bits per pixel used when capturing the image. If this value is not specified, the screen default is used. The default color density of a screen is a property of the video adapter driver. On most modern PCs it is

24-bits per pixel. This parameter is not used when images are saved to the clipboard. This parameter may be set to one of the following values:

1	Monochrome
4	4-bits per pixel
8	8-bits per pixel (256 colors)
16	16-bits per pixel
24	24-bits per pixel (True color)
32	32-bits per pixel (True color)

When making copies of all or part of the screen, do not use higher color density than necessary. Color is memory intensive and a higher color density does not improve image quality as much as it results in a larger file size. Consider what you will use the image for and set *colordepth* accordingly. Many tools do not support bitmaps with more than 256 colors. A color density setting of "8" is adequate for most situations.

WBITMAP-CAPTURE-CLIPBOARD (op-code 9)

This operation saves the current content of the Windows clipboard as a BMP file. If the clipboard is empty or contains an unsupported datatype, such as text, an error is returned. (If you are using this operation in a thin client environment, refer to section 7.2.5 of the *AcuConnect User's Guide* for special considerations.) This operation takes only one parameter, the filename to which the clipboard content must be saved.

name This required alphanumeric literal is the name of the file in which the image is saved. The *name* may include embedded spaces. The length of *name* (including path) should not exceed 90 characters. This limit may vary on different operating systems. This parameter is not affected by the FILE_PREFIX configuration variable. If a file of the specified name already exists, it is overwritten unless it is read-only.

WBITMAP-LOAD-PICTURE (op-code 10)

This operation accepts a filename and returns a handle to an IPictureDisp object. This is similar to the Visual Basic LoadPicture function. You must use the "acuclass.def" file to obtain the definition of the IPictureDisp object.

An IPictureDisp object uses the IPictureDisp interface to expose its properties through Automation. It provides a subset of the functionality available through IPicture methods. For more information, go to: msdn.microsoft.com/library and search for IPictureDisp and/or IPicture.

The following image formats are supported:

Bitmaps & device independent bitmaps (DIB)	.bmp
Icons	.ico
Joint Photographic Experts Group (JPEG)	.jpg
Windows MetaFile	.wmf
Graphics Interchange Format	.gif

This operation takes one parameter:

name This required alphanumeric literal is the name of the file you want to load. The filename must include an absolute path. The length of *name* (including path) should not exceed 90 characters.

If the operation is successful, *bitmap-handle* holds a positive value. If *bitmap-handle* is "0" or negative, an error occurred. Note that the runtime does not manage memory consumed by this function. When you are done with an image, you should remove it from memory with a "DESTROY IPictureDisp_handle" statement. See Example 3, below.

Note: The IPictureDisp handle will not accept negative values. If an error (a negative value) is returned, you must use a REDEFINES, or move the value to a signed numeric variable, in order to read the error code. See Example 3 below.

Error Handling

If the value of BITMAP-HANDLE is “0” or negative, an error has occurred. These errors are defined in “acugui.def”.

WBERR-UNSUPPORTED (value “0”) -- The system does not support bitmapped images. Currently, ACUCOBOL-GT supports the display of bitmaps on Windows systems only.

WBERR-FILE-ERROR (value “-1”) -- A file error occurred when trying to open *name*. The most common cause is that *name* does not exist. Other possibilities include a permissions error or running out of file handles.

WBERR-NO-MEMORY (value “-2”) -- The system ran out of memory trying to allocate space for the image.

WBERR-NOT-BITMAP (value “-3”) -- The named file does not contain a device-independent bitmap.

WBERR-FORMAT-UNSUPPORTED (value “-4”) -- The format of the current image is not supported.

WBERR-MISSING-DLL (value “-5”) -- The runtime has attempted to load a “.jpg” file and the file “ajpg32.dll” cannot be found.

WBERR-INVALID-HWND (value “-6”) -- The runtime cannot capture the current image. The window handle provided was invalid.

WBERR-INVALID-DATA (value “-7”) -- The runtime cannot access the bitmap object. This can happen if the window has been closed, or if the image is corrupt.

WBERR-INVALID-CLIPBOARD (value “-8”) -- The runtime has been denied access to the clipboard. This may be because another application has locked the clipboard.

WBERR-INVALID-PALETTE (value “-9”) -- The runtime is not able to create the palette for this image. The most common cause is that the image is corrupt or has a palette size that is not supported.

Example 1

The following example shows the first two op-codes in use. After the example, the parameters are described. For an example of the WBITMAP-LOAD operation, see Example 3.

Notice the two calls to W$BITMAP in the sample code. First we use W$BITMAP to display a logo, and then we use the DISPLAY verb to place two lines of text on the screen. The second call to W$BITMAP removes the logo from the screen and releases the memory it occupies.

```
* Displays the logo for 2 seconds.

display window, line 7, column 26, lines 9, size 30, color black +
   bckgrnd-white, erase, shadow, no scroll, pop-up area is window-1.

* Now we give the name of the bitmap file and
* the row and column of the upper left corner:

call "w$bitmap" using wbitmap-display, "sample/acucob85.bmp", 2, 5.

* We save the handle so we'll have it when we want
* to remove the image: move return-code to bitmap-handle.

* Check to make sure the logo file was found and
* that no error occurred:

if bitmap-handle <= zero
   close window window-1
else
   display "Copyright (c) 1985 - 2000", line 7, size 30, centered
   display "Acucorp, Inc.", line 8, size 30, centered
   accept single-char, auto, before time 200
end-if
close window window-1

* We remove the bitmapped image and release the
* memory it uses:

call "w$bitmap" using wbitmap-destroy, bitmap-handle.
```

Example 2

Following are several examples of different W$BITMAP calls to capture screen shots and desktop images:

```
* This call captures the standard active window as a file
```

```
* named “myfile.bmp”
call "W$BITMAP" using WBITMAP-CAPTURE-IMAGE “myfile.bmp”.

* This call captures the standard active window onto
* the Windows clipboard
call "W$BITMAP" using WBITMAP-CAPTURE-IMAGE “ ”.

* This call captures the client area of the active
* window as a file named “myfile.bmp” using 8-bit color
call "W$BITMAP"
   using WBITMAP-CAPTURE-IMAGE “myfile.bmp” 0 1 8.

* This call captures the indicated window handle using 16-bit
* color and saves it as a file named “myfile.bmp”
inquire mycontrol system handle in hWND
call "W$BITMAP"
   using WBITMAP-CAPTURE-IMAGE “myfile.bmp” hWND 0 16.

* This call captures the entire desktop as a file named
* “myfile.bmp” using 32-bit color
call "W$BITMAP" using WBITMAP-CAPTURE-DESKTOP “myfile.bmp” 32.

* This call captures the current bitmap content of the Windows
* clipboard as a file named “myfile.bmp”
call "W$BITMAP" using WBITMAP-CAPTURE-CLIPBOARD “myfile.bmp”.
```

Example 3

The following sample program calls W$BITMAP to load an IPicture object.

```
       ...

SPECIAL-NAMES.
   COPY "acuclass.def".
     .

WORKING-STORAGE SECTION.
   COPY "acugui.def".
     77  myIPictureDisp     HANDLE OF IPictureDisp.
     77  myErrorTest REDEFINES myIPictureDisp PIC S9(9) COMP-5.

   ...
PROCEDURE DIVISION.
   MAIN-001.

  CALL "W$BITMAP" USING WBITMAP-LOAD-PICTURE
```

```
       "C:\MyDir\MyBitmaps.bmp"
        GIVING myIPictureDisp.

EVALUATE myErrorTest
     WHEN WBERR-FILE-ERROR
        DISPLAY MESSAGE BOX
          "File not found"
          TITLE "Error"
          INITIALIZE myIPictureDisp
     WHEN WBERR-FORMAT-UNSUPPORTED
        DISPLAY MESSAGE BOX
          "Format not supported"
           TITLE "Error"
           INITIALIZE myIPictureDisp
     WHEN OTHER
          DISPLAY MESSAGE BOX
          "File successfully loaded"
          TITLE "Success"
END-EVALUATE.
DESTROY myIPictureDisp.
GOBACK.
```

W$BROWSERINFO

The W$BROWSERINFO routine provides information about a requesting Web browser. This routine is used in conjunction with the ACUCOBOL-GT Web Runtime. See the book titled *A Programmer's Guide to the Internet* for more information about using the Web Runtime.

Usage

```
CALL "W$BROWSERINFO"
    USING BROWSERINFO-DATA
```

Parameters

BROWSERINFO-DATA Group item as follows:

```
01 BROWSERINFO-DATA.
   03 USER-AGENT-STRING       PIC X(50).
   03 BROWSWER-MAJOR-VERSION  PIC X COMP-X.
   03 BROWSWER-MINOR-VERSION  PIC X COMP-X.
```

BROWSERINFO-DATA is found in the COPY library "acucobol.def". The values are as follows:

USER-AGENT-STRING This is the browser's user_agent field. It contains the name of the browser as it is sent to the HTTP server. It may also contain version numbers, product name, and operating system name. Netscape browsers set the first seven characters of this field to "Mozilla". Microsoft Internet Explorer sets this field to "Microsoft Internet Explorer".

BROWSER-MAJOR-VERSION This is the major version number reported by the browser. This is not the same as the major version number displayed in the browser's "About" screen. Many browsers simply place a "0" in this field.

BROWSER-MINOR-VERSION This is the minor version number reported by the browser. This is not the same as the minor version number displayed in the browser's "About" screen.

Description

Upon return from W$BROWSERINFO, all of the data elements contained in BROWSWERINFO-DATA are filled in. If you call W$BROWSERINFO when the COBOL application is not running in a Web browser via the Web Runtime, the first field is set to spaces, and the last two fields are set to zero ("0").

W$FLUSH

The W$FLUSH routine causes the screen and/or cursor to be refreshed. It can be used to ensure that the user sees the most current display (see the explanation under Description, below).

Usage

```
CALL "W$FLUSH"
    USING PARAM-NUM
```

Parameters

PARAM-NUM numeric data item (optional)

Specifies what part of the display (screen or cursor) is refreshed.

Description

Calling this routine refreshes the display so that the user sees everything that is current, even if an ACCEPT has not been performed. Normally, the runtime ensures that the information the user sees is correct only when input is required from the user. This means that if you DISPLAY some text or control, and then perform extensive processing without also performing an ACCEPT, the text or control may not be immediately visible to the user. Possible values are:

0 (default) the screen is made current, but the cursor may appear in an arbitrary location. (This is due to output optimization.)

1 the screen and cursor are made current

2 on UNIX machines using the BUFFERED-SCREEN configuration option, the screen is made current at the next clock tick

on all other machines this has the same behavior as setting the value to “1”.

3 on UNIX machines using the BUFFERED-SCREEN configuration option, the screen and cursor are made current at the next clock tick

on all other machines this has the same behavior as setting the value to “1”.

Note: Prior to the introduction of W$FLUSH, some programs worked around this problem using the statement “ACCEPT OMITTED BEFORE TIME 0”.

If you are experiencing a performance penalty for these runtime calls, especially during file processing where the screen is updated after each record, you can choose to inhibit calls to W$FLUSH. This can be useful if you want to simulate “mass update” functionality for ActiveX controls that do not have this capability built in. Use one of the following parameters:

256 inhibits any future calls to W$FLUSH, including those made internally by the runtime system. Only a call with parameter “257” will be honored. This parameter must be turned off to restore user interaction.

257 cancels the inhibited state caused by parameter “256”. This parameter does not cause a flush itself; it just allows future calls to W$FLUSH to function. You must call this parameter before interacting with the user.

For example:

```
78  INHIBIT-FLUSH   VALUE 256.
78  ALLOW-FLUSH     VALUE 257.

CALL "W$FLUSH" USING INHIBIT-FLUSH
PERFORM LARGE-ACTIVEX-UPDATE
CALL "W$FLUSH" USING ALLOW-FLUSH
```

W$FONT

The W$FONT routine provides general support for selecting fonts and determining their characteristics.

Usage

```
CALL "W$FONT"
    USING OP-CODE, FONT-HANDLE, WFONT-DATA
    GIVING WFONT-STATUS
```

Parameters

OP-CODE Numeric value

Selects the W$FONT function to perform. These operations are described below.

FONT-HANDLE USAGE HANDLE or HANDLE OF FONT

When retrieving a font from the system, the system stores a handle to the font in this item. With other operations, this value specifies the font to act on.

WFONT-DATA Group item as follows:

```
01 WFONT-DATA.
   03 WFONT-FACE-DATA.
      05 WFONT-DEVICE                  HANDLE, VALUE NULL.
         88 WFDEVICE-CONSOLE           VALUE NULL.
         88 WFDEVICE-WIN-PRINTER       VALUE 1.
      05 WFONT-NAME                    PIC X(33).
      05 WFONT-CHAR-SET                PIC X COMP-X.
         88 WFCHARSET-DONT-CARE        VALUE 0.
         88 WFCHARSET-DEFAULT          VALUE 1.
         88 WFCHARSET-WIN-OEM          VALUE 2.
         88 WFCHARSET-WIN-SYMBOL       VALUE 3.
         88 WFCHARSET-WIN-SHIFTJIS     VALUE 4.
         88 WFCHARSET-WIN-HANGUL       VALUE 5.
         88 WFCHARSET-WIN-GB2312       VALUE 6.
         88 WFCHARSET-WIN-CHINESEBIG5  VALUE 7.
         88 WFCHARSET-WIN-JOHAB        VALUE 8.
         88 WFCHARSET-WIN-HEBREW       VALUE 9.
         88 WFCHARSET-WIN-ARABIC       VALUE 10.
         88 WFCHARSET-WIN-GREEK        VALUE 11.
         88 WFCHARSET-WIN-TURKISH      VALUE 12.
         88 WFCHARSET-WIN-VIETNAMESE   VALUE 13.
         88 WFCHARSET-WIN-THAI         VALUE 14.
         88 WFCHARSET-WIN-EASTEUROPE   VALUE 15.
         88 WFCHARSET-WIN-RUSSIAN      VALUE 16.
         88 WFCHARSET-WIN-MAC          VALUE 17.
         88 WFCHARSET-WIN-BALTIC       VALUE 18.
      05 WFONT-SIZE                    PIC X COMP-X.
      05 WFONT-BOLD-STATE              PIC X COMP-X.
         88 WFONT-BOLD                 VALUE 1, FALSE 0.
      05 WFONT-ITALIC-STATE            PIC X COMP-X.
         88 WFONT-ITALIC               VALUE 1, FALSE 0.
      05 WFONT-UNDERLINE-STATE         PIC X COMP-X.
         88 WFONT-UNDERLINE            VALUE 1, FALSE 0.
      05 WFONT-STRIKEOUT-STATE         PIC X COMP-X.
         88 WFONT-STRIKEOUT            VALUE 1, FALSE 0.
      05 WFONT-PITCH-STATE             PIC X COMP-X.
         88 WFONT-FIXED-PITCH          VALUE 1, FALSE 0.
      05 WFONT-FAMILY                  PIC X COMP-X.
```

```
          88 WFFAMILY-DONT-CARE          VALUE 0.
          88 WFFAMILY-MODERN             VALUE 1.
          88 WFFAMILY-ROMAN              VALUE 2.
          88 WFFAMILY-SWISS              VALUE 3.
          88 WFFAMILY-SCRIPT             VALUE 4.
          88 WFFAMILY-DECORATIVE         VALUE 5.
    03 WFONT-CHOOSE-DATA.
       05 WFONT-CHOOSE-FLAGS             PIC X COMP-X.
       05 WFONT-CHOOSE-MIN-SIZE          PIC X COMP-X.
       05 WFONT-CHOOSE-MAX-SIZE          PIC X COMP-X.
       05 WFONT-CHOOSE-RED               PIC X COMP-X.
       05 WFONT-CHOOSE-GREEN             PIC X COMP-X.
       05 WFONT-CHOOSE-BLUE              PIC X COMP-X.
       05 WFONT-CHOOSE-COLOR-NUM         PIC X COMP-X.
    03 WFONT-ANGLE                       PIC X(2) COMP-X.
```

WFONT-DATA contains parameters that are used by the various W$FONT operations. The descriptions below detail how each operation uses this data item.

WFONT-STATUS Signed numeric data item

WFONT-STATUS returns the status of the operation. Unless otherwise specified below, a value of “1” indicates success and “0” or a negative value indicates failure.

All of the data items and definitions required by this routine can be found in the COPY library “fonts.def”. To avoid problems should the format of WFONT-DATA change in a future version of ACUCOBOL-GT, it is recommended that you always use the “fonts.def” COPY file.

Description

W$FONT performs a variety of operations depending on the specified op-code. The operations are as follows:

WFONT-SUPPORTED (op-code 1)

This operation returns a value that indicates whether the host system supports W$FONT. If it does not, WFONT-STATUS is set to WFONTERR-UNSUPPORTED (value “0”). This indicates that the host system does not use fonts (as would be normal for a non-graphical host). A

return value of WFONT-FONT-SUPPORT (value “1”) indicates that fonts are supported, but the WFONT-CHOOSE-FONT operation is not available. Finally, a value of WFONT-FULL-SUPPORT (value “2”) indicates that all W$FONT operations are supported on the system.

The FONT-HANDLE and WFONT-DATA parameters are not used with this op-code and should be omitted.

WFONT-GET-FONT (op-code 101)

This operation returns a font on the system that matches the specifications in WFONT-DATA. If a matching font is found, its handle is returned in FONT-HANDLE and WFONT-STATUS is set to “1”. If no matching font is found, FONT-HANDLE is set to NULL and WFONT-STATUS returns the WFONTERR-FONT-NOT-FOUND error condition.

Note: Fonts occupy memory. If you no longer need a font, you can free the memory used by the font by using the DESTROY verb on the font’s handle. The runtime automatically destroys all fonts when it terminates.

In WFONT-GET-FONT, the WFONT-DATA fields are used to describe the desired font. These fields are described in detail below. You should always INITIALIZE the group item WFONT-DATA prior to filling in these fields to ensure that unused fields contain the proper default values.

If you do not specify WFONT-NAME, WFONT-CHAR-SET, or WFONT-SIZE, the corresponding aspect of the font will be arbitrary. In general, you should always specify WFONT-NAME and WFONT-SIZE to get useful results.

WFONT-GET-CLOSEST-FONT (op-code 102)

This operation is identical to WFONT-GET-FONT, except that it doesn’t fail if it cannot find the specified font. Instead, it returns the closest matching font. Note that only Windows systems are capable of performing this operation. On other systems, this call is treated as being identical to WFONT-GET-FONT. You should specify WFONT-CHAR-SET with this function to ensure that the returned font is useful. Also, specifying WFONT-FAMILY can help in locating a suitable font.

WFONT-DESCRIBE-FONT (op-code 106)

This operation returns the characteristics of the font described by FONT-HANDLE in WFONT-DATA. All of the fields contained in the subgroup WFONT-FACE-DATA are returned, within the capabilities of the host system.

WFONT-CHOOSE-FONT (op-code 2)

This operation presents the user with a dialog box that allows for the direct selection of a font. If the user cancels the dialog box, W$FONT returns a status of WFONTERR-CANCELLED. Otherwise, W$FONT returns a status of "1" and sets WFONT-FACE-DATA to a description of the chosen font. FONT-HANDLE is not used and should be passed as "NULL". To generate a font handle for the user's choice, take the WFONT-FACE-DATA values returned by this operation and pass them to the WFONT-GET-FONT operation.

Values contained in WFONT-CHOOSE-DATA modify the behavior of the dialog box and return additional information. These fields are described below. You should always INITIALIZE the group item WFONT-DATA prior to filling any fields. This ensures that you have the correct default values for unused fields.

Not all systems that support fonts support this function. See the WFONT-SUPPORTED operation above for details.

WFONT-DATA

The fields of WFONT-DATA are used as follows:

WFONT-DEVICE -- This item identifies the device that a font is associated with. When a font is associated with a particular device, it should not be used on another device, because these results are undefined. Under Windows, fonts can be shared between devices, but the size is incorrect, because Windows fonts are internally stored with their sizes represented in pixels/device-units instead of points. For example, using a screen font that is 15

pixels high on a laser printer (with 300 or 600 dpi) produces tiny letters. All settings of WFONT-DEVICE are machine-dependent except for the following:

WFDEVICE-CONSOLE	This setting (default value "NULL") associates the font with the user's screen.
WFDEVICE-WIN-PRINTER	This setting (default value "1") is meaningful only under Windows systems. It associates the font with the currently selected printer for the Windows spooler. If you are using WIN$PRINTER, this item must be set to "true" before you call W$FONT.

WFONT-NAME -- This item holds the face name of the font. This name is case-sensitive and may contain internal spaces. When asking for a font, you must match its name exactly. If this item is set to spaces, then any font is allowed to match. Under Windows, some common True Type font names are "Courier New", "Times New Roman", and "Arial".

WFONT-SIZE -- This item holds the size, in points, of the font's characters. A value of zero allows for any size font. In that case, the size chosen by the system may or may not be useful. If you are not certain which size to choose, start with "10" and adjust from there.

WFONT-BOLD -- When this item is set to TRUE, the font is a boldface font.

WFONT-ITALIC -- When this item is set to TRUE, the font is italic. Note that under Windows, most controls have difficulty displaying italic fonts correctly.

WFONT-UNDERLINE -- When this item is set to TRUE, the font is underlined.

WFONT-CHAR-SET -- This item defines the character set to use. A font's character set includes the internal representations used for each character. The possible settings are as follows:

WFCHARSET-DONT-CARE	This allows for any character set. When returned by a call to WFONT-DESCRIBE-FONT, it indicates that the font's character set is unknown or does not correspond to one of the following settings. When requesting a specific font, use this setting only if you are certain that the face name you are requesting uses the character set you want. This may be necessary when you are asking for a font that uses a character set other than any of the following (as might be true for some oriental character sets).
WFCHARSET-DEFAULT	This setting corresponds to a host-dependent default character set. This is the character set most commonly used by the host system. Under Windows, this is the ANSI character set. If you are not certain which setting of WFONT-CHAR-SET to use, use this one.
WFCHARSET-WIN-OEM	This setting is meaningful only under Windows. It corresponds to the host hardware's "OEM" character set. For IBM-style PCs, this is the traditional MS-DOS character set. This is the same character set used by the built-in font "TRADITIONAL-FONT".

WFCHARSET-WIN-*name* The following settings are meaningful only in a Windows environment. They correspond to various character sets supported by Windows fonts (where *name* is the supported font name from the list below). You may select the character set directly in the "script" portion of the font chooser dialog box. Note that most fonts only support a subset of these character sets.

WFCHARSET-WIN-SYMBOL

WFCHARSET-WIN-SHIFTJIS

WFCHARSET-WIN-HANGUL

WFCHARSET-WIN-GB2312

WFCHARSET-WIN-CHINESEBIG5

WFCHARSET-WIN-JOHAB

WFCHARSET-WIN-HEBREW

WFCHARSET-WIN-ARABIC

WFCHARSET-WIN-GREEK

WFCHARSET-WIN-TURKISH

WFCHARSET-WIN-VIETNAMESE

WFCHARSET-WIN-THAI

WFCHARSET-WIN-EASTEUROPE

WFCHARSET-WIN-RUSSIAN

WFCHARSET-WIN-MAC

WFCHARSET-WIN-BALTIC

WFONT-STRIKEOUT -- When this item is set to TRUE, the font is struck-out (i.e., has a line running horizontally through the middle of the characters).

WFONT-FIXED-PITCH -- When this item is set to TRUE, the font is fixed-pitch. This means that each character in the font is the same width.

WFONT-FAMILY -- This item describes the look of the font in very general terms. It is used to aid in selecting a font when WFONT-NAME is not specified or not found. No font is ever rejected because it does not match the requested family. Usually this field can be set to WFFAMILY-DONT-CARE, but specifying other choices can be useful when you don't know what fonts are on a system and you want to find one close to a specific font. Some graphical systems do not have the concept of font families. On these systems, the value of this field is ignored. The possible values of WFONT-FAMILY are:

WFFAMILY-DONT-CARE	This setting allows for any font family. When set by WFONT-DESCRIBE-FONT, it indicates that the font family is unknown or does not match any of the following possibilities.
WFFAMILY-MODERN	This setting indicates a font with a constant stroke width. These fonts are typically fixed-pitch, but need not be. Example fonts are "Courier" and "Elite".
WFFAMILY-ROMAN	This setting indicates a font with variable stroke width and serifs. Example fonts include "Times Roman" and "New Century Schoolbook".
WFFAMILY-SWISS	This setting indicates a font with variable stroke width and no serifs. Example fonts include "Helvetica" and "MS Sans Serif".

WFFAMILY-SCRIPT	This setting indicates a font designed to look like handwriting. Example fonts are “Script” and “Comic Sans MS”.
WFFAMILY-DECORATIVE	This setting indicates a font for decorative or novelty purposes. “Old English” is an example of such a font.

WFONT-ANGLE -- This item holds a value specifying the angle at which the font will print. The value can range from the default of “0”, which is the normal horizontal orientation, to “360”, which is the same as “0”. For example, to print at a 45-degree angle, set WFONT-ANGLE to “45”. To print upside down, set it to “180”. Any font supported by the destination printer can be loaded with this setting. This feature works only when printing a font, not when displaying a font on screen.

Note: You should use the WINPRINT-SET-CURSOR operation of W$PRINTER to set the position of the cursor prior to making a WRITE statement using this feature.

The following fields are part of the subgroup WFONT-CHOOSE-DATA and are used only with the WFONT-CHOOSE-FONT operation.

WFONT-CHOOSE-FLAGS -- This item modifies the behavior of WFONT-CHOOSE-FONT based on the following settings. You can set any combination of these by adding together the corresponding values (all contained in “fonts.def”):

WFCHOOSE-FIXED-ONLY	This setting allows the user to select only a fixed-pitch font. Otherwise, any font can be chosen.
WFCHOOSE-INITIALIZE	This setting causes the various fields of the dialog box to be initialized based on the values contained in WFONT-FACE-DATA. Otherwise, the fields will have no initial value.

WFCHOOSE-EFFECTS-OK	This setting causes the special-effects section of the dialog box to appear. This section allows the user to select the following traits: underline, strike-out, and color. If it's not set, then this section does not appear in the dialog box.

WFCHOOSE-ANSI-ONLY

As Windows has evolved, the fonts listing in the W$FONT dialog no longer contains only ANSI fonts. While UNICODE fonts may be used with ACUCOBOL-GT, using them relies on translation back and forth between locales, a process the COBOL programmer has no means of influencing. For COBOL programmers who want to let their users have the liberty of selecting fonts, it is therefore beneficial to limit the fonts to ANSI fonts only.

This setting allows the user to select only ANSI fonts. Otherwise any font can be chosen. This flag can be used in combination with any other flag value.

Note that this flag is for use with the WFONT-CHOOSE-FONT operation only.

Default value is off. Programmers must make an active choice, e.g. turn this flag on, to get the functionality.

Prior to the call to W$FONT set the flag:

```
MOVE WFCHOOSE-ANSI-ONLY
  TO WFONT-CHOOSE-FLAGS
CALL "W$FONT" USING
  WFONT-CHOOSE-FONT,
  NULL, WFONT-DATA
```

If you have used one of the other flags for WFONT-CHOOSE-FLAGS, you can combine it with this new setting by adding instead of moving:

```
ADD WFCHOOSE-ANSI-ONLY TO
  WFONT-CHOOSE-FLAGS
```

WFONT-CHOOSE-MIN-SIZE -- This item, when set to a positive value, specifies the minimum font size (in points) that the user can choose.

WFONT-CHOOSE-MAX-SIZE -- This item, when set to a positive value, specifies the maximum font size (in points) that the user can choose. When it's set to zero, no maximum is established.

WFONT-CHOOSE-RED -- This item is used only when the WFCHOOSE-EFFECTS-OK flag is set. It returns the red component of the color chosen. This is a number in the range of "0" to "255". See the W$PALETTE Routine for ways to use this value. On systems that do not support palettes, this value is always zero.

WFONT-CHOOSE-GREEN -- This item is similar to WFONT-CHOOSE-RED, except it returns the green component of the color.

WFONT-CHOOSE-BLUE -- This item is similar to WFONT-CHOOSE-RED, except it returns the blue component of the color.

WFONT-CHOOSE-COLOR-NUM -- This item is used only when the WFCHOOSE-EFFECTS-OK flag is set. It returns the ACUCOBOL-GT color number (in the range of "1" to "16") if the color chosen by the user corresponds to a color in the current window's palette. If it does not, then this value is set to zero. Non-zero values can be used directly in the COLOR phrase of a screen item to get the correct color.

Error Handling

The following values are returned by various operations when an error occurs. All error values are less than or equal to zero.

WFONTERR-UNSUPPORTED -- This error indicates that the current host system does not use fonts and the W$FONT routine is not supported.

WFONTERR-CANCELLED -- This error indicates that the dialog box used by WFONT-CHOOSE-FONT was canceled by the user.

WFONTERR-FONT-NOT-FOUND -- This error indicates that no font was found that matched the requested font.

WFONTERR-INVALID-HANDLE -- This error indicates that the value in FONT-HANDLE does not correspond to an existing font.

W$FORGET

The W$FORGET routine causes the ACUCOBOL-GT Terminal Manager to reinitialize itself.

Usage

```
CALL "W$FORGET"
```

Description

The exact effects of this routine depend on the host machine, but include such things as setting the current screen image to "unknown," repositioning the cursor to the last line on the screen, and placing the current screen attribute to "unknown." Note that once the current screen image is *forgotten*, any *pop-up* windows that you create afterwards will act as if the original screen image were spaces. This may erase some portions of the screen when you close those windows.

The reasons for calling this routine are fairly specialized. This routine allows the window manager to function correctly when the user's screen has been changed in ways that the window manager is unaware of. Since the window manager keeps an exact image of the user's screen, it must manage every change to the screen itself. If it does not, then you must tell it to *forget* its current screen image. You can use this routine to restart the window manager after a sequence of ANSI-style ACCEPT or DISPLAY statements.

W$GETC

The W$GETC routine retrieves the next keystroke from the user and returns it to the program. The keystroke is not echoed. Use this routine when you require detail management of the keyboard.

Caution: In the current implementation, W$GETC does not interact well with event procedures. You should avoid using W$GETC if you use event procedures at the same time. In this case, you can use single character ACCEPT statements instead.

Usage

```
CALL "W$GETC"
    USING KEY-FOUND
```

Parameter

KEY-FOUND PIC X(2)

Description

W$GETC places the keystroke found into its single parameter. If the character is a single 8-bit value, the first character of the return value is a space and the second character is the character typed. For example, if the user types an “A”, the return value will be a space followed by “A”.

If the key typed is a special character such as a function key or the operating system’s backspace key, W$GETC returns the two-character keycode found in the *ACUCOBOL-GT User’s Guide* **section 4.3.2.3, “Table of keys.”** For example, if the user types function key 5, the return value will be “k5”.

Finally, if an end-of-file condition occurs, then W$GETC returns the two-character sequence “-1”.

W$GETURL

W$GETURL works with the Web Runtime. It tells the runtime to pass a given URL (Uniform Resource Locator) to the host browser. The browser handles this URL as if it were typed into the URL entry field. After a CALL to W$GETURL, subsequent URL requests are ignored until the CALL completes. See the manual *A Programmer’s Guide to the Internet* for a description of the Web Runtime.

Usage

```
CALL "W$GETURL"
    USING URL, TARGET
```

Parameters

URL PIC X(n)

Contains the complete URL. This can be of any type, such as http, ftp, news, mailto, gopher, or javascript.

TARGET PIC X(n)

Represents the destination for displaying the URL. This can be the name of a window or a frame, or one of several special target names. If you specify "_current" or "_self" or "_top", the response data is written to the browser window, and the Web Runtime is unloaded. If you specify "_new" or "_blank", the response data is written to a new browser window.

You can also write the response data to a frame by specifying the frame name as the target parameter.

Comments

After a CALL is made to W$GETURL, subsequent URL requests are ignored until the CALL completes.

Description

Each URL that you pass with the W$GETURL routine contains a protocol and a path, separated by a colon. For example, http://www.microfocus.com/ tells the browser to use the HyperText Transfer Protocol and to contact the Web server "www.microfocus.com" and to ask for the root page (/).

Sending e-mail uses the "mailto" protocol. For example, mailto:support@microfocus.com opens an e-mail message to the user "support" at the machine "microfocus.com".

JavaScript is also supported as a protocol, so you can execute JavaScript sequences that display dialog boxes, create Web pages, build text files, and so forth.

Note: This routine is available only when the calling COBOL program is running in a Web browser window via the ACUCOBOL-GT Web Runtime. The routine is not available to programs run by the standard runtime when the standard runtime is executed by a Web browser. The RETURN-CODE register is set to "1" after a successful call and set to "0" if this routine is unavailable.

$WINHELP

If you have the Microsoft Windows Software Development Kit (SDK), you can access help files created with the SDK help compiler by calling the $WINHELP library routine. You can also access compiled HTML files that have the ".chm" extension. This is a support routine for Windows and Windows NT, and is not portable to other systems.

$WINHELP provides a direct interface to the Microsoft "WinHelp" library routine for files created with the SDK help compiler. This routine allows you to perform various functions using Microsoft Help. Normally, you would use this to allow the user to browse help information associated with your application.

For compiled HTML files whose names end with the ".chm" extension, $WINHELP invokes the Microsoft HTML Help Viewer application "hh.exe."

Note: $WINHELP is not supported in Microsoft Windows Vista. Vista does not support WinHelp files (.hlp) meaning any programs that use the $WINHELP routine will no longer function. Microsoft is encouraging developers to transition their help file formats to CHM, HTML, or XML.

Usage

```
CALL "$WINHELP"
    USING HELP-FILE, OP-CODE, PARAM-VAL
```

Parameters

HELP-FILE PIC X(n)

This is the file name of the help file.

OP-CODE Numeric parameter

This value indicates the desired operation.

PARAM-VAL Type depends on op-code

This is an operation-dependent parameter. It may be omitted for several operations.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

HLP Files

Microsoft Help allows the user to search through ".HLP" files in a variety of ways. In order to use Microsoft Help in your application, you must create one or more ".HLP" files. To do this, you need a program that converts text files into *help* files. This is called a *help compiler.* You can find a help compiler (called "hc") and instructions on its use in the Microsoft Windows SDK. You will also find information on how to construct the text files that get compiled into help files.

Note: Microsoft is encouraging developers to move away from .HLP files, as this file type is not supported in Windows Vista.

WinHelp

After you've created help files, use the WinHelp library routine to interact with the Microsoft Help application that is bundled with Windows. This allows you to perform a variety of functions such as opening one of your help files and directing Help to a particular topic. WinHelp is fully explained in the SDK documents.

$WINHELP provides a direct interface to WinHelp. The advantages to using $WINHELP are:

- you don't have to relink the runtime system
- you use a simplified calling sequence

WinHelp takes four parameters: the handle of the main application window, the name of the help file, an operation code and an operation-dependent parameter. $WINHELP takes similar parameters, but with the following differences:

- You do not pass the window-handle parameter. $WINHELP always supplies this parameter using the main ACUCOBOL-GT window. Thus, you pass (at most) three parameters to $WINHELP instead of the four that are passed to WinHelp.
- If the operation-dependent parameter is not needed, you may omit it. In this case, $WINHELP will pass a value of zero to WinHelp.
- All parameters are passed BY REFERENCE (the default for COBOL).

In summary, you pass either two or three parameters to $WINHELP. These are (in order):

1. the filename of the help file
2. the desired operation code
3. an operation-dependent parameter (which can be omitted)

As a convenience, the COBOL declarations for the operation codes can be found in the file "winhelp.def". The operation codes found there exactly correspond to the operation codes documented for WinHelp in the SDK.

Commonly Used Operations

Most of the operation codes are used to manage context-sensitive help. For more information about ACUCOBOL-GT support for context-sensitive help, see **Chapter 10: Help Automation** in Book 2, *User Interface Programming*. For a simple, context-*independent* help, you will typically use just the following operations:

HELP-CONTENTS --displays the topic specified by the Contents option in the [OPTIONS]section of the .HPJ file. Microsoft indicates that this command is for backward compatibility. New applications should provide a .CNT file and use the HELP-FINDER command. You do not use the third parameter for this call.

HELP-FINDER -- displays the Help Topics dialog box with the last tab used. You do not use the third parameter for this call.

HELP-PARTIALKEY -- Use this operation to bring up the keyword-search window. The third parameter should be the entire or partial keyword you want to find. Typically, this will be set to LOW-VALUES to allow key searches throughout the keyword list. One easy way to do this is to use "x'00'" as the third parameter.

HELP-HELPONHELP -- Calls up Help's own help file. The third parameter is not used.

HELP-QUIT -- Use this to exit the Help application. Microsoft requires that each application using Help logs out when it is done (if it doesn't, then Help keeps running after the application has quit). Note that $WINHELP tracks each help file used and automatically logs out each one when the runtime shuts down. Because of this, you need to use HELP-QUIT only if you want to shut down Help prior to exiting your application.

The help system provided with the ACUCOBOL-GT debugger uses these calls. The debugger presents three options to the user: "Contents", "Search", and "Help on Help". The COBOL code that corresponds to these cases is:

```
     evaluate menu-selection
        when menu-contents
           call "$winhelp" using
           "acudebug.hlp",
           help-contents
*Note newer applications should consider using help-finder
instead of help-contents.

        when menu-search
           call "$winhelp" using
           "acudebug.hlp",
           help-partialkey, x"00"

        when menu-help-on-help
```

```
            call "$winhelp" using
            "acudebug.hlp",
            help-helponhelp
        end-evaluate.
```

This is all that's required to provide a simple help system. For information on creating a context-sensitive help system, see the SDK documentation.

CHM Files

Microsoft makes an online help system called HTML Help. HTML Help uses the underlying components of Microsoft Internet Explorer to display help content. HTML Help supports HTML, ActiveX, Java, scripting (JavaScript and Microsoft Visual Basic Script), and HTML image formats (.jpeg, .gif, .png).

To create online help for your application, use the authoring tool in HTML Help called HTML Help Workshop (HHW). HHW will help you create a compiled HTML file that ends with the ".chm" extension. The "Official Microsoft Help Authoring Kit" from Microsoft Press has more information about help authoring. Contact Microsoft for more information about compiled HTML files.

$WINHELP will automatically detect the ".chm" file extension and invoke the Microsoft HTML Help Viewer application (hh.exe) to access the file. Only two operation codes are supported, HELP-CONTENTS and HELP-CONTEXT. When the operation code is HELP-CONTENTS, $WINHELP simply passes the specified compiled HTML help file name as a command-line argument to hh.exe. When the second parameter to $WINHELP, the operation code, is HELP-CONTEXT, $WINHELP constructs a command line using the following template:

```
hh.exe -mapid context_id help_file
```

where "help_file" is the first parameter to $WINHELP and "context_id" is the third parameter to $WINHELP. For example, the COBOL call:

```
CALL "$WINHELP" USING "c:\mydir\myhelp.chm",
HELP-CONTENTS
```

translates to the system command line:

```
hh c:\mydir\myhelp.chm
```

and

```
CALL "$WINHELP" USING "c:\mydir\myhelp.chm",
    HELP-CONTEXT, 72
```

translates to:

```
hh -mapid 72 c:\mydir\myhelp.chm
```

W$KEYBUF

This routine lets you add characters to the runtime's keyboard input buffer. This allows a program to simulate user input.

Usage

```
CALL "W$KEYBUF"
    USING OP-CODE, parameters
```

Parameters

OP-CODE PIC 9(2)

parameters Vary depending on the op-code chosen

Description

The first parameter to W$KEYBUF is a number that determines the action of the routine. The following options are supported:

1. Use op-code "1" to add keystrokes to the keyboard input buffer. In this case, the second parameter to W$KEYBUF specifies the key or keys you want to add. Optionally, you may also specify a third parameter that contains the number of characters you want to add. If you omit the third parameter, then all of the second parameter is used (including any trailing spaces). When keystrokes are added to the input buffer, they are placed after any previously added keystrokes, but before any keys typed by the user. This ensures that the user does not interfere with your programmatic control.
2. This is the same as "1" except that the characters are added at the beginning of the input buffer instead of at the end.

3. Use op-code “3” to empty the input buffer. This can be useful in some error handling routines. Note that this routine does *not* empty the operating system’s input buffer--any keys queued up by the user are still available. Only keys added with W$KEYBUF are removed.

4. Op-code “4” turns on the keystroke recording mechanism. You must pass as a second parameter the name of a buffer in which you want to place the recorded keystrokes. The buffer is a field you have defined in the Data Division. The size of this buffer limits the number of keystrokes recorded. You may optionally pass a third parameter that is the size of the buffer. If you omit the third parameter, then the entire buffer is used. The keystroke recorder does not initialize the buffer, and it does not modify any part of the buffer that follows the last recorded keystroke.

5. Op-code “5” turns off the keystroke recorder. This action sets RETURN-CODE to the number of characters used in the buffer to hold the recorded keystrokes.

6. Op-code “6” inquires whether or not the keystroke recorder is active. This will set RETURN-CODE to “1” if keystroke recording is currently turned on. If it is turned off, then RETURN-CODE will be set to “0”.

7. Op-code “7” causes keys typed by the user to be recorded in a file. You pass the name of the file as the first parameter after the op-code. If that file exists, it is deleted first. If the file is successfully created, then RETURN-CODE is set to zero. If the file cannot be created, then RETURN-CODE is set to “1”. You use op-code “5” to turn off the recording. Only one recording mode can be active at once, so this op-code will cancel any other active keystroke recording. In thin client environments, keys typed by the user are recorded in a temporary local file. That file is automatically uploaded to the server when recording is stopped with op-code “5”.

8. Op-code “8” is identical to op-code “7”, except that the file is appended to if it already exists. In thin client environments, the file is downloaded from the server to the client, appended to, and then uploaded to the server with op-code “5”.

9. Op-code “9” causes a previously recorded file to be “played back.” The keystrokes recorded in that file are treated as input from the user. The file name is passed as the first parameter after the op-code.

RETURN-CODE is set to zero if the file is opened successfully, otherwise RETURN-CODE is set to "1". Keystrokes inserted into the keyboard buffer using op-codes "1" or "2" of W$KEYBUF are processed before the keystrokes recorded in the file. Once the keystrokes in the file have all been used, control passes back to the user's keyboard.

In thin client environments, this op-code first tries to open the file on the client machine using the path specified in the CALL statement. If the file is not found, the thin client requests that the file be downloaded from the server. In this case the path specified in the CALL statement refers to the server machine's file system. The specified path can be an absolute path or a path relative to the current working directory, as specified in the alias file. If the file is found, it is downloaded to the client machine and stored in the thin client's temporary cache directory (the directory specified by the TEMP environment variable; often "C:\WINDOWS\TEMP". If the TEMP environment variable is not set, the files are stored in the client's current working directory. The file is not downloaded again unless it changes or is deleted from the client.

There is a runtime command-line option that causes the immediate playback of a keystroke file. This option, "-k", causes the next command-line argument to be treated as the name of a file that contains recorded keystrokes. The runtime internally calls W$KEYBUF using op-code "9" and this file name prior to executing the first COBOL program. The effect is that the keystrokes recorded in the file are treated as the runtime's first user input.

10. Op-code "10" allows you to specify how long the system waits after each simulated keystroke is processed. The delay, expressed in hundredths of a second, is passed as the second parameter. For example, to add a quarter-second delay to each keystroke, use the following:

    ```
    CALL "W$KEYBUF" USING 10, 25
    ```

    You can terminate a pause early by pressing any key.

11. Op-code "11" defines a key that causes the playback to pause for an indefinite time period, allowing you to explain a special feature. The playback resumes when another key is pressed. To designate the pause

key, use op-code “11” with the ASCII value of the key. Note that you can only define a pause key that is a simple ASCII key, not a complex key like a function key.

12. Op-code “12” defines a key that stops the playback and returns the program to interactive mode. Use op-code “12” with the ASCII value of the cancel key.

Special keystrokes

You may specify special keystrokes by placing code names in curly braces. Within curly braces, you may use the caret (^) to indicate Control characters or the “at” symbol (@) to indicate ALT characters. For example, “{^M}” indicates Control+M and “{@L}” indicates ALT+L.

You may also use a special key’s two-character name as found in the Table of Keys in the *ACUCOBOL-GT User’s Guide*, section 4.3.2.3. For example, you may refer to function key 2 with “{k2}”.

Menu selections are encoded as {m#} where “#” is the numeric ID of the menu item.

You may insert specific pauses in a simulated input stream using the following character sequences:

{p#} where # is in hundredths of a second

{P#} where # is in seconds

{P} 1-second pause

A programmed pause also includes any pause introduced by op-code “10”, described above. You can terminate a pause early by pressing any key.

Finally, if you require an opening curly brace on its own, specify it twice. For example: “{{“.

The following line specifies that the next characters processed should be “abc”, Tab, “def”, and function key 1:

```
CALL "W$KEYBUF" USING 1, "abc{^I}def{k1}".
```

The keystroke recording mechanism records normal keys as native characters. The keystroke recorder will not record a “time out” event.

In order to operate correctly with graphical controls on Windows systems, the W$KEYBUF routine converts characters placed into the keyboard buffer into keyboard scan codes. Thus, you can use only those characters that have a corresponding keyboard scan code. As a result, you can “play back” non-English characters only when you have installed a keyboard that contains those characters.

The behavior of this routine is affected by the FILENAME_SPACES configuration variable. The value of FILENAME_SPACES determines whether spaces are allowed in a file name. See the entry for **FILENAME_SPACES** in Appendix H for more information.

W$MENU

You construct and control menus with the W$MENU library routine. You use the CALL statement to access this routine.

Usage

```
CALL "W$MENU"
    USING OP-CODE, parameters,
    GIVING WMENU-RESULT
```

Parameters

OP-CODE Numeric parameter

This indicates the desired operation. Level 78 symbolic names for these operations can be found in “acugui.def”.

parameters Vary depending on the op-code chosen

WMENU-RESULT PIC S9(9)

Holds the return value from W$MENU. A value of zero indicates failure for all operations.

Description

The routine takes one or more parameters, which are always passed BY REFERENCE (the default in COBOL). The first parameter is always an operation code. This code defines what the routine will do. The remaining parameters depend on the operation selected. The operation codes are defined in the COPY file “acugui.def”. The available codes are described here:

WMENU-NEW — This operation constructs a new empty menu bar. W$MENU returns a *handle* for this menu bar in the special register WMENU-RESULT. This handle should be stored in a variable declared as PIC S9(9) COMP-4. All future references to this menu bar are made with this handle. This operation can fail if the system runs out of memory. In this case, WMENU-RESULT will be set to zero.

WMENU-NEW-POPUP — This operation constructs a new empty pop-up menu. W$MENU returns a *handle* for this menu in the special register WMENU-RESULT. This handle should be stored in a variable declared as PIC S9(9) COMP-4. All future references to this pop-up menu are made with this handle. This operation can fail if the system runs out of memory. In this case, WMENU-RESULT will be set to zero. This option is available for Windows systems only.

WMENU-POPUP — This operation displays a pop-up menu and gets the user’s response. After the user makes a selection, the pop-up menu is removed. (This is not the typical way to display a pop-up window. It is easier to assign the pop-up menu to a window or control and let the runtime handle activation.) The second parameter passed in conjunction with this operation code is the handle of the pop-up menu to show. The next two parameters are optional. If they are omitted, the menu appears where the mouse is currently located. If specified, they are the row and column (respectively) of the location where the menu should appear. This is expressed in screen base units (pixels under Windows), with “1,1” being the upper left corner of the physical screen. The user’s response is sent to the current window and is treated as if it were a regular entry from the widow’s menu bar. This option is available for Windows systems only.

WMENU-DESTROY — You must pass the handle of a valid menu as the second parameter. That menu is destroyed and all memory it occupies is released. Any submenus that it contains are also destroyed. If the menu is currently being displayed, it is removed from the screen first. After destroying a menu, you must not use it any more. You should not destroy a submenu directly—use the WMENU-DELETE operation instead.

WMENU-DESTROY-DELAYED — You must pass the handle of a valid menu as the second parameter. If that menu is not being displayed, the menu is destroyed and all memory it occupies is released. Any submenus that it contains are also destroyed. If the menu is currently being displayed, it is not destroyed immediately. Instead, it is marked for destruction later. It is actually destroyed immediately after the next WMENU-SHOW operation. After destroying a menu, you must not use it any more. You should not destroy a submenu directly—use the WMENU-DELETE operation instead.

WMENU-ADD — Adds a menu item to a menu. This operation takes several parameters. Note that all of the parameters are passed as integers (either USAGE DISPLAY or COMP-4) except for the "text" parameter, which is alphanumeric. You must use WMENU-SHOW to display an altered menu at the top level.

The parameters are, in order:

Handle This is the handle of the menu to which you are adding the item.

Position This is the location at which you want to place the new item. If this value is zero, then the item is appended to the menu. Otherwise, this value is the ID of the item in front of which you want to insert the new item.

Flags This is a combination of one or more of the following:

W-CHECKED indicates that a check mark should be placed beside the item. Check marks appear only on submenus.

W-DISABLED indicates that the item may not be selected. It will appear gray on the menu.

W-SEPARATOR indicates that this item consists of a bar that separates items on the menu. This item may not be selected by the user. The "text" field is ignored for this item. Separator bars may appear only on submenus.

You may combine flags by adding their values together. Values are defined in the file "acugui.def". The default value is "0", which indicates an enabled, unchecked menu item.

Text

This is the text of the menu item that will appear on the menu bar. Text is limited to 50 characters. The text will appear exactly as you specify it, so be sure to use the desired case. (Graphical environments typically have conventions that you may want to follow, for consistency with other applications. For example, under Windows it is conventional to capitalize the first letter of each major word in a menu item.)

You may place an "&" symbol in front of the letter you want to use as the *key letter* for this menu item. The user can select this menu item by pressing <Alt> and typing its key letter. The key letter typically appears underlined on the menu (you can change this appearance; see **Section 8.9.4, "Menu Configuration With the Generic Menu Handler"** in Book 2, *ACUCOBOL-GT User Interface Programming*).

If you omit the "&" symbol, no underline will appear, and the first letter of the item will be used as the item's key letter. If the same key letter represents more than one item, and a user types that letter, the system will highlight the next menu item with that key letter and will wait for the user to confirm the selection by pressing <return>. If "flags" contains the W-SEPARATOR option, you should either omit this parameter or pass NULL.

Id This is an unsigned integer less than or equal to 4095 that you will use to identify this menu item. Use this ID anytime you need to refer to this menu item. This is the ID that's returned to your program when the item is selected by the user. Assign a unique ID to each item in a particular menu hierarchy. Normal (text) menu items must be given an ID. Submenu names and separator bars may optionally be given an ID of zero. These become anonymous menu items that you may never refer to again. If this parameter is omitted, an ID of zero is used.

Submenu This parameter is omitted or set to zero for normal menu items and separator bars. You can create a submenu by setting this value to the handle of another menu. When the user selects this item, that other menu will pop up. Pop-up menus are placed down and to the right, unless they fit better in another position.

WMENU-CHANGE — This operation takes the same parameters as WMENU-ADD (handle, position, flags, text, id, and submenu). The "position" parameter must not be zero. It indicates the ID of the menu entry you want to change. That entry is deleted and the entry described by the current parameters is inserted in the same location. If the item that you are replacing is a submenu, that submenu's entries are also destroyed. If you change the main menu bar, you must use WMENU-SHOW to display the changed menu.

WMENU-DELETE — Pass the handle of a menu as the second parameter, and the ID of an entry in that menu as the third parameter. The indicated entry is removed by this operation. If the deleted item is a submenu name, the submenu is also destroyed. If you change the main menu bar, you must use WMENU-SHOW to display the changed menu.

WMENU-CHECK — This operation places a check mark in front of a menu item. You pass two additional parameters: the handle of the menu and the ID of the item you want to change. Only items that appear in submenus can have check marks.

WMENU-UNCHECK — Use this operation to remove a check mark from a menu item. The second and third parameters are the handle of the menu and the ID of the item you want to change.

WMENU-DISABLE — Use this operation to disable individual menu items. Disabled menu items are displayed with gray text. The second and third parameters are the menu's handle and the ID of the item you want to disable. You may disable items that appear on the top level of the menu. If you disable a submenu, then access to the submenu's items is denied. If you change the main menu bar, you must use WMENU-SHOW to display the changed menu. If you call W$MENU using this option, and pass the menu's main handle, without specifying an item on the menubar, the entire menu bar is disabled.

WMENU-ENABLE — Use this operation to enable a menu item. The second and third parameters are the menu's handle and the menu item's ID. If you change the main menu bar, you must use WMENU-SHOW to display the changed menu.

WMENU-SHOW — Use this operation to display a menu bar on the screen. The first parameter after the op-code is the handle of the menu bar to show. The second parameter (optional) is the handle of the window on which to display the menu. If the second parameter is omitted, the menu is displayed on the current window. Use this operation both to display the menu bar initially and to display any changes made to the top level of the menu.

For example, if you disable one of the top-level menu items, then you must use WMENU-SHOW to make that change visible. If the application already has another menu bar showing when you call WMENU-SHOW, then that menu bar is removed. It is not destroyed--you may re-use that menu bar again later. If you pass a menu handle of zero, then any existing menu bar is removed and no new menu bar is shown, so passing a menu handle of zero is a way to clear all menu bars off the screen.

WMENU-REFRESH — Use this operation to redraw an existing menu. This is typically used to restore a menu that has been overwritten by an external piece of software. For example, if you called the SYSTEM library routine to display the current directory, the directory listing might overwrite the menu. Use WMENU-REFRESH to redraw the menu when you are ready to see it again.

WMENU-GET-MENU — Sets WMENU-RESULT to the handle of the currently displayed menu. This returns zero if no menu is currently displayed. Use this in routines that need to save the current menu bar before replacing it with their own menu.

WMENU-RELEASE — This operation logically removes the menu from the screen, but doesn't update the screen. The menu is still visible, but not operational. This makes the entire screen available to your program, but doesn't scroll the current screen contents. This is occasionally useful when you need to remove a menu bar and clear the entire screen. If you instead use WMENU-SHOW to remove the menu bar, you'll have additional screen activity as it removes the menu bar and (if that bar was static) scrolls the screen. The screen update isn't needed if you are going to clear the whole screen anyway.

Under Windows, this function blocks the menu bar, but leaves it in place. The runtime uses this function just prior to shutting down.

This operation takes no parameters.

WMENU-GET-CONFIGURATION — Returns the generic menu handler's current configuration. You must pass it one parameter that has the layout described in the next section. This gets filled in with the current configuration. This operation sets WMENU-RESULT to "1" if the configuration was retrieved. It sets WMENU-RESULT to "0" if the host machine does not use the generic menu handler. In this case, the configuration information is not used and is meaningless.

WMENU-SET-CONFIGURATION — You must pass one parameter to this operation that has the layout described in the next section. It causes this parameter to become the current configuration for the generic menu handler. It sets WMENU-RESULT to "1" if the generic menu handler is being used, or "0" if it is not. In the latter case, the configuration will not be used.

If you have a menu displayed when you change the configuration, you should immediately use WMENU-SHOW to update that menu. Alternately, you may use WMENU-SHOW to remove that menu and display a new menu with the new configuration. If you have a menu displayed and change the configuration, you can get undefined results if you fail to use WMENU-SHOW.

WMENU-BLOCK — This inhibits the menu. It works by maintaining a *blocking-count*. Initially, the blocking-count is set to zero. This call adds one to the blocking-count. Anytime the blocking-count is greater than zero, the user will not be allowed to use the menu. This is typically used by "modal" routines that need to ensure that the user completes some action

before continuing. For example, if you are prompting for a file name, you might want to disable the menu until the user has entered a name. See also WMENU-UNBLOCK, WMENU-GET-BLOCK, and WMENU-SET-BLOCK.

WMENU-UNBLOCK — If the blocking-count is currently greater than zero, this subtracts one from the blocking-count. If this results in the blocking-count reaching zero, then the user will once again be able to use the menu. See also WMENU-BLOCK, WMENU-GET-BLOCK, and WMENU-SET-BLOCK.

WMENU-GET-BLOCK — Sets WMENU-RESULT to the current blocking-count. This is typically used by routines that need to save the current menu state before replacing the menu with their own menu. They can save the blocking-count, set it to zero (with WMENU-SET-BLOCK), and then reset to the saved state when they exit. For example, the ACUCOBOL-GT debugger uses this call when replacing the application menu with its own menu. See also WMENU-BLOCK, WMENU-UNBLOCK, and WMENU-SET-BLOCK.

WMENU-SET-BLOCK — Sets the blocking-count to the value of the second parameter. Use this in conjunction with WMENU-GET-BLOCK to save and restore the current blocking-count. See also WMENU-BLOCK, WMENU-UNBLOCK, and WMENU-GET-BLOCK.

The W$MENU routine always sets WMENU-RESULT. Except as described above, this is either “1” to indicate success or “0” to indicate that the operation failed.

The ACUCOBOL-GT generic menu handler allows you to configure several aspects of its look and feel. This is done with get/set configuration operations described in the previous section. For both of these operations, you must pass a parameter that has the following layout:

```
01 MENU-CONFIGURATION.
   03 MENU-STYLE                          PIC 9 COMP-X.
      88 MENU-IS-STATIC                   VALUE 0.
      88 MENU-IS-POPUP                    VALUE 1.
   03 MENU-CHECK-MARK                     PIC X.
   03 MENU-SUBMENU-MARK                   PIC X.
   03 MENU-COLOR-ATTRIBUTES.
      05 MENU-NORMAL-COLOR-ATTRIBUTES.
```

```
          07 MENU-NORMAL-COLOR            PIC 9(4) COMP-X.
          07 MENU-NORMAL-COLOR-KEY-1      PIC 9(4) COMP-X.
          07 MENU-NORMAL-COLOR-KEY-2      PIC 9(4) COMP-X.
       05 MENU-SELECTED-COLOR-ATTRIBUTES.
          07 MENU-SELECTED-COLOR          PIC 9(4) COMP-X.
          07 MENU-SELECTED-COLOR-KEY-1    PIC 9(4) COMP-X.
          07 MENU-SELECTED-COLOR-KEY-2    PIC 9(4) COMP-X.
       05 MENU-DISABLED-COLOR-ATTRIBUTES.
          07 MENU-DISABLED-COLOR          PIC 9(4) COMP-X.
          07 MENU-DISABLED-COLOR-KEY-1    PIC 9(4) COMP-X.
          07 MENU-DISABLED-COLOR-KEY-2    PIC 9(4) COMP-X.
    03 MENU-MONO-ATTRIBUTES.
       05 MENU-NORMAL-MONO-ATTRIBUTES.
          07 MENU-NORMAL-MONO             PIC 9(4) COMP-X.
          07 MENU-NORMAL-MONO-KEY-1       PIC 9(4) COMP-X.
          07 MENU-NORMAL-MONO-KEY-2       PIC 9(4) COMP-X.
       05 MENU-SELECTED-MONO-ATTRIBUTES.
          07 MENU-SELECTED-MONO           PIC 9(4) COMP-X.
          07 MENU-SELECTED-MONO-KEY-1     PIC 9(4) COMP-X.
          07 MENU-SELECTED-MONO-KEY-2     PIC 9(4) COMP-X.
       05 MENU-DISABLED-MONO-ATTRIBUTES.
          07 MENU-DISABLED-MONO           PIC 9(4) COMP-X.
          07 MENU-DISABLED-MONO-KEY-1     PIC 9(4) COMP-X.
          07 MENU-DISABLED-MONO-KEY-2     PIC 9(4) COMP-X.
```

A copy of this data item can be found in “acugui.def”.

When a menu is shown, the current configuration defines how it is presented. The fields have the following meaning:

MENU-STYLE — Determines whether the menu is static (value “0”) or pop-up (value “1”). The default is static.

MENU-CHECK-MARK — Sets the character used to display check marks. The default character is an asterisk.

MENU-SUBMENU-MARK — Sets the character used to indicate that the item is a submenu. This is not used on the main menu bar. The default character is a greater-than sign.

MENU-COLOR-ATTRIBUTES — Defines the display attributes used if the station supports color. This is described in detail below.

MENU-MONO-ATTRIBUTES — Defines the display attributes used if the station is monochrome. See below for details.

Notice that there are two sets of attributes, one for color stations and one for monochrome stations. Within each set, there are three group items that determine the attributes used for a particular case. Using color stations as an example, we have:

MENU-NORMAL-COLOR-ATTRIBUTES — Determines the attributes used for the normal menu items. This is used for the menu's background color along with all items that are not highlighted or disabled.

MENU-SELECTED-COLOR-ATTRIBUTES — Determines the attributes used for the currently selected (highlighted) menu items.

MENU-DISABLED-COLOR-ATTRIBUTES — Determines the attributes used by disabled menu items.

Finally, within each group, there are three fields that determine the attributes used for the appropriate menu items. For example, with normal menu items on a color station, we have:

MENU-NORMAL-COLOR — This is the attribute used to draw the menu item and its background.

MENU-NORMAL-COLOR-KEY-1 — This is the attribute used to draw the key letter in the menu item (if any).

MENU-NORMAL-COLOR-KEY-2 — This is used only if MENU-NORMAL-COLOR-KEY-1 contains the underline attribute and the station does not support underlining. In this case, the actual attribute used is determined by this field.

This scheme of having two possible attributes available for the key letter makes it easier to implement a portable set of attributes. Traditionally, key letters are shown underlined, and so underlining is typically used in the "KEY-1" attribute. Since many stations do not support underlining, having a second attribute provides a backup system.

Attributes are set using the same scheme as that used for the COLOR phrase in an ACCEPT or DISPLAY statement. The only exception is that the "Protected" attribute (32768) has a special meaning when used in conjunction with disabled menu items. If you specify "protected" as the disabled item's color, then disabled items will appear in parentheses on the menu. This helps distinguish them from normal items.

Attribute values are numeric. They represent combinations of colors and other video features such as intensity. You make combinations by adding appropriate values together from these tables:

Color	**Foreground**	**Background**
Black	1	32
Blue	2	64
Green	3	96
Cyan	4	128
Red	5	160
Magenta	6	192
Brown	7	224
White	8	256

Attribute	**Value**
Reverse video	1024
Low intensity	2048
High intensity	4096
Underline	8192
Blink	16384
Protected	32768

The default settings for attributes are as follows:

Color Defaults

Type	Main Attribute	Key-1 Attribute	Key-2 Attribute
Normal	Black on Cyan (129)	Black on Cyan, Underlined (8321)	High-intensity White on Cyan (4232)
Selected	White on Blue (72)	White on Blue, Underlined (8264)	High-intensity White on Blue (4168)
Disabled	White on Cyan (136)	White on Cyan (136)	White on Cyan (136)

Monochrome Defaults

Type	Main Attribute	Key-1 Attribute	Key-2 Attribute
Normal	Black on White (257)	Black on White, Underlined (8449)	High-intensity Black on White (4353)
Selected	White on Black (40)	White on Black, Underlined (8232)	High-intensity White on Black (4136)
Disabled	Black on White, Parenthesized (32935)	Black on White (257)	Black on White (257)

The ACUCOBOL-GT debugger always uses the default configuration for its own menu bar.

W$MOUSE

This routine allows you to control the behavior of the mouse. Before using W$MOUSE to respond to mouse actions, be sure that you have unmasked the actions you want to notice (see Book 2, **Section 7.3**).

Usage

```
CALL "W$MOUSE"
    USING OP-CODE, parameters,
    GIVING MOUSE-STATUS
```

Parameters

OP-CODE Numeric parameter

This indicates the desired operation. Level 78 symbolic names for these operations can be found in the file “acugui.def”.

parameters Vary depending on the op-code chosen.

Code	Operation
0	TEST-MOUSE-PRESENCE
1	GET-MOUSE-STATUS
2	GET-MOUSE-SCREEN-STATUS
3	SET-MOUSE-POSITION
4	SET-MOUSE-SCREEN-POSITION
5	SET-MOUSE-SHAPE
6	SET-DELAYED-MOUSE-SHAPE
7	GET-MOUSE-SHAPE
8	CAPTURE-MOUSE
9	RELEASE-MOUSE
10	ENABLE-MOUSE
19	SET-MOUSE-HELP
23	SET-MOUSE-POSITION-EX
24	SET-MOUSE-SCREEN-POSITION-EX
25	SET-MOUSE-POSITION-PIXEL
26	SET-MOUSE-SCREEN-POSITION-PIXEL

MOUSE-STATUS A numeric data item

Holds the return value from W$MOUSE. This is used only with the ENABLE-MOUSE, TEST-MOUSE-PRESENCE, and GET-MOUSE-SHAPE operations.

Description

The first parameter you pass to W$MOUSE is an operation code. This code tells W$MOUSE what to do. The number and type of the remaining parameters depend on the operation selected. All parameters are passed BY REFERENCE (the default in COBOL). The operation code can be one of the following:

ENABLE-MOUSE (op-code 10)

Turns on the mouse on character-based systems. (By default the mouse is invisible.) The value of RETURN-CODE is set to "1" if a mouse is present, "0" if no mouse is found. (Takes no additional parameters.)

On some machines there is an appreciable delay when the mouse is enabled.

Under graphical environments such as Windows, ENABLE-MOUSE is not necessary. It sets the value of RETURN-CODE properly, but no other action occurs, because the mouse is *always* enabled in these environments.

TEST-MOUSE-PRESENCE (op-code 0)

Used to detect the presence of a mouse. The value of RETURN-CODE is set to "1" if a mouse is available on the system. It is set to "0" if no mouse is present. (Takes no additional parameters.) Note that on character-based systems, the mouse must be enabled before its presence can be detected. So if you haven't enabled the mouse (see ENABLE-MOUSE above), RETURN-CODE will have a value of "0".

GET-MOUSE-STATUS (op-code 1)

Returns information about the mouse's location and the state of each of its buttons. You must pass a group item with the following structure (defined in "acugui.def"):

```
01  MOUSE-INFO.
   03  MOUSE-ROW               PIC 9(4) COMP-1.
       88  MOUSE-OFF-SCREEN    VALUE ZERO.
   03  MOUSE-COL               PIC 9(4) COMP-1.
   03  LBUTTON-STATUS          PIC 9.
       88  LBUTTON-DOWN        VALUE 1.
   03  MBUTTON-STATUS          PIC 9.
       88  MBUTTON-DOWN        VALUE 1.
   03  RBUTTON-STATUS          PIC 9.
       88  RBUTTON-DOWN        VALUE 1.
   03  MOUSE-ROW-EX            PIC 9(6)V99 COMP-4, SYNC.
   03  MOUSE-COL-EX            PIC 9(6)V99 COMP-4.
   03  MOUSE-ROW-PIXEL         PIC 9(8) COMP-4.
   03  MOUSE-COL-PIXEL         PIC 9(8) COMP-4.
```

The routine fills in this structure with data about the mouse. Each of the three "status" fields is set to "1" if the corresponding button is depressed. Otherwise, they are set to zero. The various row and column fields are set to the location of the mouse within the current ACUCOBOL-GT window. If the mouse is outside of the current window, then these values are set to zero.

Here's an example of a call to W$MOUSE that returns the menu item the mouse is pointing to:

```
* find-mouse-menu-row - returns (in mouse-row)
* the menu item that the mouse pointer is
* currently on. If the mouse is not on a menu
* item, it returns zero.

find-mouse-menu-row.
   call "w$mouse" using get-mouse-status, mouse-info
   if mouse-row >= menu-row and
         mouse-row < menu-row + num-menu-items * 2
      compute mouse-row = mouse-row - menu-row + 1
   else
      move zero to mouse-row.
```

After an ACCEPT statement is executed, all CALLs to W$MOUSE pertain to that ACCEPT statement, until another ACCEPT is executed. So, you always get the right mouse status. By synchronizing the mouse actions with the appropriate exception values, the runtime ensures that you process the mouse correctly.

GET-MOUSE-SCREEN-STATUS (op-code 2)

This function is the same as the GET-MOUSE-STATUS function, except that the row and column coordinates are relative to the application's virtual screen instead of the current window. If the mouse is not located anywhere in the application's window, then the coordinates are set to zero.

SET-MOUSE-POSITION (op-code 3)

You must pass a group item with the same structure as described under GET-MOUSE-STATUS. The mouse is placed at the coordinates named by MOUSE-ROW and MOUSE-COL, relative to the current ACUCOBOL-GT window. The button-status fields are not used.

SET-MOUSE-POSITION-EX (op-code 23)

You must pass a group item with the same structure as described under GET-MOUSE-STATUS. The mouse is placed at the coordinates named by MOUSE-ROW-EX and MOUSE-COL-EX, relative to the current ACUCOBOL-GT window. These coordinates are similar to MOUSE-ROW and MOUSE-COL, except that the positioning of the mouse is performed using a precision of hundredths of cells. The button-status fields are not used.

SET-MOUSE-POSITION-PIXEL (op-code 25)

You must pass a group item with the same structure as described under GET-MOUSE-STATUS. The mouse is placed at the coordinates named by MOUSE-ROW-PIXEL and MOUSE-COL-PIXEL, relative to the current ACUCOBOL-GT window. These coordinates describe the mouse position in terms of the display's base units (which are pixels for graphical systems). Unlike MOUSE-ROW and MOUSE-COL, or MOUSE-ROW-EX and MOUSE-COL-EX, the uppermost pixel is row "0"/column "0" instead of row "1"/column "1". The button-status fields are not used.

SET-MOUSE-SCREEN-POSITION (op-code 4)

This is the same as SET-MOUSE-POSITION, except that the position is relative to the application's virtual screen.

SET-MOUSE-SCREEN-POSITION-EX (op-code 24)

The same as SET-MOUSE-POSITION-EX, except that the position is relative to the application's virtual screen.

SET-MOUSE-SCREEN-POSITION-PIXEL (op-code 26)

The same as SET-MOUSE-POSITION-PIXEL, except that the position is relative to the application's virtual screen.

SET-MOUSE-SHAPE (op-code 5)

The second parameter defines the desired shape of the mouse pointer. The shape is changed immediately, even if the mouse has not been moved. The possible values are:

ARROW-POINTER	The default arrow shape
BAR-POINTER	A vertical bar
CROSS-POINTER	Cross hairs
HELP-POINTER	A question mark
WAIT-POINTER	An hourglass

These are defined in "acugui.def". Note that the runtime system changes the shape of the mouse when you are using automatic mouse handling. When an ACCEPT statement finishes execution, the last shape defined by SET-MOUSE-SHAPE is restored.

SET-DELAYED-MOUSE-SHAPE (op-code 6)

This is identical to SET-MOUSE-SHAPE, except that the mouse pointer is not changed immediately. Instead, it is changed as soon as the user moves it. The automatic mouse handler uses this function to provide a smoother mouse appearance.

GET-MOUSE-SHAPE (op-code 7)

The value of RETURN-CODE is set to the current mouse shape. If SET-DELAYED-MOUSE-SHAPE is in effect, waiting for the mouse to move, then the delayed shape is returned. (Takes no additional parameters.)

CAPTURE-MOUSE (op-code 8)

Normally, only mouse actions that occur within the application's *virtual screen* are processed by the runtime system. (The virtual screen is the drawing space allocated to the application.) Actions that occur outside of this space are handled by other applications or by the environment. The CAPTURE-MOUSE function causes the runtime to process all mouse messages, regardless of where they occur. This should be done only in special cases, because it prevents the user from using the mouse in any other application. Normally, you capture the mouse only when the user is marking or dragging some object on the screen. By capturing the mouse, you can control the action even if the user accidentally moves the mouse outside of the application. (Takes no additional parameters.)

When the mouse is captured, if the mouse is outside of the legal boundaries, the GET-MOUSE-STATUS and GET-MOUSE-SCREEN-STATUS functions no longer return zero in the row and column fields. Instead, the row and column fields contain the nearest legal coordinate to the mouse's actual position.

RELEASE-MOUSE (op-code 9)

This reverses the effects of the CAPTURE-MOUSE function. You must call this sometime after calling CAPTURE-MOUSE, or the environment will not be able to use the mouse again. (Takes no additional parameters.)

SET-MOUSE-HELP (op-code 19)

When the second parameter is "1", this op-code turns the mouse pointer into a *help-mode* pointer--the mouse is captured and the pointer takes the shape of a question mark. When the second parameter is "0", the help-mode state is turned off--the mouse is released and the pointer returns to its default shape. This option is only supported under Microsoft Windows.

Note: If you use *help automation* (see **Chapter 10: Help Automation** in Book 2, *User Interface Programming*), you don't need to use this option. Help automation automatically manages the help-mode mouse states for you.

Mouse Handling: Sample Code

```
identification division.
program-id.  mouse-sample.
remarks.
   This program provides an example of programmed
   mouse handling.
data division.
working-storage section.

77  key-entered    pic 9(3).
    88  mouse-button-clicked  value 81, 87.
    88  left-button-clicked   value 81.
    88  right-button-clicked  value 87.

77  menu-selection  pic 9 value 1.
    88  exit-selected  value 9.

77  MOUSE-FLAGS    pic 9(5).
77  pointer-idx    pic 9 value zero.

01  name-array.
  03 name-data.
    05  filler    pic x(9) value "arrow".
    05  filler    pic x(9) value "bar".
    05  filler    pic x(9) value "cross".
    05  filler    pic x(9) value "hourglass".
  03  mouse-name redefines name-data occurs 4    pic x(9).

copy "acugui.def".
screen section.
01  main-screen.
  03  "The Screen", reverse high line 1 col 20.
  03  "Change Pointer", reverse high line 2 col 3.
  03  "Describe Pointer", reverse high line 4 col 3.
  03  "EXIT", reverse high line 6 col 3.
procedure division.
main-logic.
```

```
* Test for presence of mouse on system.
    call "w$mouse" using test-mouse-presence.
    if return-code = zero
      display "No mouse present"
      go to main-logic-exit.

* Enable program to recognize left and right button clicks.
    add allow-left-down, allow-right-down giving mouse-flags.
    set environment "mouse-flags" to mouse-flags.

    display main-screen.
    perform main-screen-handling
      until exit-selected.
main-logic-exit.

    stop run.

main-screen-handling.

    accept omitted, line 1, control key in key-entered.

    if mouse-button-clicked
      call "w$mouse" using get-mouse-status, mouse-info
      evaluate mouse-row
        when 2   perform change-mouse-shape
        when 4   perform display-mouse-shape
        when 6   set exit-selected to true
        when other
          display "Clicked"
        end-evaluate.

change-mouse-shape.
    if pointer-idx < 4
      add 1 to pointer-idx
    else
      move 1 to pointer-idx.
    call "w$mouse" using set-mouse-shape, pointer-idx.

display-mouse-shape.
    call "w$mouse" using get-mouse-shape.
display "Current mouse shape is" mouse-name (return-code).
```

W$PALETTE

In graphical environments, you can customize the basic set of colors that you use in your programs.

ACUCOBOL-GT allows programs to reference 16 distinct colors (8 low-intensity and 8 high-intensity). For most machines, this set of colors is fixed (black, blue, green, cyan, red, magenta, brown, and white). On Windows machines, you can select which 16 colors you will be using. You are given access to a *palette* of 16 colors that defines which color corresponds to each color number.

You control the palette through the library routine W$PALETTE.

Usage

```
CALL "W$PALETTE"
    USING OP-CODE, WPALETTE-DATA
    GIVING RESULT
```

Parameters

OP-CODE Numeric value

Selects which palette function to perform. The operations are described below.

WPALETTE-DATA Group item as follows:

```
01  WPALETTE-DATA.
    03  WPAL-COLOR-ID    PIC X COMP-X.
    03  WPAL-FLAGS REDEFINES
        WPAL-COLOR-ID    PIC X COMP-X.
    03  WPAL-RED         PIC X COMP-X.
    03  WPAL-USER-COLOR-ID REDEFINES
        WPAL-RED         PIC X COMP-X.
    03  WPAL-GREEN       PIC X COMP-X.
    03  WPAL-BLUE        PIC X COMP-X.
```

This provides information and holds results for certain operations described below. It may be omitted from those operations that do not use it.

RESULT Signed numeric data item.

Returns the status of the operation. Unless otherwise stated below, “1” indicates success, and a zero or negative result indicates failure.

The WPALETTE-DATA group item and all of the level 78 symbolic names described below can be found in the COPY library “palette.def”.

Description

W$PALETTE performs a variety of operations depending on the passed OP-CODE. These operations are as follows:

WPALETTE-SUPPORTED (op-code 1)

This determines the level of support the host machine provides for the W$PALETTE routine. Use this to determine if the host machine will allow you to perform certain operations.

WPALETTE-DATA is not used. The RESULT value will be one of the following:

WPAL-NO-SUPPORT (value “0”) -- This value indicates that the host machine does not support palettes. Only the WPALETTE-SUPPORTED and WPALETTE-NUM-COLORS operations will function. Currently, this result is returned for all platforms other than Windows and Windows NT.

WPAL-PALETTE-SUPPORTED (value “1”) -- This value indicates that all W$PALETTE functions are supported except for the WPALETTE-CHOOSE-COLOR function.

WPAL-FULL-SUPPORT (value “2”) -- This value indicates that all W$PALETTE functions are available.

WPALETTE-NUM-COLORS (op-code 2)

This operation sets RESULT to the number of distinct solid colors that the host machine can display simultaneously.

WPALETTE-DATA is not used. For monochrome machines, the RESULT value will be “2”. For color machines other than Windows, this value will be “16”.

For Windows machines, this value will depend on the host hardware and drivers installed. For a standard VGA system, this value will be “16”. For Super VGA systems with the proper driver installed, this value can be “256” or higher. If the machine supports more than 32,767 distinct colors, then “32767” will be returned.

For machines with 16 or fewer colors, the standard ACUCOBOL-GT palette represents the entire range of solid (pure) colors. In order to display any other color, that color must be simulated by dithering two or more colors together. Windows allows us to do this, but only for background colors. If you attempt to display a dithered color in the foreground, Windows will automatically substitute the nearest solid color. Dithered background colors can also make the foreground text look ragged, depending on the exact colors used. For this reason, you may want to limit yourself to solid colors. One way to do this is to allow the user to change the palette only when the machine supports 256 colors or more.

WPALETTE-GET-COLOR (op-code 3)

This operation retrieves the color that currently corresponds to a particular color number. Colors are numbered “1” through “16”. The first 8 colors correspond to the low-intensity colors.

Their initial values are as follows:

1	Black
2	Blue
3	Green
4	Cyan
5	Red
6	Magenta
7	Brown
8	White

The second set of 8 colors contains the same colors in their high-intensity forms. To determine the current definition of any color number, move the color number desired into the WPAL-COLOR-ID field of WPALETTE-DATA and call W$PALETTE.

When it returns, the WPAL-RED, WPAL-GREEN, and WPAL-BLUE fields of WPALETTE-DATA will be filled in with the current definition of that color number. Each of these fields will contain a value from "0" to "255" that indicates the intensity of the red, green, and blue components of the color. A red-green-blue (RGB) combination of "0", "0", "0" indicates black. A RGB value of "255", "255", "255" is bright white. Other values cover the entire range of colors possible under Windows.

WPALETTE-SET-COLOR (op-code 4)

This function complements WPALETTE-GET-COLOR; it lets you assign a new color to a particular color number. At entry, WPAL-COLOR-ID should contain the color number you want to change. WPAL-RED, WPAL-GREEN, and WPAL-BLUE should contain the RGB value of the new color (see WPALETTE-GET-COLOR for a description of RGB values).

For example, if you want to make color number "2" represent a dark blue-green, you could use the following values:

WPAL-COLOR-ID	2
WPAL-RED	0
WPAL-GREEN	64
WPAL-BLUE	64

Note: Changing the palette will change the colors currently shown on the screen as well as all future displays. In the previous example, if you had displayed any data (or background) using color number "2", that data would now change to dark blue-green.

The color palette is the lowest level of color handling in the ACUCOBOL-GT system. It defines the basic set of colors used. Although you are free to change the palette as you see fit, ACUCOBOL-GT makes certain assumptions that you should be aware of. First of all, ACUCOBOL-GT assumes that color "1" is always black, and color "16" is always bright white. Likewise, colors "8" and "9" are assumed to be shades of gray. These assumptions affect the rendering of window shadows and "3-D" lines. Additionally, the first 8 colors are assumed to be low-intensity, and the second 8 are high-intensity.

ACUCOBOL-GT computes the high- or low-intensity version of a color by adding or subtracting "8" from its color number. This assumption also affects the rendering of "3-D" lines if you use them on a colored background. Although you are not required to maintain any of these assumptions, be aware of them so that you can anticipate the total effects of your changes.

WPALETTE-UPDATE (op-code 5)

In order to allow you to change several colors efficiently, W$PALETTE does not immediately apply the new palette after you change it. It waits until the next screen update (usually caused by either an ACCEPT or DISPLAY statement). In most cases this is adequate. If you want to force the screen to be updated immediately with the new palette, you can use the WPALETTE-UPDATE function. WPALETTE-DATA is not used. The effect is to update the screen with the new palette and return.

WPALETTE-CHOOSE-COLOR (op-code 6)

This operation provides a simple method for getting a color selection from the user. You can use this to simplify the process of constructing your color palette according to your user's desires. When this operation executes, a standard color selection box pops up over your application. This box is similar to the "Color Palette" portion of the Windows' Control Panel application. It should be familiar to most users.

This box contains a selection of pre-determined colors (called the "Basic Colors") that the user can choose from. Windows selects these colors. Typically, there are 48 of them drawn from the entire spectrum (there can be fewer on some systems).

The user can pick one of these by clicking the mouse on it or by using the arrow keys.

Beneath these colors is a set of 16 "Custom Colors". Initially, these colors are all white. The user can select the "Define Custom Colors" option to define new colors. This pulls up a color chart that the user can select from. After selecting a custom color, the user can add it to the set of 16 custom colors by selecting the "Add to Custom Colors" button. This color is then available for future selection.

The user selects the “OK” button (or presses <enter>) to complete selection of the color. The RGB value of the color selected is returned in WPAL-RED, WPAL-GREEN, and WPAL-BLUE.

Alternately, the user can select the “Cancel” button (or press <escape>). In this case, no color is returned and the value of RESULT is set to WPERR-CANCELLED (see below).

The initial default color selection is black (RGB value “0”, “0”, “0”). You may supply a different default value by moving the desired color into WPAL-RED, WPAL-GREEN, and WPAL-BLUE. You must also set WPAL-FLAGS to WPAL-USE-DEFAULT (value “1”). If you do not use this option, then you should set WPAL-FLAGS to zero before calling W$PALETTE.

The color selection box is a standard component of Windows and Windows NT.

Note: In some Windows setups “COMMDLG.DLL” provides support for the color selection dialog (as well as other standard dialogs). Because ACUCOBOL-GT is used to build applications for use in a variety of nations and languages, we do not distribute this file. The standard dialogs defined in this DLL contain text that is specific to the local language.

WPALETTE-SET-USER-COLOR (op-code 7)

This operation allows you to assign selected system colors to COBOL color numbers. It uses two fields in the WPALETTE-DATA group item.

WPAL-COLOR-ID contains the number of the COBOL color you want to change. Values range from “1” (low-intensity black) to “16” (high-intensity white).

WPAL-USER-COLOR-ID contains the number of the system color to use. You can use either WPUSER-COLOR-3D, which matches the system’s color for the background of 3-D objects, or WPUSER-COLOR-BACKGROUND, which matches the system’s color for window backgrounds. These values are defined in “palette.def”.

The following example redefines the COBOL color number “8” (which is gray in the default palette) to be the same as the system’s 3-D color:

```
MOVE 8 TO WPAL-COLOR-ID
MOVE WPUSER-COLOR-3D TO WPAL-USER-COLOR-ID
CALL "W$PALETTE" USING
   WPALETTE-SET-USER-COLOR, WPALETTE-DATA
```

Errors are handled in the same manner as the WPALETTE-SET-COLOR function. Note that under some systems, such as Microsoft Windows, the system colors can change dynamically at runtime. If this occurs, the runtime will automatically remap the colors in the palette as needed to match the new system colors.

For additional color configuration options, see the USER-GRAY, USER-WHITE, and USER-COLORS options of the DISPLAY FLOATING WINDOW verb.

Error Handling

RESULT is set to a positive value if the call to W$PALETTE is successful. Except for the WPALETTE-SUPPORTED and WPALETTE-NUM-COLORS operations, the success value is always “1”. A zero or negative value indicates a problem. The following values are possible:

WPERR-UNSUPPORTED (value “0”) -- Either the requested operation cannot be performed on the host system, or you passed an invalid operation code.

WPERR-BAD-ARG (value “-1”) -- Either you did not pass WPALETTE-DATA when it was required, or WPALETTE-DATA contains some invalid data.

WPERR-CANCELLED (value “-2”) -- The user selected the “cancel” operation of the color selection box, or the user closed the selection box.

W$PROGRESSDIALOG

This routine provides a general way to show a user how an operation is progressing. It is typically used when deleting, copying, moving, uploading, or downloading large files or a large number of files. It can also be used when performing a time-consuming operation that you want to allow the user to cancel at any time.

Usage

```
CALL "W$PROGRESSDIALOG"
   USING OP-CODE, parameters
   GIVING WPROGRESS-RESULT.
```

Parameters

OP-CODE Numeric parameter

This indicates the desired operation. Level 78 symbolic names for these operations can be found in “acugui.def”.

parameters Vary depending on the op-code chosen

Description

W$PROGRESSDIALOG provides access to the features of the Windows progress dialog box, which is exposed through the IProgressDialog COM interface. This interface is part of the BROWSEUI.DLL Windows system library and was originally part of Internet Explorer 5.

W$PROGRESSDIALOG can be used to create a modal or modeless window containing a progress dialog, set its title, animation, text lines, progress, and cancel message. The progress dialog can be configured to automatically estimate and display the time remaining until the operation completes.

The progress dialog runs on a background thread. This allows the progress dialog to update its display, estimate the time remaining until the operation completes, and handle the user cancelling the operation independently of the work being done by the COBOL program. The progress dialog updates and

remains responsive even during long operations such as C$COPY across a thin client connection. There is no need to set the FILE_IO_PROCESSES_MESSAGES configuration variable.

OP-CODES and Parameters

WPROGRESSDIALOG-CREATE (op-code 1)

This operation creates and starts the progress dialog. A handle to the progress dialog is returned in the data item specified in the GIVING clause. This handle should be stored in a variable declared as USAGE HANDLE.

This op-code takes six additional optional parameters, which appear in order below:

title
A literal or data item containing the text that will appear in the title bar of the progress dialog.

cancel-message
A literal or data item containing the text that is shown on line 3 (underneath the progress bar) when the user clicks the Cancel button. Since the progress dialog operates on a separate background thread there will be a delay between the time the user presses the Cancel button and the time the COBOL program calls W$PROGRESSDIALOG WPROGRESSDIALOG-QUERY-CANCEL. Since this delay might be significant, the progress dialog provides the user with immediate feedback by clearing text lines 1 and 2 and displaying the cancel message on line 3. The message is intended to let the user know that the delay is normal and that the progress dialog box will be closed shortly. Typically, it is set to something like “Please wait while ...”.

flags
A numeric literal or data item, which specifies flags that determine the operation of the progress dialog. This can be a combination of the following values:

WPROGRESSDIALOG-NORMAL (value 0)
Normal progress dialog behavior.

WPROGRESSDIALOG-MODAL (value 1)

The progress dialog box will be modal to the current window. By default, a progress dialog box is modeless.

WPROGRESSDIALOG-AUTOTIME (value 2)

Automatically estimate the remaining time and display the estimate on line 3. If this flag is set, WPROGRESSDIALOG-SET-LINE can be used only to display text on lines 1 and 2.

WPROGRESSDIALOG-NOTIME (value 4)

Do not show the “time remaining” text.

WPROGRESSDIALOG-NOMINIMIZE (value 8)

Do not display a minimize button on the dialog box’s title bar.

WPROGRESSDIALOG-NOPROGRESSBAR (value 16)

Do not display a progress bar. Normally, an application can quantitatively determine how much of the operation remains and periodically pass that value to WPROGRESSDIALOG-SET-PROGRESS. The progress dialog uses this information to update its progress bar. This flag is typically set when the calling application needs to wait for an operation to finish but does not have any quantitative information it can use to update the dialog box.

animation-type

A numeric literal or data item, which specifies the type of AVI clip that will run in the dialog box. It can be one of the following values:
WPROGRESSDIALOG-ANIMATION-NONE (value 0)
WPROGRESSDIALOG-ANIMATION-FILECOPY (value 1)
WPROGRESSDIALOG-ANIMATION-FILEMOVE (value 2)
WPROGRESSDIALOG-ANIMATION-FILEDEL (value 3)
WPROGRESSDIALOG-ANIMATION-FILEDELR (value 4)
WPROGRESSDIALOG-ANIMATION-FILENUKE (value 5)
WPROGRESSDIALOG-ANIMATION-SEARCH (value 6)
WPROGRESSDIALOG-ANIMATION-FINDCOMP (value 7)
WPROGRESSDIALOG-ANIMATION-FINDFILE (value 8)
WPROGRESSDIALOG-ANIMATION-CUSTOM (value 99)

If you specify WPROGRESS-ANIMATION-CUSTOM, there are two additional parameters:

resource-dialog

A literal or data item containing the name of a DLL or EXE that includes the AVI file as a resource.

resource-id

The resource id of the AVI file in the file specified by resource-dialog.

For example, if you want to use an AVI file named MYANIMATION.AVI that is included as a resource in a DLL named MYRES.DLL with ID 1007, you can specify it with the following parameters:

```
CALL "W$PROGRESSDIALOG" USING WPROGRESSDIALOG-CREATE
"Title" "Cancel Message"
WPROGRESSDIALOG-MODAL
WPROGRESSDIALOG-ANIMATION-CUSTOM
"MYRES.DLL"
1007
GIVING PD-HANDLE.
```

Note: The requirement for the AVI file to be a resource is a limitation of the Microsoft IProgressDialog COM interface. There is no provision for loading an AVI file directly.

WPROGRESSDIALOG-DESTROY (op-code 2)

This operation destroys the progress dialog box. It takes only one parameter, the handle of the progress dialog returned by WPROGRESSDIALOG-CREATE.

WPROGRESSDIALOG-SET-PROGRESS (op-code 3)

This operation updates the progress dialog box with the current state of the work being monitored. It takes three parameters:

handle

The handle of the progress dialog returned by WPROGRESSDIALOG-CREATE.

completed

A numeric literal or data item specifying an application-defined value that indicates what proportion of the work has been completed so far.

total

A numeric literal or data item specifying an application-defined value that specifies what value the 'completed' parameter will have when the work is complete.

WPROGRESSDIALOG-QUERY-CANCEL (op-code 4)

This operation checks whether the user has pressed the cancel button. You must periodically use this function to poll the progress dialog box object to determine whether the operation has been canceled.

This operation takes one parameter, the handle of the progress dialog returned by WPROGRESSDIALOG-CREATE. It returns "1" if the user has pressed the cancel button, "0" otherwise, in the data item specified with the GIVING clause or in the special RETURN-CODE register.

WPROGRESSDIALOG-SET-LINE (op-code 5)

This operation sets the text lines that are displayed in the progress dialog. It takes these parameters:

handle

The handle of the progress dialog returned by WPROGRESSDIALOG-CREATE.

string

A data item containing the text to display.

line-num

A numeric literal or data item containing the line number on which the text is to be displayed. This can be either 1, 2, or 3. If WPROGRESSDIALOG-AUTOTIME was specified in the flags parameter when the progress dialog was created, then only lines 1 and 2 can be used. The estimated time will be displayed on line 3.

compact-path

A numeric literal or data item whose value is “1” or “0”. The default value for this parameter is “1”, set it to “0” to turn off path string compaction or the following described behavior. There is a defect in the Microsoft's dialog design of the text string field. If a string is longer than what the field can contain the default behavior is to wrap rather than clip the text. Only the top-most pixels of the wrapped text can be seen and so is cosmetically unpleasing and functionally useless. However, if compact-path is a 1 then text that is too long for the field will be truncated and an ellipsis (...) gets appended to the end of the string. This is better default behavior for ACUCOBOL-GT programmers.

Note: This parameter has no effect on strings or paths less than the field width.

WPROGRESSDIALOG-RESET-TIMER (op-code 6)

This operation resets the progress dialog box timer to zero. It takes one parameter, the handle of the progress dialog returned by WPROGRESSDIALOG-CREATE.

The timer is used to estimate the remaining time. It is started when your application calls WPROGRESSDIALOG-CREATE. Unless your application will start immediately, it should call WPROGRESS-RESET-TIMER just before starting the work. This practice ensures that the time estimates will be as accurate as possible. This method should not be called after the first call to WPROGRESSDIALOG-SET-PROGRESS.

WPROGRESSDIALOG-C-COPY (op-code 7)

This operation specifies that the progress dialog should monitor the progress of the C$COPY when transferring files to or from a remote machine in a thin client environment. It takes one parameter, the handle of the progress dialog returned by WPROGRESSDIALOG-CREATE.

W$STATUS

W$STATUS works with the ACUCOBOL-GT Web Runtime. It tells the runtime to display a text message in the browser's status bar. See the the manual, *A Programmer's Guide to the Internet*, for information about the Web Runtime.

Usage

```
CALL "W$STATUS"
    USING STATUS-MESSAGE
```

Parameter

STATUS-MESSAGE PIC X(n)

Contains the message to be displayed in the browser's status bar.

This routine is available only when the calling COBOL program is running in a Web browser window via the ACUCOBOL-GT Web Runtime. The routine is not available to programs run by the standard runtime when the standard runtime is executed by a Web browser. The RETURN-CODE register is set to "1" after a successful call and set to "0" if this routine is unavailable.

W$TEXTSIZE

The W$TEXTSIZE routine allows you to measure the height and width of a string of text in a particular font.

Usage

```
CALL "W$TEXTSIZE"
    USING TEXT-STRING, TEXTSIZE-DATA
```

Parameters

TEXT-STRING PIC X(n)

Contains the string to be measured

TEXTSIZE-DATA PIC X(n)

The TEXTSIZE-DATA data item is found in the COPY file “acugui.def”. It is defined as follows:

```
01 TEXTSIZE-DATA.
   03 TEXTSIZE-FONT      HANDLE OF FONT.
   03 TEXTSIZE-WINDOW    HANDLE OF WINDOW.
   03 TEXTSIZE-SIZE-X    PIC 9(7)V99  COMP-4.
   03 TEXTSIZE-CELLS-X   PIC 9(7)V99  COMP-4.
   03 TEXTSIZE-BASE-X    PIC 9(9)     COMP-4.
   03 TEXTSIZE-SIZE-Y    PIC 99V99    COMP-4.
   03 TEXTSIZE-CELLS-Y   PIC 99V99    COMP-4.
   03 TEXTSIZE-BASE-Y    PIC 9(4)     COMP-4.
   03 TEXTSIZE-FLAGS     PIC X        COMP-X.
      88 TEXTSIZE-STRIP-SPACES VALUE 1, FALSE ZERO.
```

Description

The W$TEXTSIZE routine measures the average height and width of TEXT-STRING according to the parameters found in TEXTSIZE-DATA and returns the results in TEXTSIZE-DATA. The measurement is returned using a variety of units.

The data elements in TEXTSIZE-DATA are used as follows:

Input Items:

TEXTSIZE-FONT (HANDLE OF FONT) -- This item holds the handle of the font that you want the runtime to measure. If the value is NULL, DEFAULT-FONT is used. TEXTSIZE-FONT is initialized to NULL in “acugui.def”.

TEXTSIZE-WINDOW (HANDLE OF WINDOW) -- This item holds the handle of the window that you want to use when measuring the number of window cells the text occupies (see TEXTSIZE-CELLS-X and TEXTSIZE-CELLS-Y below). If this value is NULL, the current window is used. If there is no current window, the “cells” measurement is returned with the value zero. TEXTSIZE-WINDOW is initialized to NULL in “acugui.def”.

TEXTSIZE-FLAGS (PIC X COMP-X) -- If TEXTSIZE-STRIP-SPACES is true, W$TEXTSIZE does not include any trailing spaces in the measurement of TEXTSTRING. Otherwise, trailing spaces are included in the measurement. TEXTSIZE-FLAGS is initialized to NULL in "acugui.def".

Output Items:

TEXTSIZE-SIZE-X (PIC 9(7)V99 COMP-4) -- Returns the width of the text in *label size* units. This is the SIZE value required for a label to exactly contain the text. It is computed by dividing the length of the text by the size of the font's "0" character. For fixed-pitch fonts, the value returned is the same as the number of characters in TEXT-STRING.

TEXTSIZE-SIZE-Y (PIC 99V99 COMP-4) -- Returns the height of the text in *label size* units. This is the LINES value required for a label to exactly contain the text. By definition, this value is always "1".

TEXTSIZE-CELLS-X (PIC 9(7)V99 COMP-4) -- Returns the width of the text in window cells. It is computed by dividing the width of the text by the width of a window cell.

TEXTSIZE-CELLS-Y (PIC 99V99 COMP-4) -- Returns the height of the text in window cells. It is computed by dividing the height of the text by the height of a window cell.

TEXTSIZE-BASE-X (PIC 9(9) COMP-4) -- Returns the width of the text in *base units*. On a graphical system, a base unit is a pixel. On a character-based system, a base unit is a character cell.

TEXTSIZE-BASE-Y (PIC 9(4) COMP-4) -- Returns the height of the text in *base units*. On a graphical system, a base unit is a pixel. On a character-based system, a base unit is a character cell.

WIN$PLAYSOUND

The WIN$PLAYSOUND routine lets you play a ".WAV" file on Microsoft Windows machines (wave-form sound). You can also play sounds that the user has assigned to system events in the control panel.

WIN$PLAYSOUND is supported and can be used by applications deployed in our Thin Client environment.

Usage

```
CALL "WIN$PLAYSOUND"
    USING SOUND-NAME, SOUND-FLAGS
    GIVING SOUND-STATUS
```

Parameters

SOUND-NAME PIC X(n)

Identifies the sound to play. This is either the name of a registered system sound or the name of a ".WAV" file.

SOUND-FLAGS numeric parameter

One or more optional values added together. The SOUND-FLAG options are described below. The option names are contained in the COPY library "acugui.def".

SOUND-STATUS signed numeric data item

Indicates the status of the operation as follows:

-1	Operation not available - host machine is not Windows
0	Operation failed
1	Operation succeeded

Description

WIN$PLAYSOUND causes the sound specified in SOUND-NAME to be played. If SOUND-NAME contains the name of a system event, the sound associated with that event is played (the association is made via the Windows Control Panel). Otherwise, WIN$PLAYSOUND assumes that SOUND-NAME contains the name of a ".WAV" audio file.

Note: When you are running in a thin client environment, and a file name beginning with "@[DISPLAY]" is passed to this routine, it will attempt to access the file in the display host's file system. It does not download the file from the server. For more information, refer to section 7.2, "Using Library Routines and DLLs in Thin Client." of the *AcuConnect User's Guide*.

If SOUND-NAME does not correspond to a system event and the file cannot be found, the default system sound is played. The default sound is also played when there is not enough memory available to load the specified file. If a default sound is not available, the routine does nothing and returns "0" in SOUND-STATUS.

This routine searches for the specified ".WAV" file in the object libraries, the working directory, and then the directories specified in the PATH environment variable. You can add ".WAV" files to your object library by using the COPY RESOURCE statement or CBLUTIL utility program. For more information about the COPY RESOURCE statement, see **Section 2.4.1** in Book 3, *ACUCOBOL-GT Reference Manual*. For more information about the CBLUTIL utility program, see **Section 3.2** in Book 1, *ACUCOBOL-GT User's Guide*. Specifying a SOUND-NAME of spaces stops any sound that is currently playing.

Note: The behavior of this library routine is affected by the setting of the FILENAME_SPACES configuration variable that may or may not allow spaces in a file name. See the documentation on FILENAME_SPACES in **Appendix H, "Appendix H: Configuration Variables,"** for information about the terminating character for path names.

System event names are implementation dependent. The Windows API documents that the following sounds are always available:

SystemAsterisk

SystemExclamation

SystemExit

SystemHand

SystemQuestion

SystemStart

Other system sound events are defined in the registry under Windows 98. Use “regedit” to look in the registry location:

HKEY_CURRENT_USER\AppEvents\EventLabels

The naming conventions for system sound events is implementation dependent.

The following options can be specified in SOUND-FLAGS. To use them, add together the values of the options and assign them to SOUND-FLAGS. The optional values have level 78 names associated with them. These names are defined in “acugui.def”.

SND-SYNC (value 0) -- This option causes the program to pause while the sound is being played. WIN$PLAYSOUND will not return until the sound has finished.

SND-ASYNC (value 1) -- This option causes the program to continue to run while the sound is playing. Note that you can halt a sound that is playing by passing a SOUND-NAME of spaces to a subsequent call to WIN$PLAYSOUND.

SND-LOOP (value 8) -- To work, this option must be used with the SND-ASYNC option. This option causes the sound to play continuously, restarting from the beginning when the end is reached. The sound can be stopped by passing a SOUND-NAME of spaces on a subsequent call.

SND-NOSTOP (value 16) -- Normally, any sound playing will be stopped when a new sound is specified. With NOSTOP, if a sound is already playing, it will continue to play and WIN$PLAYSOUND will return a SOUND-STATUS value of “0”.

Printing with theWindows Print Spooler (-Q and -P)

In Windows, there are two ways that your application can print something. It can either send data directly to an output device (by opening the appropriate port), or it can use the Windows print spooler. Most applications use the spooler because it allows the user to queue-up a series of documents and then print them out *in the background*, while performing other tasks. In this regard, the Windows print spooler works much like spoolers common to other operating systems (for example, the lp program on UNIX machines).

In other ways, however, the Windows print spooler is very different from classical spoolers. These differences derive from the graphical nature of Windows and Windows applications and can affect what you can accomplish from COBOL. This section discusses how the Windows print spooler works and how it affects your programs. It includes information on using the "-Q" and "-P" configuration options to assign printers to the print spooler and details for using these options with the **WIN$PRINTER** library routine.

Programs that use traditional spoolers usually work like this:

1. The program sends text, data, and printer-control sequences to the spooler.
2. The spooler saves the data on disk. As resources become available, the spooler sends the information on to the port driver that manages the system's printer.
3. The port driver delivers the data stream to the printer, which produces the document.

In this scenario, the program provides all of the printer-control coding. The spooler and port driver simply coordinate transfer of the data to the printer.

The print spooling method under Windows is different. The sequence of operations typically works like this:

1. The program calls the Windows graphical Application Programming Interface (API) to describe a logical image of each page in the document. This Windows graphical API is called the Graphical Device Interface (GDI).

2. The GDI subsystem constructs a low-level description of each page, which is passed to the print driver. The print driver and GDI work together to construct the data stream needed to produce the proper output on the printer. This data stream is temporarily stored on disk.

3. As resources become available, the spooler sends the disk data to the port driver for the printer. From this point on, the process is the same as for a traditional spooler.

The operation of the spooler itself is very similar under both scenarios. What is different is that, under Windows, the GDI and print driver are responsible for producing the printer-control sequences, while in the traditional model, the program produces the control sequences.

This approach to the printing process allows Windows applications to produce graphical output without knowing how to drive specific types of printers. This greatly simplifies the printing task for sophisticated programs such as word processors and drawing programs.

However, for simple reports, this is much more complicated than the traditional approach. Instead of simply sending text data and carriage control codes, the application must go through an involved process of getting a *device context*, selecting and measuring an appropriate font, formatting lines of text and drawing them to the device context, and maintaining the necessary line and position information.

Fortunately, ACUCOBOL-GT simplifies this work. The runtime system contains print drivers that know how to simulate traditional style printing using the Windows spooler. To take advantage of these drivers, simply assign the print file to "-Q <*printername*>" or to "-P SPOOLER", as described in the following sections. You can change fonts in the middle of a report when using the Windows spooler. Simply select the new font via the **W$FONT** library routine while the print file is open. You can change fonts at any time, even mid-line. Make sure that WFDEVICE_WIN_PRINTER is set to TRUE before you call W$FONT. When advancing lines, the runtime uses the height of the selected font to determine the height of the line, and the font must be associated with the selected printer.

Note: The process used by Windows to print (via the GDI) interferes with programs that attempt to control the printer directly. Programs that embed control codes in their print data (to perform various functions such as changing the printer's pitch, shifting to *compressed print* mode, or drawing a form on a laser printer) will not work under Windows because the GDI does not understand the control codes. Instead, it tries to treat the codes as regular text data to be drawn.

The process used by Windows has the advantage that an application does not need to know how to drive an individual printer, but has the disadvantage that an application cannot choose to drive the printer directly.

If you have an application that needs to control the printer directly, you have three choices:

1. Examine the WIN$PRINTER routine described below. It allows you to perform some basic control operations directly from COBOL.
2. Send the report directly to the printer by assigning the print file to the printer's device. This allows you to directly control the printer, but you lose the advantages of using the Windows print spooler.
3. Use the GDI to describe the print image you want. This generally involves calling an external function written in C or some other language that has direct GDI support. This choice provides the greatest flexibility, but can be a large amount of work.

-Q <*printername*>

If you want the Windows spooler to format the pages of your report, but you want to use a particular printer, assign your print file to:

```
PRINTER1 -Q \\printername
```

in the configuration file ("CBLCONFI"). *Printername* is the printer designation as given in the Printers folder under Settings in the Start menu. The name may be up to 80 characters long and contain embedded spaces. The name may not include the semicolon character (;) or be surrounded by single or double quotes. The pages are printed in the manner described in "-P SPOOLER" below. The sample programs "graphprn.cbl" and "prndemox.cbl" contain examples of these functions.

To determine a valid *printername*, use the WIN$PRINTER library routine to obtain the name of the desired printer. (This is described under the **"WINPRINT-SET-PRINTER"** operation code in "Specifying a Printer.") Then add the following line to your code:

```
MOVE "-Q \\printername" TO WS-PRINTER-NAME.
```

When the runtime opens a file assigned to "-Q <*printername*>" it sets the Windows print spooler to use this printer. The printer driver must be installed on the computer from which you print. If *printername* is not recognized by the runtime, a dialog box allows you to choose a printer manually.

Setting Options

You may also use "-Q <*printername*>" to set several other printing options in the configuration file using the following syntax:

```
<-Q printername>[;option1=x][;option2=x][;option3=x]...
```

The following options may appear in any order. Options not supported by the printer driver are ignored. *Printername* should appear as shown in WINPRINT-NAME, but the options are case insensitive.

Note: The options PITCH, COLS, LINES, and FONT are all mutually dependent. Omitting one or more of these options may cause the resulting printout to look wrong.

%%ACU-CURRENT%%	May be set immediately after the -Q instead of providing a **printer name**. This specifies to use whatever printer is currently considered the runtime's default printer. NOTE: This entry is case sensitive, it must be all upper case.
%%WINDOWS-STANDARD%%	May be set immediately after the -Q instead of providing a **printer name**. This specifies to use whatever printer is currently considered the Windows default printer. NOTE: This entry is case sensitive, it must be all upper case.

CHARSET	Specifies one of the character sets defined in “fonts.def”. Refer to the supported values described in the table below. If you use CHARSET, you must also use FONT. If CHARSET is not specified, it has the same effect as CHARSET=WIN-DEFAULT. To specify an alternative character set, the necessary fonts must be present on the computer.
COLLATE	Specifies that multiple print copies should be collated. In order for COLLATE to have any effect, COPY must also be set to a value greater than 1. COLLATE takes the following values: **0, NO, FALSE** Implies no collating, or collating off. **1, YES, TRUE** Specifies collating, or collating on.
COLOR	Indicates to print in color. COLOR may be set to a legal RGB color code number. See the library function **WINPRINT-SET-TEXT-COLOR** for more information on the color value.
COLS	Specifies the number of columns (width) on the page. This number is not validated by the runtime or the spooler. Choose a number of columns that coordinate with the selected font and pitch when designing the report layout.
COPY	If your printer supports this feature, COPY allows you to specify the number of copies to print.
DIRECT	Setting DIRECT to “ON” causes the job to print as if the configuration file was set to “-P SPOOLER-DIRECT” (described later in this section). This option also disables any use of additional options. Setting DIRECT to the default of “OFF” causes the job to print to the selected printer as if the configuration file was set to “-P SPOOLER”.

DUPLEX	If supported by the printer, enables printing on both sides of the paper. If not supported by the printer, single-sided printing occurs without any "warning." DUPLEX takes the following values: **1, FALSE, or NO** This is the default value and implies no duplex. **2, TRUE, or YES** Implies vertical duplexing **3** Implies horizontal duplexing
FONT	Use FONT to specify a single font name. The font name may have embedded spaces, but may not contain double or single quotes. If the font does not exist, the closest matching font is chosen. The runtime does not align columns. If you are printing a report containing columns, you should use a fixed-width font.
LINES	Specifies the lines (rows of characters) on the page. This number is not validated by the runtime or the spooler. Choose a number of lines that is compatible with the selected font and pitch when designing the report layout.
ORIENTATION	If your printer supports this feature, ORIENTATION allows you to specify LANDSCAPE or PORTRAIT orientation for the report. The default value of ORIENTATION is driver specific.

PAPER	Specifies the paper size to be used for the print job. PAPER is set to a paper format number for the target printer and the target tray. See library function **WINPRINT-GET-PRINTER-MEDIA** for more information on paper formats. NOTE: Paper formats and paper trays are individual and unique to each printer installation. They can also be modified by users. Because of this, you should use a "calibration" routine when installing your software. This is a program that will list the available printers, paper formats, and trays. The output from the calibration program can then be used to assign the PAPER and PAPERTRAY properties. A sample calibration program called "PaperInfo.cbl" is provided in the samples directory of ACUCOBOL-GT. The output from this program appears in the **Example section** and is used to demonstrate the setting of PAPER and PAPERTRAY.
PAPERTRAY	Specifies the paper size to be used for the print job. PAPER is set to a paper format number for the target printer and the target tray. See library function **WINPRINT-GET-PRINTER-MEDIA** for more information on paper formats. See the important note for the PAPER variable. Also note that if there is no match between the paper format designated to a tray in Windows and the value you set in PAPER, the paper choice will take precedence and the printer driver will choose the tray that supports your paper choice.
PITCH	This value specifies the point size of the font. Pitch does not determine the number of characters per line. If you use a larger pitch, the characters simply appear more crowded. For example, when you are printing 132 columns, a pitch of 10 produces better character spacing than a pitch of 12.
SPOOLER-RESET	Resets the printer to its start up defaults. SPOOLER-RESET takes the following values: **0, NO, FALSE** No change. **1, YES, TRUE** Reset printer to start up defaults.

CHARSET Values

CHARSET can take one of the following values:

Variant 1	Variant 2	Variant 3
DEFAULT	WFCHARSET-DEFAULT	1
WIN-OEM	WFCHARSET-WIN-OEM	2
WIN-SYMBOL	WFCHARSET-WIN-SYMBOL	3
WIN-SHIFTJIS	WFCHARSET-WIN-SHIFTJIS	4
WIN-HANGUL	WFCHARSET-WIN-HANGUL	5
WIN-GB2312	WFCHARSET-WIN-GB2312	6
WIN-CHINESEBIG5	WFCHARSET-WIN-CHINESEBIG5	7
WIN-JOHAB	WFCHARSET-WIN-JOHAB	8
WIN-HEBREW	WFCHARSET-WIN-HEBREW	9
WIN-ARABIC	WFCHARSET-WIN-ARABIC	10
WIN-GREEK	WFCHARSET-WIN-GREEK	11
WIN-TURKISH	WFCHARSET-WIN-TURKISH	12
WIN-VIETNAMESE	WFCHARSET-WIN-VIETNAMESE	13
WIN-THAI	WFCHARSET-WIN-THAI	14
WIN-EASTEUROPE	WFCHARSET-WIN-EASTEUROPE	15
WIN-RUSSIAN	WFCHARSET-WIN-RUSSIAN	16
WIN-MAC	WFCHARSET-WIN-MAC	17
WIN-BALTIC	WFCHARSET-WIN-BALTIC	18

Examples

To use the Windows spooler with an HP Laserjet printer driver located on SERVER1, and specify the font, font size, width and number of lines in the report, enter the following into “CBLCONFI”:

```
PRINTER1 -Q \\SERVER1\HP Laserjet IV;FONT=Times New Roman;PITCH=12;COLS=132;LINES=65.
```

To print three copies directly to the printer on a server named GUTENBERG in Landscape orientation, enter the following into “CBLCONFI”:

```
PRINTER1 -Q \\GUTENBERG\HP Laserjet IV;DIRECT=ON;ORIENTATION=LANDSCAPE;COPY=3
```

To specify the Greek character set on a server named SERVER5, enter the following into "CBLCONFI":

```
PRINTER1 -Q \\SERVER5\Laserjet;FONT=Courier New;PITCH=12;LINES=60;COLS=80;CHARSET=11
```

Output from sample program - PaperInfo.cbl

PaperInfo.cbl is provided in the sample directory of "AcuGT". Its output provides values that can be used with **PAPER** and PAPERTRAY.

```
Dell Laser MFP 1815 PCL 6
Paper formats
  (001) - Letter 8 1/2 x 11 in
  (005) - Legal 8 1/2 x 14 in
  (007) - Executive 7 1/4 x 10 1/2 in
  (009) - A4 210 x 297 mm

Paper Trays
  (007) - Auto
  (001) - Upper tray
  (006) - Manual envelope
```

To direct your print to the printer Dell Laser MFP 1815 PCL 6, on paper format A4 210 x 297 mm and using the Upper tray, enter the following into "CBLCONFI":

```
PRNFILE -Q Dell Laser MFP 1815 PCL 6;COPY=2;COLLATE=1
```

-P SPOOLER

If you want to use the default printer and font, simply assign your print file to "-P SPOOLER". For example, to assign "PRINTER1" to the spooler, enter the following line in your COBOL configuration file ("CBLCONFI"):

```
PRINTER1  -P  SPOOLER
```

By default, the runtime system assigns the "PRINTER" device to the spooler. You may change this in the configuration file by assigning "PRINTER" to some other name.

When the runtime opens a file assigned to "-P SPOOLER", it automatically initiates a job with the Windows spooler and constructs print pages in accordance with your program. The runtime uses the default printer and font. If the user looks for the job in the spooler, it is named with the current title of the ACUCOBOL-GT window.

Note: The Windows spooler operates by *drawing* your report on each page. It constructs its own control codes to handle formatting. If you assign your print file to "-P SPOOLER" and your file contains device-dependent control sequences (such as those used to shift to a condensed font, or to print a form and then fill it in), *the codes will be passed to the spooler as data* and thus will not be interpreted correctly. If you have reports that depend on embedded control codes, print those directly to the device, or assign the print file to "-P SPOOLER-DIRECT," as described in the section "*Direct Control*" below.

Direct Control

If you want to control the format of the printout yourself using embedded control codes, simply assign your print file to "-P SPOOLER-DIRECT" or to "-Q <*printername*>" using the "DIRECT=ON" option. For example, to assign the print job "PRINTER1" to the spooler and retain direct control over formatting, enter the following line in your COBOL configuration file ("CBLCONFI"):

```
PRINTER1  -P  SPOOLER-DIRECT
```

Or, use the following command to assign PRINTER1 to the spooler for printing to a specific printer while retaining direct formatting control:

```
PRINTER1 -Q printername;DIRECT=ON
```

Both of these methods cause the print job to be sent to the printer via the Windows spooler, but the program does not use the spooler to format the pages. You must use embedded control codes to handle formatting (much as you would under UNIX if you used the UNIX spooler).

When using the "-P SPOOLER-DIRECT" option, you may use the WIN$PRINTER library routine to choose a printer. But because you completely control the printer, the various options provided by WIN$PRINTER are ignored. For example, WIN$PRINTER does not set the

page size, page orientation, or font. Information returned from WIN$PRINTER, such as number of lines and columns on the page, may not be accurate and should not be used.

Printing Multiple Jobs Simultaneously

If you need to print multiple jobs at the same time, you must open multiple File Descriptors that point to "-P SPOOLER" or "-P SPOOLER-DIRECT" simultaneously. For example, you may have two simultaneous print jobs:

```
SELECT FIRST-FILE
       ASSIGN TO PRINTER "-P SPOOLER".

SELECT SECOND-FILE
       ASSIGN TO PRINTER "-P SPOOLER".

..

PROCEDURE DIVISION.

..

       OPEN OUTPUT FIRST-FILE.
       OPEN OUTPUT SECOND-FILE.
```

and both will print to the default Windows printer without interfering with each other. You can call WIN$PRINTER using WINPRINT-SETUP or WINPRINT-SETUP-USE-MARGINS before one or both of the OPEN statements. Each file may have individual file status variables, or may refer to a common file status variable.

This does not mean that you can open a single File Descriptor multiple times. For example, the following will return file status indicating that the file is already opened:

```
SELECT FIRST-FILE
       ASSIGN TO PRINTER "-P SPOOLER".

..

PROCEDURE DIVISION.

..
```

```
OPEN OUTPUT FIRST-FILE.
OPEN OUTPUT FIRST-FILE.
```

This is normal behavior and is consistent with the way file handling is implemented in COBOL and in other programming languages.

If you are using only the verbs OPEN, CLOSE, and WRITE, no further changes to your code are needed. If you are using WIN$PRINTER (other than WINPRINT-SETUP or WINPRINT-SETUP-USE-MARGINS), you need to specify which print job is affected. This can be done in one of two ways:

1. The simplest way is to execute the WIN$PRINT operation immediately after an OPEN or WRITE statement on the intended job. Every execution of OPEN and WRITE sets the current job as the default so that subsequent activity using WIN$PRINTER is automatically directed to the job that was last accessed with an OPEN or WRITE statement.

   In this situation, if you have multiple jobs running, and you close one of them, the runtime switches to the next job in the list. For example, if you are printing jobs 1, 2, and 3, and you close job 2, the close command sets the current job to 3. If there is no job 3, the runtime attempts to set to the job that preceded the closed job (which in this case is job 1). If there are no jobs, the current job is initialized.

2. The other method is to use the WINPRINT-SET-JOB operation of the WIN$PRINTER library routine.

WIN$PRINTER

The WIN$PRINTER library routine is designed to enhance the ability of COBOL to take advantage of the Windows print spooler. This routine is available on all systems that run ACUCOBOL-GT, but is useful only with Microsoft Windows. Not all printer drivers are supported by this routine.

WIN$PRINTER configures the Windows print spooler only and cannot be used to configure the printer directly.

You must assign your print file to "-P SPOOLER" or to "-Q <*printername*>" in the configuration file to access the Windows print spooler. This is described in the previous section, "*Routine to Handle the Windows Print Spooler.*"

If you have assigned your print file to "-P SPOOLER-DIRECT" or to "-Q <*printername*>" using the "DIRECT=ON" option, then you retain control over the format of the pages. In this situation, WIN$PRINTER can be used to select a printer, but not to print bitmaps or determine paper size, page orientation, fonts, margins, and the like. The information returned by WIN$PRINTER about the numbers of lines and columns on the page may not be accurate in this situation and should not be used.

CAUTIONS

Just as with changes to fonts and menus, modifications to Windows printer settings are global for a specific instance of the runtime. Settings established in one COBOL application will affect subsequent COBOL applications performed in the same runtime instance. If you do not want settings to apply globally in this case, you must reverse the settings manually within the program. However, these modified printer settings will not affect later executions unless you store the settings and reactivate them at the next instance of the runtime. Windows global settings, other windows applications and other instances of the runtime are not affected by changes made by the WIN$PRINTER library routine.

When you change Windows settings in general, and printer settings in particular, Windows posts a message informing its subsystems of the change. The runtime looks for these messages and passes that information on to the WIN$PRINTER operation codes when they are called. This may cause inconsistencies between the information stored in the COBOL program and the runtime, such as the order of printers in the internal printer list. To avoid this problem, always use the printer name, rather than the printer number, when calling op-codes that require a printer identity. The name will always be unique, while the number is relative to the internal printer list and may not be accurate.

WIN$PRINTER always performs the printer number test before the printer name test. This means that if WINPRINTER-NO-OF-PRINTERS is a positive number, the function will look for that printer number before looking for the printer name. This could result in a WPRTERR-BAD-ARG error. To

overrule this ranking and use the printer name, set the argument WINPRINT-NO-OF-PRINTERS to zero before accessing printer-specific information.

Usage

```
CALL "WIN$PRINTER"
    USING OP-CODE, parameters,
    GIVING RESULT
```

Parameters

OP-CODE

A numeric value that selects which WIN$PRINTER function to perform. The op-codes are defined in “winprint.def” and described in the WIN$PRINTER op-codes section below.

RESULT

A signed numeric data item that returns the status of the operation. The data type of the returned value is SIGNED-INT or PIC S9(9) COMP-5. Unless otherwise stated below, “1” indicates success, and a zero or negative result indicates failure.

OP-CODE Parameters

The remaining parameters vary depending on the operation code chosen. They provide information and hold results for the operations specified. Parameters may be omitted from those operations that do not require them. The parameters that apply to WIN$PRINTER op-codes are WINPRINT-DATA, WINPRINT-SELECTION, WINPRINT-COLUMN, WINPRINT-MEDIA, WINPRINT-JOB-STATUS and data defined in WORKING-STORAGE by the user. These are all defined in “winprint.def”. The parameters correspond to each of the op-codes as follows:

OP-CODE	Parameter
WINPRINT-SUPPORTED	none
WINPRINT-SETUP	none

OP-CODE	Parameter
WINPRINT-SETUP-USE-MARGINS	none
WINPRINT-GET-SETTINGS-SIZE	none
WINPRINT-GET-SPOOL-ERR	none
WINPRINT-SET-JOB	none
WINPRINT-UPDATE-PRINTERS	none
WINPRINT-GET-SETTINGS	user defined
WINPRINT-SET-SETTINGS	user defined
WINPRINT-SET-DATA-COLUMNS	WINPRINT-COLUMN
WINPRINT-CLEAR-DATA-COLUMNS	WINPRINT-COLUMN
WINPRINT-SET-PAGE-COLUMN	WINPRINT-COLUMN
WINPRINT-CLEAR-PAGE-COLUMNS	WINPRINT-COLUMN
WINPRINT-GET-PAGE-COLUMN	WINPRINT-COLUMN
WINPRINT-COLUMN-ALIGN-VERT	WINPRINT-COLUMN
WINPRINT-SET-STD-FONT	WINPRINT-DATA
WINPRINT-GET-PAGE-LAYOUT	WINPRINT-DATA
WINPRINT-SET-FONT	WINPRINT-DATA
WINPRINT-SET-LINES-PER-PAGE	WINPRINT-DATA
WINPRINT-GET-CAPABILITIES	WINPRINT-DATA
WINPRINT-PRINT-BITMAP	WINPRINT-DATA
WINPRINT-SET-MARGINS	WINPRINT-DATA
WINPRINT-GET-MARGINS	WINPRINT-DATA
WINPRINT-GRAPH-BRUSH	WINPRINT-DATA
WINPRINT-GRAPH-PEN	WINPRINT-DATA
WINPRINT-GRAPH-DRAW	WINPRINT-DATA
WINPRINT-SET-CURSOR	WINPRINT-DATA
WINPRINT-SET-TEXT-COLOR	WINPRINT-DATA

OP-CODE	Parameter
WINPRINT-SET-BKMODE	WINPRINT-DATA
WINPRINT-GET-JOB-STATUS	WINPRINT-JOB-STATUS
WINPRINT-SET-JOB-STATUS	WINPRINT-JOB-STATUS
WINPRINT-GET-PRINTER-MEDIA	WINPRINT-MEDIA
WINPRINT-GET-NO-PRINTERS	WINPRINT-SELECTION
WINPRINT-GET-PRINTER-INFO	WINPRINT-SELECTION
WINPRINT-SET-PRINTER	WINPRINT-SELECTION
WINPRINT-GET-CURRENT-INFO	WINPRINT-SELECTION
WINPRINT-GET-CURRENT-INFO-EX	WINPRINT-SELECTION
WINPRINT-SET-PRINTER-EX	WINPRINT-SELECTION
WINPRINT-GET-CURRENT-INFO-EX	WINPRINT-SELECTION
WINPRINT-GET-PRINTER-STATUS	WINPRINT-SELECTION

Description

To use this library routine you must include the COPY file “winprint.def”. You will need to copy this file into the Working-Storage section of any program that calls WIN$PRINTER. You must also assign the print file to either “-P SPOOLER” or to “-Q <printername>”, as described in your Getting Started booklet.

WIN$PRINTER takes one or more parameters. The first parameter is a mandatory operation code that indicates which sub-function of WIN$PRINTER to perform. The operation codes are described under WIN$PRINTER op-codes, below.

The other parameters are optional data items defined in “winprint.def” or defined by the user in Working-Storage. You use these data items to pass data to and from WIN$PRINTER. The specific data passed depends on the particular operation being called. Some operations do not use a data item, in which case it can be omitted.

The definitions of WINPRINT-DATA, WINPRINT-SELECTION, WINPRINT-COLUMN, WINPRINT-MEDIA, and WINPRINT-JOB-STATUS are included in “winprint.def”. These definitions may change in future versions as capabilities are added to WIN$PRINTER. However, ACUCOBOL-GT will continue to support the existing formats.

When WIN$PRINTER is called, it sets a return value to indicate whether the call succeeded or failed. A positive value indicates success. A zero or a negative value indicates an error. The error codes are defined in the section on error handling, below.

If the call to WIN$PRINTER does not include a GIVING item for the return value, the return value is placed in the special register RETURN-CODE.

See “graphprn.cbl” and “prndemox.cbl” for examples of many of the WIN$PRINTER functions.

Tip: Several operations accept parameters that have values measured in the dots-per-inch (DPI) resolution of the output device. Using the Windows *graphical device interface* (GDI), a program can get DPI information for a given printer on the system. A demonstration program written in COBOL is available in the Support area of the Micro Focus Web site. To download the program, go to: **http://supportline.microfocus.com/xmlloader.asp?type=home**. Select **Samples and Utilities** > **Acucorp Examples** > **Graphical User Interface Sample Programs** > **GetPrinterResolution.cbl**.

Error Handling

When you call WIN$PRINTER, it returns a status value. This numeric value is returned in the CALL statement’s GIVING data item, or the special register RETURN-CODE if no GIVING item is specified. A positive value indicates

that the routine succeeded. A value of zero or less indicates that an error or exception occurred. These situations have level 78 values defined for them in “winprint.def”. The defined values include:

WPRTERR-BAD-ARG -- This code indicates an unknown operation code or illegal value for any of the WIN$PRINTER functions.

WPRTERR-BAD-DRIVER -- This code is returned when the spooler can’t find a device driver that corresponds to the selected printer.

WPRTERR-BUFFER-TOO-SMALL -- This code is returned when the data item passed to the WINPRINT-GET-SETTINGS operation is too small to hold the spooler’s current configuration.

WPRTERR-CANCELLED -- This code is returned when you use the WINPRINT-SETUP or WINPRINT-SETUP-USE-MARGINS operation to display the printer setup dialog box and the user presses the “Cancel” button or closes the dialog box without pressing the “OK” button. This status can usually be ignored because the runtime automatically restores the prior configuration.

WPRTERR-DEVICE-INCAPABLE -- This code is returned when you try to print a bitmap and the printer you are using cannot print bitmaps.

WPRTERR-DRV-LOADFAIL -- This code is returned when WIN$PRINTER failed to load the driver information for the chosen printer. This could be caused by a corrupted file, bad registry settings, or a remote printer being offline.

WPRTERR-ENUM-FAIL -- This code is returned when one of the WIN$PRINTER functions does not find any available printers on the system.

WPRTERR-NO-MEMORY -- This code indicates that the system ran out of memory when trying to perform the requested operation.

WPRTERR-SPOOLER-CLOSED -- This code is returned when you attempt to print a bitmap on a closed print file.

WPRTERR-SPOOLER-OPEN -- This code indicates that the program tried to change the spooler’s configuration while a spooled print file was open.

WPRTERR-SPOOL-ERR -- This code is returned when there is an error in the Graphical Device Interface (GDI) layer that is not listed in "winprint.def". Use the operation WINPRINT-GET-SPOOL-ERR to obtain the exact Windows API code and refer to your Windows SDK documentation for a description.

WPRTERR-UNSUPPORTED -- This code is returned whenever WIN$PRINTER is called on a machine that is not a Windows machine.

WIN$PRINTER op-codes

The following is a list of WIN$PRINTER operation codes and their effects. These level 78 items are all defined in "winprint.def".

WINPRINT-DATA op-codes
WINPRINT-SELECTION op-codes
WINPRINT-COLUMN op-codes
WINPRINT-JOB-STATUS op-codes
WINPRINT-MEDIA op-codes
USER-DATA op-codes

Printer Information op-codes

The following operation codes provide general information about page layout, the print spooler buffer, and whether or not the WIN$PRINTER routine is supported.

WINPRINT-GET-SETTINGS-SIZE
WINPRINT-SETUP
WINPRINT-SETUP-USE-MARGINS
WINPRINT-SUPPORTED
WINPRINT-GET-SPOOL-ERR
WINPRINT-SET-JOB
WINPRINT-UPDATE-PRINTERS

WINPRINT-GET-SETTINGS-SIZE

This operation code retrieves the size of the buffer available in the current spooler configuration.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-SETTINGS-SIZE
    GIVING RESULT
```

Return Values

This operation returns the number of bytes needed to hold the current spooler configuration.

Description

The spooler configuration includes the destination device, paper size, and page orientation. It does not include the current font selection. Use this operation to ensure that you have a large enough buffer when using the WINPRINT-GET-SETTINGS and WINPRINT-SET-SETTINGS operations.

Note: This operation is not supported in our Thin Client environment.

WINPRINT-SETUP

This operation code calls the standard Windows Setup Printer dialog box. This allows the user to select which printer to use, the desired page orientation (landscape or portrait), and the desired paper size and source. To also get the page margins, use WINPRINT-SETUP-USE-MARGINS instead. Note that there is also a **WINPRINT-SETUP-EX** operation code for calling the PrintDlgEx function, which is considered a more modern and feature-rich printer dialog box and is fully supportd in Windows Vista.

In Windows Vista, Microsoft no longer fully supports the PrintDlg function, which leaves the function relatively useless. This in turn limits the functionality of WINPRINT-SETUP, which is used to call PrintDlg.

To minimize the impact on users of WINPRINT-SETUP, the runtime detects when it is executed on Vista and will change to use the new **WINPRINT-SETUP-EX** operation code that is used to display the PrintDlgEx printer window.

Note that the application must have a window open in order for this feature to work. If no window is open, the printer dialog will not be shown, the error WPRTERR-WINDOW-REQUIRED will be returned, and the print job may go to the default printer.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SETUP
    GIVING RESULT
```

Description

The runtime internally configures itself based on the selections chosen by the user. These become the settings used during the remainder of the run or until the next call to this operation. Settings return to their defaults when the runtime exits.

If you are using the WINPRINT-SELECTION data structure, calls to WINPRINT-SETUP must also be followed by a call to WINPRINT-GET-CURRENT-INFO(-EX) to ensure consistency between COBOL data storage information and the current Windows configuration. However, if you do not use any of the operation codes that rely on the WINPRINT-SELECTION group, there is no need to call WINPRINT-GET-CURRENT-INFO(-EX).

Note: Changing the output device with this operation will reset any columns you have set using WINPRINT-COLUMN op-codes.

WINPRINT-SETUP-USE-MARGINS

This operation code calls the standard Windows Setup Printer dialog box and utilizes the margins specified in the dialog box. This allows the user to select which printer to use, the desired page orientation, the margins, and the desired paper size and source.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SETUP-USE-MARGINS
    GIVING RESULT
```

Description

The WINPRINT-SETUP-USE-MARGINS operation code is similar to WINPRINT-SETUP in its base usage.

If you are using the WINPRINT-SELECTION data structure (like WINPRINT-SETUP), it must be followed by a call to WINPRINT-GET-CURRENT-INFO(-EX) to ensure consistency between COBOL data storage information and the current Windows configuration. However, if you do not use any of the operation codes that rely on the WINPRINT-SELECTION group, there is no need to call WINPRINT-GET-CURRENT-INFO(-EX).

WINPRINT-SETUP-USE-MARGINS is defined in "winprint.def". You can use this operation code with WINPRINT-GET-MARGINS to obtain the current margin.

Note: Changing the output device with this operation will reset any columns you have set using WINPRINT-COLUMN op-codes.

WINPRINT-SUPPORTED

This operation code determines if the WIN$PRINTER routine is supported (i.e., the host machine is a Windows machine).

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SUPPORTED
    GIVING RESULT
```

Return Values

This operation sets the return value of WIN$PRINTER to "1" if the WIN$PRINTER routine is supported. Otherwise it sets the return value to WPRTERR-UNSUPPORTED.

Comments

The WIN$PRINTER routine can only be used with Microsoft Windows.

WINPRINT-GET-SPOOL-ERR

This operation code obtains the number of the most recent Windows Graphical Device Interface (GDI) error from the Windows API.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-SPOOL-ERR
    GIVING RESULT
```

Return Values

This operation returns the most recent Windows GDI error.

Description

The complexity of Windows graphics capabilities make it impossible to catalog all possible error conditions in “winprint.def”. Therefore, if a graphical error condition occurs, the WIN$PRINTER op-code that caused the error returns WPRTERR-SPOOL-ERR to indicate that a Windows API error occurred. Use WINPRINT-GET-SPOOL-ERR to retrieve the specific Windows API error number, then refer to Windows API documentation for a description of the problem.

This operation should be called immediately after a WIN$PRINTER operation call has returned WPRTERR-SPOOL-ERR. Calling this function under other circumstances will return unpredictable values. This operation does not impact WRITE statements.

Note: Calling WINPRINT-GET-SPOOL-ERR does not reset the last GDI error internally. Do not use this operation as a method to verify that a call worked properly.

WINPRINT-SET-JOB

This operation code returns the identifier of the job that is currently spooling into the printer. The ID number returned by this operation will be compatible with, and may be used in conjunction with, the operations WINPRINT-SET-JOB-STATUS and WINPRINT-GET-JOB-STATUS.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-JOB JOB-ID
    GIVING PRINT-JOB.
```

Return Values

If you set JOB-ID to “0”, WINPRINT-SET-JOB returns the identifier of the job that is currently spooling into the printer (PRINT-JOB). You can then use that number to tell WIN$PRINTER operations which print job is the target. This is your only way to obtain the ID of a job. To restore the system default settings, simply call this operation with a JOB-ID of “-1”.

Description

This call should be issued immediately after the opening of a job. For example:

```
OPEN OUTPUT FIRST-FILE.
CALL "WIN$PRINTER" USING WINPRINT-SET-JOB JOB-ID GIVING
FIRST-ID.
```

where FIRST-ID is a variable declared signed-integer, such as:

```
77 FIRST-ID USAGE SIGNED-INT.
```

Subsequent calls to WIN$PRINTER may use FIRST-ID to identify the target for the next action, as follows:

```
      OPEN    OUTPUT           FIRST-FILE.
      CALL    "WIN$PRINTER"    USING
              WINPRINT-SET-JOB 0
              GIVING           FIRST-ID.
      OPEN    OUTPUT           SECOND-FILE. | Is now current.
*Initialize the print record for the first print job.
      MOVE    "This is job 1, printed with MS Sans Serif." TO
              RECORD-FILE-1.
*Initialize the print record for the second print job.
      MOVE    "This is job 2, printed with Script." TO
              RECORD-FILE-2.
*Set active job to the first print job.
      CALL    "WIN$PRINTER"    USING
              WINPRINT-SET-JOB FIRST-ID.
*Set the preferred font for the first print job.
```

```
INITIALIZE                WINPRINT-DATA.
MOVE    FIRST-FONT        TO WPRTDATA-FONT.
CALL    "WIN$PRINTER"     USING
        WINPRINT-SET-FONT
        WINPRINT-DATA.
```

If you try to perform an operation that requires an active print job and there is none, an error status is returned. This series of calls can be used with all WIN$PRINTER functions, with the following exceptions:

- The status of a printer cannot be determined using WINPRINT-GET-PRINTER-STATUS or WINPRINT-GET-JOB-STATUS unless the print job has already started. This is because the port monitor must both detect a print error and report it to the printer queue before it can be recognized by WIN$PRINTER functions.

- Due to a limitation in the Microsoft Windows API, computers that run Windows 9x (Windows 98, and Windows ME) do not return the spooler job ID when opening a print job. This means that you cannot use the WINPRINT-GET-JOB-STATUS and WINPRINT-SET-JOB-STATUS operations of the WIN$PRINTER library routine on these machines. (These operations are used to check and modify the status of a particular printer.)

When you are printing multiple jobs simultaneously, you should not set a printer font before the print job has been opened because the font could be applied to the wrong job. Once the print job is opened, you may set the font, using the JOB-ID of the target printer.

However, if you need to change the printer settings for subsequent job on a different printer, you should set JOB-ID to “-1” before setting WINPRINT-SET-SETTINGS or WINPRINT-SET-PRINTER(-EX). This causes WINPRINT-SET-JOB to return the ID number of the next job in the queue (after the current job). This should be done just prior to calling an OPEN statement. When JOB-ID is set to “-1”, the runtime executes the next WIN$PRINTER operation as if no current job were printing. This does not affect existing jobs, but it affects the status of subsequent jobs, unless it is an OPEN, WRITE, or CLOSE statement.

WINPRINT-UPDATE-PRINTERS

Usage

```
CALL "WIN$PRINTER"
     USING WINPRINT-UPDATE-PRINTERS
```

Description

This op-code enables you to force the runtime to reload the internal printerlist so that any changes to that list (new printer added, for example) will be detected by the COBOL program.

This op-code takes no parameters and always returns “TRUE”. By calling this op-code, the runtime printing system is told to reload the printer list. A call to this op-code should be followed by a call to one of the following op-codes to ensure synchronization between the COBOL printer list and the internal runtime:

```
78  WINPRINT-GET-NO-PRINTERS            VALUE 13.
78  WINPRINT-GET-PRINTER-INFO           VALUE 14.
78  WINPRINT-GET-CURRENT-INFO           VALUE 16.
78  WINPRINT-GET-PRINTER-INFO-EX        VALUE 28.
78  WINPRINT-GET-CURRENT-INFO-EX        VALUE 30.
```

WINPRINT-DATA op-codes

The following operation codes use the data item WINPRINT-DATA (defined in “winprint.def”). These operations are used to specify the appearance of a printed page, including features such as font, page layout and graphics placement.

WINPRINT-GET-CAPABILITIES
WINPRINT-GET-MARGINS
WINPRINT-GET-PAGE-LAYOUT
WINPRINT-GRAPH-DRAW
WINPRINT-GRAPH-BRUSH
WINPRINT-GRAPH-PEN
WINPRINT-PRINT-BITMAP
WINPRINT-SET-CURSOR
WINPRINT-SET-TEXT-COLOR
WINPRINT-SET-FONT

WINPRINT-SET-LINES-PER-PAGE
WINPRINT-SET-MARGINS
WINPRINT-SET-STD-FONT
WINPRINT-SET-BKMODE

WINPRINT-GET-CAPABILITIES

This operation code determines whether or not a printer can print bitmaps.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-CAPABILITIES, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-CAPABILITIES REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-BITMAPS-OK-FLAG      PIC 9.
         88 WPRTDATA-BITMAPS-OK        VALUE 1, FALSE ZERO.
```

Return Values

This operation returns information about the capabilities of the currently selected printer to WPRTDATA-CAPABILITIES. Currently, the only capability returned to this parameter is whether or not the printer can print bitmaps. If it can, WPRTDATA-BITMAPS-OK is true. Printers that do not use a raster technology (such as a pen plotter) cannot print bitmaps.

WINPRINT-GET-MARGINS

This operation code may be used with WINPRINT-SETUP-USE-MARGINS to obtain the margins set in the Windows printer setup dialog box or the margins set with WINPRINT-SET-MARGINS.

If used with WINPRINT-SETUP-USE-MARGINS, it returns information about the current default margin in the Windows printer setup dialog box using centimeters or inches but not with pixels or cells. This occurs because the operation code is used with WINPRINT-SETUP-USE-MARGINS and can therefore be used only with values supported by the dialog box.

If you have used WINPRINT-SET-MARGINS and later call WINPRINT-GET-MARGINS, the values returned are those originally set with WINRPRINT-SET-MARGINS, which may include inches, centimeters, pixels, or cells.

Usage

```
INITIALIZE WPRTDATA-MARGINS
CALL "WIN$PRINTER"
    USING WINPRINT-GET-MARGINS, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in “winprint.def” as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-MARGINS REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-TOP-MARGIN          PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BOTTOM-MARGIN       PIC 9(7)V99 COMP-5.
      05 WPRTDATA-LEFT-MARGIN         PIC 9(7)V99 COMP-5.
      05 WPRTDATA-RIGHT-MARGIN        PIC 9(7)V99 COMP-5.
      05 WPRTDATA-MARGIN-UNITS        UNSIGNED-SHORT.
```

WINPRINT-GET-MARGINS requires the WPRTDATA-MARGINS structure to be passed as the second parameter and WPRTDATA-MARGINS is filled with the current margins defined as the defaults in the printer dialog box.

Return Values

This operation returns one of the following values:

WPRTMARGIN-CENTIMETERS
WPRTMARGIN-INCHES

WPRTMARGIN-PIXELS (only if set with WINPRINT-SET-MARGINS)
WPRTMARGIN-CELLS (only if set with WINPRINT-SET-MARGINS)

WINPRINT-GET-PAGE-LAYOUT

This operation code determines how many columns and rows of characters will fit on a page.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PAGE-LAYOUT, WINPRINT-DATA,
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRT-PAGE-LAYOUT REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-LINES-PER-PAGE       UNSIGNED-SHORT.
      05 WPRTDATA-COLUMNS-PER-PAGE     UNSIGNED-SHORT.
```

Return Values

This operation returns the number of print rows that can fit on a page in WPRTDATA-LINES-PER-PAGE, and the number of print columns in WPRTDATA-COLUMNS-PER-PAGE. This accounts for the current page size, orientation, and printer font. You may use this routine either before or after opening a print file.

Because of the large number of variables involved with printing under Windows (for example, the font size, the paper size, and print orientation), we recommend that you use this routine to determine how many lines will fit on a page when you are formatting reports.

However, we do not recommend using this operation when printing in DIRECT mode using "-P SPOOLER" or "-Q <*printername*>". When you print in DIRECT mode, the Windows print spooler has no control of the printer, and no initialization of the printer is performed by the Windows

printer driver. This means that the print job uses the hardware defaults. For example, if you print in DIRECT mode to a printer with the hardware default paper size set to US letter format, that is the format used, even if the driver has A4 paper set as the default, in which case this operation is likely to return incorrect values.

WINPRINT-GRAPH-DRAW

This operation code specifies the size, shape and location of a graphic in the current print job using pen and brush attributes specified by **WINPRINT-GRAPH-BRUSH** and **WINPRINT-GRAPH-PEN**.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GRAPH-DRAW, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in “winprint.def” as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03  WPRTDATA-DRAW REDEFINES
   WPRTDATA-SET-STD-FONT.
      05  WPRTDATA-DRAW-START-X       PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-START-Y       PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-STOP-X        PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-STOP-Y        PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-UNITS         UNSIGNED-SHORT.
      05  WPRTDATA-DRAW-SHAPE         UNSIGNED-SHORT.
```

Return Value

This operation returns the coordinates and shape of a graphic.

Description

The printer must be open to perform this operation. WPRTDATA-DRAW should be initialized prior to use. There is no limit to the number of times this operation may be called. Once it is called, all further printing using the WRITE statement is performed using a TRANSPARENT background. The

TRANSPARENT setting is not usually applied to WRITE statements, but this mode is necessary for graphics to print correctly. (This does not apply to WINPRINT-COLUMN operations, which have a particular setting for TRANSPARENCY.)

Note: The current cursor position on the printer is not modified by this call.

If WINPRINT-GRAPH-PEN or WINPRINT-GRAPH-BRUSH have not yet been called, this operation will create and use the defaults. The default pen is solid black, with a width of “1”. The default brush is NULL, meaning there is no fill.

Note: When you call WINPRINT-GRAPH-DRAW, the operation will automatically test to see if a form feed is pending. If this is the case, the form feed will be performed before the call to this operation is executed.

WINPRINT-GRAPH-DRAW has the following values:

WPRTDATA-DRAW-UNITS -- Specifies the unit of measure of the values passed for drawing coordinates. If an illegal value is set, the unit of measure will be set to the default (WPRTUNITS-PIXELS). The unit of measure may be set to one of the following values:

WPRTUNITS-CELLS	Values are measured using the “cell size” of the currently selected font. The cell-size is determined by the height and width of the “0” character of a font. This is roughly equivalent to measuring in “characters”. If you use a proportional font, it is common for uppercase characters to be wider than this measurement. Non-integer values are allowed in the measurements.

WPRTUNITS-CELLS-ABS	Values are measured using the "cell size" of the currently selected font. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error. (Note that due to inherent differences in the hardware of printer manufacturers, this value may not provide truly device-independent results.)
WPRTUNITS-INCHES	Values are measured in inches.
WPRTUNITS-INCHES-ABS	Values are measured in inches. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error.

WPRTUNITS-CENTIMETERS	Values are measured in centimeters.
WPRTUNITS-CENTIMETERS-ABS	Values are measured in centimeters. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error.
WPRTUNITS-PIXELS (default)	Values are measured using the dots-per-inch (DPI) resolution of the output device. Only integer values are allowed in the measurements. The actual size of this measurement varies depending on the target printer's resolution. This means that a width of "5" will appear differently on a 300dpi printer than it will on a 600dpi printer. Consider the unit of measure relative to the resolution of the targeting printer before printing.

WPRTDATA-DRAW-SHAPE -- Specifies which type of shape to draw. If an illegal value is used, no shape will be drawn. The possible values are:

WPRT-DRAW-RECTANGLE	Draws a rectangle with 90-degree corners.
WPRT-DRAW-ROUND-RECTANGLE	Draws a rectangle with rounded corners.
WPRT-DRAW-LINE	Draws a line.

WPRTDATA-DRAW-START-X -- Specifies the top-left horizontal coordinate of the shape to draw. The unit of measure is set with WPRTDATA-DRAW-UNITS. The minimum value of this coordinate is “0”. The top-leftmost coordinate for all graphic operations is “0,0”

WPRTDATA-DRAW-START-Y -- Specifies the top-left vertical coordinate of the shape to draw. The unit of measure is set with WPRTDATA-DRAW-UNITS. The minimum value of this coordinate is “0”. The top-leftmost coordinate for all graphic operations is “0,0”

WPRTDATA-DRAW-STOP-X -- Specifies the lower-right horizontal coordinate of the shape to draw. The unit of measure is set with WPRTDATA-DRAW-UNITS. The maximum value of this coordinate depends on the unit of measure selected.

WPRTDATA-DRAW-STOP-Y -- Specifies the lower-right vertical coordinate of the shape to draw. The unit of measure is set with WPRTDATA-DRAW-UNITS. The maximum value of this coordinate depends on the unit of measure selected.

Example

See “graphprn.cbl” for examples of printing graphics. This example will draw a box with rounded edges:

```
INITIALIZE                WPRTDATA-DRAW.
MOVE    WPRT-DRAW-ROUND-RECTANGLE TO  WPRTDATA-DRAW-SHAPE.
MOVE    3                 TO  WPRTDATA-DRAW-START-X.
MOVE    1                 TO  WPRTDATA-DRAW-START-Y.
MOVE    40                TO  WPRTDATA-DRAW-STOP-X.
MOVE    10                TO  WPRTDATA-DRAW-STOP-Y.
MOVE    WPRTUNITS-CELLS   TO  WPRTDATA-DRAW-UNITS.
```

```
CALL    "WIN$PRINTER"
        USING     WINPRINT-GRAPH-DRAW, WINPRINT-DATA
        GIVING    CALL-RESULT.
```

WINPRINT-GRAPH-BRUSH

This operation code specifies the pattern or solid color used to fill a shape drawn with WINPRINT-GRAPH-DRAW.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GRAPH-BRUSH, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in “winprint.def” as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03  WPRTDATA-BRUSH REDEFINES
   WPRTDATA-SET-STD-FONT.
      05  WPRTDATA-BRUSH-STYLE        UNSIGNED-SHORT.
      05  WPRTDATA-BRUSH-COLOR        PIC 9(9) COMP-5
```

Return Values

This operation returns the style and color of the brush used by WINPRINT-GRAPH-DRAW

Description

The printer must be open to perform this operation. It must be called prior to WINPRINT-GRAPH-DRAW. Once executed, the brush specified will apply to all graphic operations until a new call is executed. The selected brush is released when the printer is closed and will not affect subsequent print jobs. WPRTDATA-BRUSH must be initialized before use. There is no limit to the number of times this operation may be called.

WINPRINT-GRAPH-BRUSH has the following values:

WPRTDATA-BRUSH-COLOR -- Specifies the color used to fill the shape drawn with WINPRINT-GRAPH-DRAW. The color resolution (COLORREF) is a combination of Red, Green, and Blue (RGB). The intensity of each color in the mix is determined by a number between "0" and "255". The default color (0,0,0) is black.

Note: See your Windows API documentation for more information about RGB colors and COLORREF values.

WPRTDATA-BRUSH-STYLE -- Specifies the pattern used to fill the shape drawn with WINPRINT-GRAPH-DRAW. It may be set to one of the following values:

WPRT-BRUSH-SOLID	Selects a solid brush that fills the shape with the color selected in WPRTDATA-BRUSH-COLOR .
WPRT-BRUSH-NULL (default)	The shape is not filled, and the background layer will show through.
WPRT-BRUSH-BDIAGONAL	Fills the shape with a pattern of lines angled at 45-degrees. (`///` `///`)
WPRT-BRUSH-CROSS	Fills the shape with a pattern of crosses. (`++++++`)
WPRT-BRUSH-DIAGCROSS	Fills the shape with a pattern of diagonal crosses. (`xxxxxx`)
WPRT-BRUSH-FDIAGONAL	Fills the shape with a pattern of lines angled at –45-degrees. (`\\\\\\`)
WPRT-BRUSH-HORIZONTAL	Fills the shape with a pattern of dashes. (`------`)
WPRT-BRUSH-VERTICAL	Fills the shape with a pattern of vertical bars. (`||||||`)

WPRT-BRUSH-DKGRAY	Fills the shape with solid dark gray. If color is specified, it is ignored.
WPRT-BRUSH-GRAY	Fills the shape with solid gray. If color is specified, it is ignored.
WPRT-BRUSH-LTGRAY	Fills the shape with solid light gray. If color is specified, it is ignored.

Example

See “graphprn.cbl” for examples of printing graphics. This example will fill the shapes drawn with solid gray:

```
INITIALIZE WPRTDATA-BRUSH.
MOVE WPRT-BRUSH-GRAY  TO  WPRTDATA-BRUSH-STYLE.

CALL "WIN$PRINTER"
    USING WINPRINT-GRAPH-BRUSH, WINPRINT-DATA
    GIVING CALL-RESULT.
```

WINPRINT-GRAPH-PEN

This operation code specifies characteristics of the “pen” used to draw a shape with WINPRINT-GRAPH-DRAW.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GRAPH-PEN, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in “winprint.def” as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03  WPRTDATA-PEN REDEFINES
```

```
WPRTDATA-SET-STD-FONT.
   05  WPRTDATA-PEN-STYLE          UNSIGNED-SHORT.
   05  WPRTDATA-PEN-WIDTH          UNSIGNED-SHORT.
   05  WPRTDATA-PEN-COLOR          PIC 9(9) COMP-5.
```

Return Values

This operation returns the characteristics of lines drawn with WINPRINT-GRAPH-DRAW.

Description

The printer must be open to perform this operation. It must be called prior to WINPRINT-GRAPH-DRAW. Once executed, the pen specified will apply to all graphic operations until a new call is executed. The selected pen is released when the printer is closed and will not affect subsequent print jobs. WPRTDATA-PEN must be initialized before use. There is no limit to the number of times this operation may be called.

WINPRINT-GRAPH-PEN has the following values:

WPRTDATA-PEN-WIDTH -- Specifies the thickness of the line drawn using WINPRINT-GRAPH-DRAW. The default unit of measure is pixels. For example, if WPRTDATA-PEN-WIDTH is set to “0”, the width will be one pixel, unless another unit of measure has been selected. The default value is “1”.

The actual size of this measurement is affected by the target printer’s resolution. This means that a width of “5” will appear differently on a printer with a resolution of 300 dots-per-inch (DPI) than it will on a printer with 600dpi. Consider the unit of measure relative to the resolution of the targeting printer before printing.

WPRTDATA-PEN-COLOR -- Specifies the color of a line drawn using WINPRINT-GRAPH-DRAW. The color resolution (COLORREF) is a combination of Red, Green, and Blue (RGB). The intensity of each color in the mix is determined by a number between “0” and “255”. The default color (0,0,0) is black.

See your Windows API documentation for more information about RGB colors and COLORREF values.

WPRTDATA-PEN-STYLE -- Specifies the type of line drawn. It may be set to one of the following values:

WPRT-PEN-SOLID (default)	Draws a solid line in the color selected in WPRTDATA-BRUSH-COLOR.
WPRT-PEN-NULL	Draws an "invisible" line. The background layer will show through.
WPRT-PEN-DASH	Draws a line of dashes (- - - - - -). WPRTDATA-PEN-WIDTH must equal "1" to use this style.
WPRT-PEN-DOT	Draws a line of dots (. . . . . .). WPRTDATA-PEN-WIDTH must equal "1" to use this style.
WPRT-PEN-DASHDOT	Draws a line of alternating dashes and dots (- . - . - . - . - .). WPRTDATA-PEN-WIDTH must equal "1" to use this style.
WPRT-PEN-DASHDOTDOT	Draws a line of dashes followed by two dots, (- . . - . . - . . - . . - . . - . .). WPRTDATA-PEN-WIDTH must equal "1" to use this style.
WPRT-PEN-INSIDEFRAME	Draws a solid line inside of the actual geometric area of the shape.

Example

See "graphprn.cbl" for examples of printing graphics. This example will draw lines and shapes with a solid pen that is 10 pixels thick:

```
INITIALIZE WPRTDATA-PEN.
MOVE WPRT-PEN-SOLID  TO  WPRTDATA-PEN-STYLE.
MOVE 10  TO  WPRTDATA-PEN-WIDTH.

CALL "WIN$PRINTER"
    USING  WINPRINT-GRAPH-PEN, WINPRINT-DATA
    GIVING  CALL-RESULT.
```

WINPRINT-PRINT-BITMAP

This operation code prints a bitmap in the current report.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-PRINT-BITMAP, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-PRINT-BITMAP REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-BITMAP               PIC X(4) COMP-N.
      05 WPRTDATA-BITMAP-ROW           PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BITMAP-COL           PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BITMAP-HEIGHT        PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BITMAP-WIDTH         PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BITMAP-FLAGS         UNSIGNED-SHORT.
```

Description

The print file must be open when you are using this function. The bitmap is printed according to the data contained in WPRTDATA-PRINT-BITMAP. To ensure that elements are initialized to their default values, INITIALIZE WPRTDATA-PRINT-BITMAP before filling in the elements.

Note: When you call WINPRINT-PRINT-BITMAP, the operation will automatically test to see if a form feed is pending. If this is the case, the form feed will be performed before the call to this operation is executed.

WPRTDATA-BITMAP should contain the handle of the bitmap you want to print. You can obtain this handle by calling the library routine W$BITMAP with the WBITMAP-LOAD option. This handle can be the same as the handle of the bitmap you have displayed on the screen.

The dimensions of the bitmap are specified by **WPRTDATA-BITMAP-HEIGHT** and **WPRTDATA-BITMAP-WIDTH**. The unit of measurement by which the size of the bitmap is calculated is set with **WPRTDATA-BITMAP-FLAGS**. One of the following values is used:

WPRTBITMAP-SCALE-CELLS	the height and width of a cell depends on the number of rows and columns in the report. The currently selected font for the printer determines the number of rows and columns on a page. The top left corner of a report is row 1, column 1. You may use fractional rows and columns, but if you specify a row or column less than 1, then the bitmap is placed at row 1, column 1.
WPRTBITMAP-SCALE-INCHES	the units represent inches on the printed page.
WPRTBITMAP-SCALE-CENTIMETERS	the units represent centimeters on the printed page.
WPRTBITMAP-SCALE-PIXELS	the units are based on the resolution of the output device. This is measured in dots-per-inch (DPI). Fractional values are ignored.

The location of the top left corner of a bitmap is specified by **WPRTDATA-BITMAP-ROW** and **WPRTDATA-BITMAP-COL**. By default, this coordinate is specified in cells. You may choose to use another unit of measurement by setting **WPRTDATA-BITMAP-FLAGS** to one of the following values (defined in “winprint.def”):

WPRTBITMAP-UNITS-INCHES	Values are measured in inches.
WPRTBITMAP-UNITS-CENTIMETERS	Values are measured in centimeters.

WPRTBITMAP-UNITS-PIXELS	Values are measured using the dots-per-inch (DPI) resolution of the output device. Only integer values are allowed. The actual size of this measurement varies depending on the target printer's resolution. Consider the unit of measure relative to the resolution of the targeting printer before printing.
WPRTBITMAP-UNITS-CELLS-ABS	Values are measured in cells, and the position of the bitmap is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller).
WPRTBITMAP-UNITS-INCHES-ABS	Values are measured in inches and the position of the bitmap is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller).
WPRTBITMAP-UNITS-CENTIMETERS-ABS	Values are measured in centimeters and the position of the bitmap is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller).

This is illustrated in Example 2, below.

Many printers have much higher resolution than screens do. For example, many laser printers can print 300 or 600 dots per inch while most screens display less than 100 pixels per inch. An image that is 1024 pixels wide

would fill or overflow many screens, but would be less than 2 inches wide on a 600 DPI printer. For this reason, bitmaps are usually scaled when they are printed. By default, the runtime scales the image so that the relative proportions of the printed image match those of the same image when it is viewed on the screen.

Scaling a bitmap

To scale a bitmap to a particular size, you must set WPRTDATA-BITMAP-FLAGS to the desired unit of measure (cells, inches, centimeters, or pixels). Then set the desired dimensions of the bitmap in WPRTDATA-BITMAP-WIDTH and WPRTDATA-BITMAP-HEIGHT.

You can either set both dimensions or leave one dimension at zero. When one of the dimensions is set to zero, the relative proportions of the image are unchanged after the scaling of the other dimension is complete.

You can inhibit the scaling done by the runtime by setting WPRTDATA-BITMAP-FLAGS to WPRTBITMAP-PRINTER-BITMAP. This informs the runtime that the bitmap was designed directly for printing on the current printer and should not be scaled. You can also add the value of WPRTDATA-BITMAP-FLAGS to the other scaling options discussed above to prevent the runtime from performing an adjustment to the scaling. Adjustments are usually done to account for the difference in the relative proportions of the screen's X and Y density in comparison to the printer's X and Y density. Some devices have a much higher resolution in one dimension than the other. This adjustment handles the changes needed when you are viewing a screen image on a printer. Most applications, however, should avoid this option because most bitmaps are meant to be displayed on the screen only.

Colors in the bitmap image are preserved by the runtime. It is up to the printer's driver to decide how to print color images on a black-and-white device. Most drivers turn colors into varying shades of gray.

Example 1

The following sample code prints the AcuBench logo in the center of an 80-character print line. It scales the image to be 30 characters wide to simplify the centering computation. This example assumes that the printer is already open:

```
77  LOGO-HANDLE        PIC S9(9) COMP-4.
        :
        :
CALL "W$BITMAP" USING WBITMAP-LOAD, "devsuite.bmp"
    GIVING LOGO-HANDLE
INITIALIZE WPRTDATA-PRINT-BITMAP
MOVE LOGO-HANDLE TO WPRTDATA-BITMAP
MOVE 1 TO WPRTDATA-BITMAP-ROW
MOVE 26 TO WPRTDATA-BITMAP-COL
MOVE 30 TO WPRTDATA-BITMAP-WIDTH

*Height left at zero

MOVE WPRTBITMAP-SCALE-CELLS
     TO WPRTDATA-BITMAP-FLAGS
CALL "WIN$PRINTER" USING WINPRINT-PRINT-BITMAP,
     WINPRINT-DATA
CALL "W$BITMAP" USING WBITMAP-DESTROY, LOGO-HANDLE
```

Example 2

The following example code scales a bitmap to be 3-by-3 inches square, and places the top left corner 10 centimeters away from both the left and top margins. This example assumes that the printer is already open:

```
77  LOGO-HANDLE        PIC S9(9) COMP-4.
        :
        :
CALL "W$BITMAP" USING WBITMAP-LOAD, "your_bitmap.bmp"
     GIVING LOGO-HANDLE
INITIALIZE WPRTDATA-PRINT-BITMAP
MOVE LOGO-HANDLE TO WPRTDATA-BITMAP
MOVE 10 TO WPRTDATA-BITMAP-ROW WPRTDATA-BITMAP-COL.
MOVE 3 TO WPRTDATA-BITMAP-HEIGHT WPRTDATA-BITMAP-WIDTH.
MOVE WPRTBITMAP-SCALE-INCHES TO WPRTDATA-BITMAP-FLAGS.
ADD  WPRTBITMAP-UNITS-CENTIMETERS TO WPRTDATA-BITMAP-FLAGS.

CALL "WIN$PRINTER" USING WINPRINT-PRINT-BITMAP,
```

```
     WINPRINT-DATA
CALL "W$BITMAP" USING WBITMAP-DESTROY, LOGO-HANDLE
```

WINPRINT-SET-CURSOR

This operation code allows you to change the position of the printer's write cursor. This is useful when doing multiple write statements that include a variety of fonts, font sizes and font attributes.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-CURSOR, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-DRAW REDEFINES
   WPRTDATA-SET-STD-FONT.
      05  WPRTDATA-DRAW-START-X       PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-START-Y       PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-STOP-X        PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-STOP-Y        PIC 9(7)V99 COMP-5.
      05  WPRTDATA-DRAW-UNITS         UNSIGNED-SHORT.
      05  WPRTDATA-DRAW-SHAPE         UNSIGNED-SHORT.
```

Return Values

This option returns the horizontal and vertical coordinates of the write cursor. If an error is returned, the current cursor position is not affected.

Description

When printing in Windows, the position of text on the printed page is determined by the location of the write cursor. This is usually handled automatically by the runtime, but you may use this operation to position the cursor yourself.

The printer must be open to perform this operation. WPRTDATA-DRAW should be initialized prior to use. The current cursor position on the printer is modified by this call only if WPRTDATA-DRAW-SHAPE is set to a value of “0” and the operation is successful. When the cursor is moved in this manner, subsequent WRITE statements will be affected. One exception is that the vertical position of subsequent calls to WINPRINT-COLUMNS will be altered, but not the horizontal position.

Note: When you call WINPRINT-SET-CURSOR, the operation will automatically test to see if a form feed is pending. If this is the case, the form feed will be performed before the call to this operation is executed.

If you use this operation with WPRDATA-DRAW-SHAPE set to a non-zero value, the cursor is not repositioned. This can be used to inquire the position of the write cursor without changing it. There is no limit to the number of times this operation may be called.

WINPRINT-SET-CURSOR has the following values:

WPRTDATA-DRAW-START-X -- Specifies the X coordinate of the cursor location. The unit of measure is set with WPRTDATA-DRAW-UNITS. The minimum value of this coordinate is “0”.

WPRTDATA-DRAW-START-Y -- Specifies the Y coordinate of the cursor location. The unit of measure is set with WPRTDATA-DRAW-UNITS. The minimum value of this coordinate is “0”.

WPRTDATA-DRAW-STOP-X -- Returns the lower-right horizontal coordinate of the cursor location. This parameter has no input value, the previous X coordinate is returned. The unit of measure is determined by the setting of WPRTDATA-DRAW-UNITS.

WPRTDATA-DRAW-STOP-Y -- Returns the lower-right vertical coordinate of the cursor location. This parameter has no input value, the previous Y coordinate is returned. The unit of measure is determined by the setting of WPRTDATA-DRAW-UNITS.

WPRTDATA-DRAW-UNITS -- Specifies the unit of measure used for the values passed. If an illegal value is used, the default will be used (WPRTUNITS-PIXELS). The unit of measure may be set to one of the following values:

WPRTUNITS-CELLS	Values are measured using the "cell size" of the currently selected font. The cell-size is determined by the height and width of the "0" character of a font. This is roughly equivalent to measuring in "characters". Positioning is relative to the individual printer's physical margin. Please note that the margin set in WINPRINT-SET-MARGINS is not used to determine the cursor position. If you use a proportional font, it is common for uppercase characters to be wider than this measurement. Non-integer values are allowed in the measurements.
WPRTUNITS-CELLS-ABS	Values are measured using the "cell size" of the currently selected font. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error. (Note that due to inherent differences in the hardware of printer manufacturers, this value may not provide truly device-independent results.)

WPRTUNITS-INCHES	Values are measured in inches. Positioning is relative to the individual printer's physical margin. Please note that the margin set in WINPRINT-SET-MARGINS is not used to determine the cursor position.
WPRTUNITS-INCHES-ABS	Values are measured in inches. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error.
WPRTUNITS-CENTIMETERS	Values are measured in centimeters. Positioning is relative to the individual printer's physical margin. Please note that the margin set in WINPRINT-SET-MARGINS is not used to determine the cursor position.
WPRTUNITS-CENTIMETERS-ABS	Values are measured in centimeters. Positioning is based on the left edge of the paper, regardless of the physical left margin determined by the printer (even if the absolute position is smaller). If the dimensions of the area to be printed are less than the printer's left or top physical margin, or greater than the printer's right or bottom physical margin, WIN$PRINTER will return an error.

WPRTUNITS-PIXELS (default)	Values are measured using the dots-per-inch (DPI) resolution of the output device. Only integer values are allowed in the measurements. Positioning is relative to the individual printer's physical margin. Please note that the margin set in WINPRINT-SET-MARGINS is not used to determine the cursor position. The actual size of this measurement varies depending on the target printer's resolution. This means that a coordinate of "5" will appear in a different location on a 300dpi printer than it will on a 600dpi printer. Consider the unit of measure relative to the resolution of the targeting printer before printing.

WPRTDATA-DRAW-SHAPE -- Determines if subsequent WRITE statements will be affected. A value of "0" sets the position of the write cursor. A non zero value will simply return the coordinates of the current position of the write cursor. Return values are determined by the setting of WPRTDATA-DRAW-UNITS.

Note: If you are using WPRDATA-DRAW-SHAPE to inquire the position of the write cursor, and WPRTDATA-DRAW-UNITS is set to a value other than "WPRTUNITS-PIXELS", there is a possibility that the cursor position returned may not be 100% accurate, due to rounding errors.

WINPRINT-SET-TEXT-COLOR

This operation code specifies the foreground color for text.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-TEXT-COLOR, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03  WPRTDATA-TEXT-COLOR REDEFINES
       WPRTDATA-SET-STD-FONT.   PIC 9(9) COMP-5.
```

Return Values

This option returns text color prior to the change. Use this value to restore a temporary change.

Description

When printing, color resolution is a result of a combination of three basic colors; Red, Green, and Blue (RGB). The intensity of each color in the mix is determined by a number between "0" and "255". For example, the lowest possible intensity, (0,0,0), produces black, and the highest possible intensity (255,255,255), produces white. This formula of three numbers is referred to as the COLORREF.

The ACUCOBOL-GT runtime does not provide a mechanism with which to determine the COLORREF. If you want to specify a color, you must calculate the value yourself. The following C formula can be used to calculate the COLORREF:

```
(((BYTE) (R) | ((WORD) (BYTE 9g)) <<8))|(((DWORD) (BYTE) (b)) <<16))
```

See your Windows API documentation for more information about RGB colors and COLORREF values.

The printer must be open to perform this operation. There is no need to reset this function. WPRTDATA-TEXT-COLOR should be initialized prior to use. This operation affects the color used with subsequent WRITE statements. Cursor position is not affected by this operation. This operation is ignored on non-color printers. WINPRINT-SET-TEXT-COLOR has the following value:

WPRTDATA-TEXT-COLOR -- Specifies the color used to write text. The color is indicated by the COLORREF, representing the percentage used of the three basic colors; Red, Green, and Blue (RGB). The default of "0" is solid black.

Example

This example will set the current text foreground color to light blue when printed on a color printer:

```
INITIALIZE WPRTDATA-TEXT-COLOR.
MOVE 96 TO RGB-RED.
MOVE 106 TO RGB-GREEN.
MOVE 232 TO RBG-BLUE.
PERFORM CALC-COLORREF.
MOVE COLORREF TO WPRTDATA-TEXT-COLOR.
CALL "WIN$PRINTER" USING
     WINPRINT-SET-TEXT-COLOR
     WPRTDATA-TEXT-COLOR
     GIVING CALL-RESULT.
```

The sample program "graphprn.cbl" contains an example of using the C formula described above to determine the RGB color value (CALC-COLORREF).

WINPRINT-SET-FONT

This operation code allows you to select any printer font for spooled reports.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-FONT, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-SET-FONT REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-FONT     HANDLE OF FONT.
```

Description

First, obtain a handle to the desired font with the W$FONT routine described earlier. Once you have the font handle, you can select it as the current font by setting WPRTDATA-FONT to the desired font handle and using WINPRINT-SET-FONT. The font is now associated with the current printer until you change the font again or the runtime finishes. Note that the font is only used for reports printed by the runtime's spooler handler, which you use when you assign a print file to "-P SPOOLER".

Fonts are device-specific. If you let the user change printers via WIN$PRINTER's setup operation, then the user should get a new font handle from W$FONT and associate it with the new printer.

You should not DESTROY a font handle that is currently selected as the print font, unless the printer is closed and you will not open it again. If you do, then that font will not be available to the printer.

You may use proportionally spaced fonts in print files. The runtime handles the proportional spacing correctly (in other words, it does not use a fixed width for each character). The runtime computes the number of columns that fit on a page for a proportional font by using the font's average width. (See "winspool.cbl" for a sample program that uses arbitrary fonts chosen by the user.)

WINPRINT-SET-LINES-PER-PAGE

This operation code sets the number of lines that should be printed on a page.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-LINES-PER-PAGE, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRT-PAGE-LAYOUT REDEFINES
```

```
WPRTDATA-SET-STD-FONT.
05 WPRTDATA-LINES-PER-PAGE       UNSIGNED-SHORT.
05 WPRTDATA-COLUMNS-PER-PAGE     UNSIGNED-SHORT.
```

Description

Printing forms using the Windows spooler is sometimes difficult because you cannot easily control the height of a chosen font. Most Windows fonts do not conform to older standards about font height. For example, 12 point Courier does not necessarily print at 6 lines per inch. You can use the "WINPRINT-GET-PAGE-LAYOUT" operation to determine the number of lines that fit on a page for a given font. Sometimes, however, you need to be able to set the line height explicitly.

This operation allows you to do that. It sets the number of lines that should be printed on a page. The runtime uses this number to adjust the height of the printed font. Note that the font is not scaled--it is simply printed in the specified vertical space.

In order to specify the number of lines that will fit on a page, you must consider the height or vertical resolution of each line. WINPRINT-SET-LINES-PER-PAGE obtains the page height, calculates the physical margins of the page, and sets the font height. The value of the font height includes the visible height of a letter plus its top and bottom margins. If your font height is 7 dots-per-inch (dpi), this includes the top and bottom margins of the font itself, so the *actual* font height might be only 5 dpi. Note that these values differ from font to font.

When using this option, set the number of lines desired in WPRTDATA-LINES-PER-PAGE (also defined in "winprint.def"). Then pass WINPRINT-DATA to the routine. For example:

```
* Set 60 lines per page
     MOVE 60 TO WPRTDATA-LINES-PER-PAGE
     CALL "WIN$PRINTER"
        USING WINPRINT-SET-LINES-PER-PAGE,
        WINPRINT-DATA
```

You can set the lines per page with the spooler open or closed. If you set it when it is open, then the new font height takes effect the next time the page position is advanced. In either case, the lines per page is reset to the default value the next time the spooler is closed. You can explicitly reset to the default font height by setting WPRTDATA-LINES-PER-PAGE to zero.

Note: We do not recommend using this operation when printing with "-P SPOOLER-DIRECT" or "-Q <*printername*> DIRECT=ON". When you print in DIRECT mode, the Windows print spooler has no control over the printer, the printer is not initialized by the Windows printer driver. This means that the print job uses the hardware defaults. For example, if you print in DIRECT mode to a printer with the hardware default paper size set to US letter format, that is the format used, even if the driver has A4 paper set as the default. In this situation, the operation is likely to return incorrect values.

WINPRINT-SET-MARGINS

This operation code allows you to set the margins on a report.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-MARGINS, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
   03 WPRTDATA-MARGINS REDEFINES
      WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-TOP-MARGIN          PIC 9(7)V99 COMP-5.
      05 WPRTDATA-BOTTOM-MARGIN       PIC 9(7)V99 COMP-5.
      05 WPRTDATA-LEFT-MARGIN         PIC 9(7)V99 COMP-5.
      05 WPRTDATA-RIGHT-MARGIN        PIC 9(7)V99 COMP-5.
      05 WPRTDATA-MARGIN-UNITS        UNSIGNED-SHORT.
```

Note: This group item should be initialized before it is used.

Description

When combined with the ability to set the exact height of a line (see WINPRINT-SET-LINES-PER-PAGE), this operation lets you reliably print on pre-printed forms with many different devices.
WINPRINT-SET-MARGINS sets the margins for the next report if the printer is not open, or for the current report if the printer is open. If the printer is open and the current page is blank, the margin change occurs for the current page. Otherwise, the margin change occurs on the next page.

Note: Most printers have minimum margins that cannot physically be printed in, regardless of the setting of the logical margins.

Setting margins

The four margin fields should be set to the values you want to use. For example, to set half-inch top and bottom margins you would move ".5" to WPRTDATA-TOP-MARGIN and WPRTDATA-BOTTOM-MARGIN. You can set a margin to zero to use the printer's default margins.

Note: This operation is calculates the printable area of a report when determining the capacity for lines or columns. Setting a value for WPRTDATA-RIGHT-MARGIN does not cause the print line to be truncated.

Before you can set the margins, the MARGIN-UNITS field must be set to a level 78 that describes the measurement units. The choices are:

WPRTMARGIN-DEFAULT-MARGINS -- Use printer default margins (this is the default).

WPRTMARGIN-PIXELS -- Margins expressed in printer-dependent units. Laser printers, for example, use either 300 or 600 units per inch.

WPRTMARGIN-CELLS -- Margins expressed as a number of rows or columns based on the currently selected font.

WPRTMARGIN-INCHES -- Margins expressed in inches.

WPRTMARGIN-CENTIMETERS -- Margins expressed in centimeters.

To use the margin-setting feature to simplify printing on pre-printed forms, we suggest this sequence:

1. Establish which printer you want to use.
2. Select the font.
3. Set the margins.
4. Set the line height.

It is important to note that step (3) should precede step (4) because the line height depends on the top and bottom margins that you have set. For an 11" form, a typical scenario might be:

```
Font: Courier New, 12 point
Margins: .5" top and bottom
Lines per page: 60
```

This would result in 6 lines per inch with 3 blank lines at the top and bottom of each page. The code to set the margins for this case would be:

```
INITIALIZE WPRTDATA-MARGINS
MOVE .5 TO WPRTDATA-TOP-MARGIN, WPRTDATA-BOTTOM-MARGIN
MOVE WPRTMARGIN-INCHES TO WPRTDATA-MARGIN-UNITS
CALL "WIN$PRINTER" USING WINPRINT-SET-MARGINS,
      WINPRINT-DATA
```

WINPRINT-SET-STD-FONT

This operation code allows you to select one of a number of predefined fonts to use for the report. You must make this selection prior to opening the print file.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-STD-FONT, WINPRINT-DATA
    GIVING RESULT
```

Parameters

WINPRINT-DATA Group item defined in "winprint.def" as follows:

```
01 WINPRINT-DATA.
   03 WPRTDATA-SET-STD-FONT.
      05 WPRTDATA-STD-FONT            PIC X COMP-X.
      05 FILLER                       PIC X(21).
```

Description

The font selected is used until explicitly changed or the runtime exits. You specify which font to use by moving one of the following level 78 values to WPRTDATA-STD-FONT before calling WIN$PRINTER:

WPRTFONT-DEFAULT -- Requests the printer's default font. This is the initial setting.

WPRTFONT-COURIER-12 -- Requests a 12-point TrueType Courier font.

WPRTFONT-COURIER-12-COMP Requests a 12-point TrueType Courier font and rescales it so that at least 132 columns of print will fit on a page. This is similar to the *compressed print* mode supported by many printers.

WPRTFONT-COURIER-10 -- Requests a 10-point TrueType Courier font.

WPRTFONT-COURIER-10-COMP - Requests a 10-point TrueType Courier font and rescales it so that at least 132 columns of print will fit on a page.

When you are using either of the *compressed print* modes, the rescaling of the font occurs when the print file is opened. This ensures that the font is scaled correctly for the current page size and orientation. The rescaling operation normally results in skinny characters, but can actually result in stretched characters if more than 132 characters would naturally fit on a page. The runtime asks the Windows TrueType font engine to scale the font to fit exactly 132 characters on a line even if more would normally fit.

You should be aware that the TrueType font engine does not always produce exact results, particularly when rescaling a font. You may end up with a font that fits more than 132 columns on a page. You may also end up with a font that is a slightly different height when compressed than when not. You can also end up with a font that is only vaguely related to the requested one if, for example, the user has removed the TrueType fonts or if the print driver

cannot handle TrueType fonts. These effects are due to the internal workings of the TrueType font engine and the way that Windows handles fonts in general. Under Windows, an application cannot select a particular font. Instead, it describes the font's characteristics and Windows selects the closest matching font using a weighted-penalty system for deciding which font is the closest match. Sometimes, no font matches exactly, so Windows substitutes the font that has the closest match.

WINPRINT-SET-BKMODE

Usage

```
CALL "WIN$PRINTER"
     USING WINPRINT-SET-BKMODE
     desired-bkmode
```

Parameters

bkmode

bkmode may be either:

```
78 WPRT-BKMODE-TRANSPARENT              VALUE 1.
78 WPRT-BKMODE-OPAQUE                   VALUE 2.
```

Description

WINPRINT-SET-BKMODE enables you to set the background mode for printing. This is useful for adding watermark effect to prints.

By calling this op-code, the printing mode of the current print job is set accordingly. Note that if you have multiple print jobs, and you are not setting the current print job, WINPRINT-SET-JOB must be called prior to this op-code to target the correct print. A call to this function with no active print will be ignored, and a value that differs from those specified above will be ignored.

WINPRINT-SELECTION op-codes

The following operation codes use the data item WINPRINT-SELECTION (defined in "winprint.def"). These operations are used to set the properties of a printer, including features such as printer selection, number of copies, page orientation and collating.

WINPRINT-GET-CURRENT-INFO
WINPRINT-GET-CURRENT-INFO-EX
WINPRINT-GET-NO-PRINTERS
WINPRINT-GET-PRINTER-INFO
WINPRINT-GET-PRINTER-INFO-EX
WINPRINT-GET-PRINTER-STATUS
WINPRINT-SET-PRINTER
WINPRINT-SET-PRINTER-EX
WINPRINT-SETUP-EX

WINPRINT-GET-CURRENT-INFO

This operation code returns information about the currently selected printer.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-CURRENT-INFO, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in "winprint.def" as follows:

```
01 WINPRINT-SELECTION.
   03 WINPRINT-NAME                    PIC X(80).
   03 WINPRINT-PORT                    PIC X(80).
   03 WINPRINT-DRIVER                  PIC X(80).
   03 WINPRINT-DRV-VERSION             SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS          SIGNED-SHORT.
      88 WPRTERR-NO-PRINTERS           VALUE -1.
   03 WINPRINT-IS-DEFAULT              SIGNED-SHORT.
      88 WPRT-IS-NOT-DEFAULT           VALUE 0.
      88 WPRT-IS-DEFAULT               VALUE 1.
   03 WINPRINT-COPIES                  SIGNED-SHORT.
```

```
   88 WPRT-HAS-NO-COPY              VALUE 1.
03 WINPRINT-ORIENTATION             SIGNED-SHORT.
   88 WPRT-HAS-NO-LANDSCAPE         VALUE 0.
   88 WPRT-HAS-LANDSCAPE            VALUE 1.
03 WINPRINT-QUALITY                 SIGNED-SHORT.
03 WINPRINT-CURR-ORIENTATION        SIGNED-SHORT.
03 WINPRINT-CURR-COPIES             SIGNED-SHORT.
```

Return Values

A printer is considered selected if it has performed a print using "-Q <*printername*>" or "-P SPOOLER", or if WIN$PRINTER has executed using any of the WINPRINT-DATA op-codes. If no printer is selected, this operation will return information about the Windows default printer.

Description

The printer may be open or closed to perform these functions. There is no need to reset any of these functions. WINPRINT-SELECTION should be initialized prior to use. WINPRINT-GET-CURRENT-INFO has the following values:

WINPRINT-NAME -- Returns the name of the currently selected printer.

WINPRINT-PORT -- Specifies the printer port (or UNC address) of the currently selected printer.

WINPRINT-DRIVER -- Specifies the printer driver name.

WINPRINT-DRV-VERSION -- Specifies the version number of the printer driver (vendor-specific).

WINPRINT-NO-OF-PRINTERS -- Specifies the number of the currently selected printer in the runtime's internal list.

WINPRINT-IS-DEFAULT -- Determines if the printer is the Windows print spooler default printer. If yes, the returned value is WPRT-IS-DEFAULT.

WINPRINT-COPIES -- Returns the number of copies the printer is capable of producing. Typically this number is 99 or 999.

WINPRINT-ORIENTATION -- Determines if the printer supports landscape orientation. If yes, the return value is WPRT-HAS-LANDSCAPE.

WINPRINT-QUALITY -- Returns the current setting for the varying grades of print quality.

This value applies to dot-matrix printers. Most inkjet and laser printers do not support this method of determining the level of print quality.

WINPRINT-CURR-COPIES -- Returns the current default value for the number of copies to print.

WINPRINT-GET-CURRENT-INFO-EX

This operation code returns additional information about the currently selected printer, extending the functionality of WINPRINT-GET-CURRENT-INFO.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-CURRENT-INFO-EX, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in “winprint.def” as follows:

```
01  WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
   03 WINPRINT-DRV-VERSION           SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS        SIGNED-SHORT.
   03 WINPRINT-IS-DEFAULT            SIGNED-SHORT.
   03 WINPRINT-COPIES                SIGNED-SHORT.
   03 WINPRINT-ORIENTATION           SIGNED-SHORT.
   03 WINPRINT-QUALITY               SIGNED-SHORT.
   03 WINPRINT-CURR-ORIENTATION      SIGNED-SHORT.
   03 WINPRINT-CURR-COPIES           SIGNED-SHORT.
   03 WINPRINT-DUPLEX                SIGNED-SHORT.
   03 WINPRINT-COLLATE               SIGNED-SHORT.
```

```
03 WINPRINT-COLOR                  SIGNED-SHORT.
03 WINPRINT-CURR-DUPLEX            SIGNED-SHORT.
03 WINPRINT-CURR-COLLATE           SIGNED-SHORT.
03 WINPRINT-CURR-PAPERSIZE         SIGNED-SHORT.
03 WINPRINT-CURR-TRAY              SIGNED-SHORT.
03 WINPRINT-CURR-COLOR             SIGNED-SHORT.
03 WINPRINT-JOB-TITLE              PIC X(80).
```

This group item has numerous conditional variables. See “winprint.def” for the complete list.

Return Values

A printer is considered selected if it has performed a print using “-Q <*printername*>” or “-P SPOOLER”, or if WIN$PRINTER has executed using any of the WINPRINT-DATA op-codes. If no printer is selected, this operation will return information about the Windows default printer.

Description

The printer may be open or closed to perform these functions. There is no need to reset any of these functions. WINPRINT-SELECTION should be initialized prior to use. WINPRINT-GET-CURRENT-INFO-EX has all the same values as WINPRINT-GET-CURRENT-INFO plus the following additional values:

WINPRINT-DUPLEX -- Determines if the currently selected printer supports duplex printing. If yes, the returned value is WPRT-HAS-DUPLEX.

WINPRINT-COLLATE -- Determines if the currently selected printer supports collating. If yes, the returned value is WPRT-HAS-COLLATE.

WINPRINT-COLOR -- Determines if the currently selected printer can print in color. If color printing is supported, the returned value is WPRT-HAS-COLOR.

WINPRINT-CURR-DUPLEX -- Returns the current duplex setting of the printer. Possible values are: WPRT-SIMPLEX, WPRT-DUPLEX-VERTICAL, and WPRT-DUPLEX-HORIZONTAL.

WINPRINT-CURR-COLLATE -- Determines if the collating feature of the currently selected printer is turned on or off.

WINPRINT-CURR-PAPERSIZE -- Returns the current paper size selected in the printer driver. Values less than 42 should correspond to the PAPER-SIZES table in "winprint.def". Values greater than 41 and less than 69 are defined by version 4.x of Windows NT. Values greater than 68 and less than 119 are defined in Windows 2000. Values greater than 118 and less than 256 are considered undefined. Values greater than 255 are considered user defined.

WINPRINT-CURR-TRAY -- Returns the currently selected paper tray as defined in the printer driver. Values less than 16 should correspond to the PAPER-TRAYS table in "winprint.def". Values greater than 15 and less than 256 are considered undefined. Values greater than 255 are considered device specific.

Values 12 and 13 are not defined in the PAPER-SIZES table in "winprint.def". This matches a similar gap in the Windows API. Refer to "prndemox.cbl" for an example of how to compensate for these undefined values.

WINPRINT-CURR-COLOR – Determines if the printer is in the proper mode to print in color. If yes, WPRT-COLOR is returned. Monochromatic printers or color printers with color support disabled return WPRT-MONOCHROME.

Note: When using one of these extended operations, it is best to pair it with a corresponding extended operation. For example, use WINPRINT-GET-CURRENT-INFO-EX with WINPRINT-SET-PRINTER-EX, instead of with WINPRINT-SET-PRINTER.

WINPRINT-GET-NO-PRINTERS

This operation code retrieves the number of printers installed on a system.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-NO-PRINTERS, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in "winprint.def" as follows:

```
01 WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
   03 WINPRINT-DRV-VERSION           SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS        SIGNED-SHORT.
      88 WPRTERR-NO-PRINTERS         VALUE -1.
   03 WINPRINT-IS-DEFAULT            SIGNED-SHORT.
      88 WPRT-IS-NOT-DEFAULT         VALUE 0.
      88 WPRT-IS-DEFAULT             VALUE 1.
   03 WINPRINT-COPIES                SIGNED-SHORT.
      88 WPRT-HAS-NO-COPY            VALUE 1.
   03 WINPRINT-ORIENTATION           SIGNED-SHORT.
      88 WPRT-HAS-NO-LANDSCAPE       VALUE 0.
      88 WPRT-HAS-LANDSCAPE          VALUE 1.
   03 WINPRINT-QUALITY               SIGNED-SHORT.
   03 WINPRINT-CURR-ORIENTATION      SIGNED-SHORT.
   03 WINPRINT-CURR-COPIES           SIGNED-SHORT.
```

Return Values

The number returned by this operation will be stored in WINPRINT-NO-OF-PRINTERS.

The number in WINPRINT-NO-OF-PRINTERS may differ depending on the host operating system. On a Windows 98 system, the number in WINPRINT-NO-OF-PRINTERS will represent the number of printers that are attached with a local driver. But in a Windows NT/Windows 2000 environment, the number in WINPRINT-NO-OF-PRINTERS will represent the number of printers that are attached with a local driver and possibly include the number of remote printers with drivers stored on network servers.

Description

This op-code does not alter any of the current printer settings. It is recommended, but not required, that this op-code be executed before the WINPRINT-GET-PRINTER-INFO op-code.

WINPRINT-GET-PRINTER-INFO

This operation code retrieves information about a particular printer. It does not alter any of the current printer settings.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PRINTER-INFO, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in "winprint.def" as follows:

```
01 WINPRINT-SELECTION.
   03 WINPRINT-NAME                    PIC X(80).
   03 WINPRINT-PORT                    PIC X(80).
   03 WINPRINT-DRIVER                  PIC X(80).
   03 WINPRINT-DRV-VERSION             SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS          SIGNED-SHORT.
      88 WPRTERR-NO-PRINTERS           VALUE -1.
   03 WINPRINT-IS-DEFAULT              SIGNED-SHORT.
      88 WPRT-IS-NOT-DEFAULT           VALUE 0.
      88 WPRT-IS-DEFAULT               VALUE 1.
   03 WINPRINT-COPIES                  SIGNED-SHORT.
      88 WPRT-HAS-NO-COPY              VALUE 1.
   03 WINPRINT-ORIENTATION             SIGNED-SHORT.
      88 WPRT-HAS-NO-LANDSCAPE         VALUE 0.
      88 WPRT-HAS-LANDSCAPE            VALUE 1.
   03 WINPRINT-QUALITY                 SIGNED-SHORT.
   03 WINPRINT-CURR-ORIENTATION        SIGNED-SHORT.
   03 WINPRINT-CURR-COPIES             SIGNED-SHORT.
```

Return Values

Device names up to 80 characters in length will be stored. If a name is wider than 80 characters, it will be truncated from the rightmost position. These names may contain embedded spaces. The following information will be returned:

WINPRINT-NAME -- Holds the name of the selected printer as given in the Printers folder under Settings.

WINPRINT-PORT -- Holds the name of the selected port (or UNC address) as given in the properties of the printer.

WINPRINT-DRIVER -- Holds the name of the assigned driver as seen in the properties of the printer. Note that for remote printers, this name will almost always be given as “winspool”.

WINPRINT-DRV-VERSION -- Holds the version number of the driver for the requested printer.

WINPRINT-NO-OF-PRINTERS -- Holds the number of the current printer. Remember, this number is based on the order of printers in the computer’s internal printer list, and may change from time to time. It is not recommended to identify a printer by number unless you first enumerate the printers by calling WINPRINT-GET-NO-PRINTERS.

WINPRINT-IS-DEFAULT -- Holds the value of 1 if the printer is the Windows default printer, otherwise it is set to 0.

WINPRINT-COPIES -- Holds the maximum number of copies the printer is able to provide. The most common value is 99. If a printer is not copy capable, it will have the value of 1.

WINPRINT-CURR-COPIES -- Returns the current number of copies the driver is set to print. Note that some printers return a value of 1, indicating that the original is copy number 1. Other printers appear to return a value of 0, indicating an original plus 0 copies. You can change this value to the number of copies you wish to print.

WINPRINT-QUALITY -- Returns the current setting for the varying grades of print quality.

This value applies to dot-matrix printers. Most inkjet and laser printers do not support this method of determining the level of print quality.

WINPRINT-ORIENTATION -- Holds a value indicating the orientation ability of the printer. If portrait and landscape modes are supported, this value is set to 1. If only portrait mode is supported, it is set to 0.

WINPRINT-CURR-ORIENTATION -- Returns the current orientation set in the driver, if portrait mode is active, the value is set to 1. If landscape mode is active, the value is set to 2. A value of 0 uses the printer's default setting. You can change this value to set the orientation you prefer.

Description

Calling the op-code WINPRINT-GET-NO-PRINTERS before calling WINPRINT-GET-PRINTER-INFO is recommended. However, it is not necessary to perform WINPRINT-GET-NO-PRINTERS each time you run WINPRINT-GET-PRINTER-INFO. You can perform the operation once and store the data until it is needed.

Note: If you want to retrieve information about all the printers on the system, start with a value of "1" in WINPRINT-NO-OF-PRINTERS in the WINPRINT-SELECTION record. Increment this value by one for each new execution until a negative value is returned.

WINPRINT-GET-PRINTER-INFO-EX

This operation code returns additional information about a particular printer, extending the functionality of WINPRINT-GET-PRINTER-INFO. It does not alter any of the current printer settings.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PRINTER-INFO-EX, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in "winprint.def" as follows:

```
01  WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
   03 WINPRINT-DRV-VERSION           SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS        SIGNED-SHORT.
```

```
03 WINPRINT-IS-DEFAULT              SIGNED-SHORT.
03 WINPRINT-COPIES                  SIGNED-SHORT.
03 WINPRINT-ORIENTATION             SIGNED-SHORT.
03 WINPRINT-QUALITY                 SIGNED-SHORT.
03 WINPRINT-CURR-ORIENTATION        SIGNED-SHORT.
03 WINPRINT-CURR-COPIES             SIGNED-SHORT.
03 WINPRINT-DUPLEX                  SIGNED-SHORT.
03 WINPRINT-COLLATE                 SIGNED-SHORT.
03 WINPRINT-COLOR                   SIGNED-SHORT.
03 WINPRINT-CURR-DUPLEX             SIGNED-SHORT.
03 WINPRINT-CURR-COLLATE            SIGNED-SHORT.
03 WINPRINT-CURR-PAPERSIZE          SIGNED-SHORT.
03 WINPRINT-CURR-TRAY               SIGNED-SHORT.
03 WINPRINT-CURR-COLOR              SIGNED-SHORT.
03 WINPRINT-JOB-TITLE               PIC X(80).
```

This group item has numerous conditional variables. See “winprint.def” for the complete list.

Return Values

This operation returns information about the currently selected printer using the values described in WINPRINT-GET-PRINTER and the values described below.

Description

The printer may be open or closed to perform these functions. There is no need to reset any of these functions. WINPRINT-SELECTION should be initialized prior to use. WINPRINT-GET-PRINTER-INFO-EX has all the same values as WINPRINT-GET-PRINTER-INFO plus the following additional values:

WINPRINT-DUPLEX -- Determines if the currently selected printer supports duplex printing. If yes, the returned value is WPRT-HAS-DUPLEX.

WINPRINT-COLLATE -- Determines if the currently selected printer supports collating. If yes, the returned value is WPRT-HAS-COLLATE.

WINPRINT-COLOR -- Determines if the currently selected printer can print in color. If color printing is supported, the returned value is WPRT-HAS-COLOR.

WINPRINT-CURR-DUPLEX -- Returns the current duplex setting of the printer. Possible values are: WPRT-SIMPLEX, WPRT-DUPLEX-VERTICAL, and WPRT-DUPLEX-HORIZONTAL.

WINPRINT-CURR-COLLATE -- Determines if the collating feature of the currently selected printer is turned on or off.

WINPRINT-CURR-PAPERSIZE -- Returns the current paper size selected in the printer driver. Values less than 42 should correspond to the PAPER-SIZES table in "winprint.def". Values greater than 41 and less than 69 are defined by version 4.x of Windows NT. Values greater than 68 and less than 119 are defined in Windows 2000. Values greater than 118 and less than 256 are considered undefined. Values greater than 255 are considered user defined.

WINPRINT-CURR-TRAY -- Returns the currently selected paper tray as defined in the printer driver. Values less than 16 should correspond to the PAPER-TRAYS table in "winprint.def". Values greater than 15 and less than 256 are considered undefined. Values greater than 255 are considered device specific.

Values 12 and 13 are not defined in the PAPER-SIZES table in "winprint.def". This matches a similar gap in the Windows API. Refer to "prndemox.cbl" for an example of how to compensate for these undefined values.

WINPRINT-CURR-COLOR -- Determines if the printer is in the proper mode to print in color. If yes, WPRT-COLOR is returned. Monochromatic printers or color printers with color support disabled return WPRT-MONOCHROME.

Note: When using one of these extended operations, it is best to pair it with a corresponding extended operation. For example, use WINPRINT-GET-PRINTER-INFO-EX with WINPRINT-SET-PRINTER-EX, instead of with WINPRINT-SET-PRINTER.

WINPRINT-GET-PRINTER-STATUS

This operation code allows you to check the current status of a printer. This can be used to see if a printer is available to perform a print job or not.

In some cases the printer may respond that it is ready when, in fact, there are jobs pending because the printer is out of paper or paused. (This is a feature of the Windows API.) We recommend that you check condition of the printer using the WINPRINT-JOB-STATUS operation codes.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PRINTER-STATUS, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in "winprint.def" as follows:

```
01 WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
   03 WINPRINT-DRV-VERSION           SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS        SIGNED-SHORT.
      88 WPRTERR-NO-PRINTERS         VALUE -1.
   03 WINPRINT-IS-DEFAULT            SIGNED-SHORT.
      88 WPRT-IS-NOT-DEFAULT         VALUE 0.
      88 WPRT-IS-DEFAULT             VALUE 1.
   03 WINPRINT-COPIES                SIGNED-SHORT.
      88 WPRT-HAS-NO-COPY            VALUE 1.
   03 WINPRINT-ORIENTATION           SIGNED-SHORT.
      88 WPRT-HAS-NO-LANDSCAPE       VALUE 0.
      88 WPRT-HAS-LANDSCAPE          VALUE 1.
   03 WINPRINT-QUALITY               SIGNED-SHORT.
   03 WINPRINT-CURR-ORIENTATION      SIGNED-SHORT.
   03 WINPRINT-CURR-COPIES           SIGNED-SHORT.
```

Return Values

This operation returns the printer status as defined in the Windows API.

A great variety of conditions can affect a single print job and printer status may be the result of a combination of values. This makes it impossible to catalog all possible status settings in "winprint.def". Refer to the Windows API documentation for a description of any status not covered in that file.

Description

This operation may be called any time, whether the printer is open or not. There is no need to reset this function. WINPRINT-SELECTION should be initialized prior to use. WINPRINT-NAME must be set to the name of the desired printer. WINPRINT-NAME passes the printer name as an input variable. Printer settings are not modified by this operation.

Note: If this function is executed on a networked printer with a missing or malfunctioning network, your application may appear to hang. Once the timeout has completed, your application will resume. This is a feature of the Windows API, not an effect of the runtime.

WINPRINT-SET-PRINTER

This operation code allows you to select a specific printer and set properties.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-PRINTER, WINPRINT-SELECTION
    GIVING RESULT
```

Parameters

WINPRINT-SELECTION Group item defined in “winprint.def” as follows:

```
01 WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
   03 WINPRINT-DRV-VERSION           SIGNED-INT.
   03 WINPRINT-NO-OF-PRINTERS        SIGNED-SHORT.
      88 WPRTERR-NO-PRINTERS         VALUE -1.
   03 WINPRINT-IS-DEFAULT            SIGNED-SHORT.
      88 WPRT-IS-NOT-DEFAULT         VALUE 0.
      88 WPRT-IS-DEFAULT             VALUE 1.
   03 WINPRINT-COPIES                SIGNED-SHORT.
      88 WPRT-HAS-NO-COPY            VALUE 1.
   03 WINPRINT-ORIENTATION           SIGNED-SHORT.
      88 WPRT-HAS-NO-LANDSCAPE       VALUE 0.
      88 WPRT-HAS-LANDSCAPE          VALUE 1.
   03 WINPRINT-QUALITY               SIGNED-SHORT.
```

```
03 WINPRINT-CURR-ORIENTATION        SIGNED-SHORT.
03 WINPRINT-CURR-COPIES             SIGNED-SHORT.
```

Description

WINPRINT-NAME must hold the name of the printer as received by WINPRINT-GET-PRINTER-INFO. If WINPRINT-COPIES is set to a positive value greater than one, you may use WINPRINT-CURR-COPIES to set the number of copies to print. If WINPRINT-CURR-COPIES is set to zero, the printer driver default is used. If WINPRINT-ORIENTATION is set to a positive value, then WINPRINT-CURR-ORIENTATION may be set to any of the following values:

WPRTSEL-ORIENT-DEFAULT -- For printer default.

WPRTSEL-ORIENT-PORTRAIT -- For portrait orientation.

WPRTSEL-ORIENT-LANDSCAPE -- For landscape orientation.

WINPRINT-QUALITY may be used to select varying grades of print quality. One may use the predefined constants for this purpose:

WPRTSEL-QUALITY-DEFAULT -- For printer default.

WPRTSEL-QUALITY-HIGH -- For high quality.

WPRTSEL-QUALITY-MEDIUM -- For medium quality.

WPRTSEL-QUALITY-LOW -- For low quality.

WPRTSEL-QUALITY-DRAFT -- For draft quality.

Note: WINPRINT-QUALITY only applies to dot-matrix-type printers. Most inkjet and laser printers do not support this method of determining different levels of print quality.

Specifying a printer

The steps for actually specifying a printer differ depending on whether or not you know the name of the printer.

If you know the name of the printer, set WINPRINT-NO-OF-PRINTERS to 0, and set WINPRINT-NAME to the name of the printer as given in the Printers folder under Settings. Specify any settings you desire and call "WIN$PRINTER" using WINPRINT-SET-PRINTER.

If you don't know the name of the printer, start by calling "WIN$PRINTER using WINPRINT-GET-NO-PRINTERS and storing the result in an appropriate variable. Next, enumerate the printers by iterating through the available printers starting with 1, ending with the value obtained from WINPRINT-NO-OF-PRINTERS. Stop when you have found the printer you want. Specify the settings you desire and call "WIN$PRINTER" again, this time using WINPRINT-SET-PRINTER. An example of this scenario can be found in the sample program "prndemo.cbl".

The WINPRINT-SET-PRINTER operation also allows you to change some printer settings while the printer is open, or spooling. For example, if you are printing a portrait-oriented report that contains a single-page table in landscape format, you would need to change the page orientation during printing. When you are using this feature, the only properties you can alter for this op-code are those controlling the number of copies, the page orientation and the print quality. All other properties for this op-code use the existing settings.

Comments

If you call WINPRINT-SET-PRINTER on an open print job, there is an implicit form feed. WINPRINT-SET-PRINTER must be called before you begin to print the page with the different setting, and after all printing is done on the page immediately preceding the page with the different setting. When the current page is finished, the print cursor is positioned at the top leftmost point on the new page. The new page accepts the current values specified for number of copies, page orientation and print quality, provided they are within legal parameters for the particular setting.

Because this feature of WINPRINT-SET PRINTER occurs during a print job, it alters the document being printed, particularly by setting any current page margins to the printer default values. This means that if you have customized margins and you call WINPRINT-SET-PRINTER on an open printer, you will have to reset the margins after the call to reestablish your custom margin settings.

Margin limitations may vary from portrait to landscape on the same printer and you may not be able to specify the same values for both orientations.

Changing the output device with this operation will reset any columns you have set using WINPRINT-COLUMN op-codes.

Example

Here is an example of how to specify a printer named “Gutenberg”:

```
INITIALIZE WINPRINT-NO-OF-PRINTERS.
MOVE "Gutenberg" TO WINPRINT-NAME.
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PRINTER-INFO WINPRINT-SELECTION.

*we want to set landscape orientation

MOVE WPRTSEL-ORIENT-LANDSCAPE TO WINPRINT-CURR-ORIENTATION.

*with this call, subsequent print jobs will print in landscape
*orientation from Gutenberg

CALL "WIN$PRINTER"
    USING WINPRINT-SET-PRINTER WINPRINT-SELECTION.
```

WINPRINT-SET-PRINTER-EX

This operation code allows you to select a specific printer and set properties, extending the functionality of WINPRINT-SET-PRINTER.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-PRINTER-EX, WINPRINT-SELECTION
```

Parameters

WINPRINT-SELECTION Group item defined in “winprint.def” as follows:

```
01  WINPRINT-SELECTION.
   03 WINPRINT-NAME                  PIC X(80).
   03 WINPRINT-PORT                  PIC X(80).
   03 WINPRINT-DRIVER                PIC X(80).
```

```
03 WINPRINT-DRV-VERSION            SIGNED-INT.
03 WINPRINT-NO-OF-PRINTERS         SIGNED-SHORT.
03 WINPRINT-IS-DEFAULT             SIGNED-SHORT.
03 WINPRINT-COPIES                 SIGNED-SHORT.
03 WINPRINT-ORIENTATION            SIGNED-SHORT.
03 WINPRINT-QUALITY                SIGNED-SHORT.
03 WINPRINT-CURR-ORIENTATION       SIGNED-SHORT.
03 WINPRINT-CURR-COPIES            SIGNED-SHORT.
03 WINPRINT-DUPLEX                 SIGNED-SHORT.
03 WINPRINT-COLLATE                SIGNED-SHORT.
03 WINPRINT-COLOR                  SIGNED-SHORT.
03 WINPRINT-CURR-DUPLEX            SIGNED-SHORT.
03 WINPRINT-CURR-COLLATE           SIGNED-SHORT.
03 WINPRINT-CURR-PAPERSIZE         SIGNED-SHORT.
03 WINPRINT-CURR-TRAY              SIGNED-SHORT.
03 WINPRINT-CURR-COLOR             SIGNED-SHORT.
03 WINPRINT-JOB-TITLE              PIC X(80).
```

This group item has numerous conditional variables. See “winprint.def” for the complete list.

Description

The printer may be open or closed to perform these functions. There is no need to reset any of these functions. WINPRINT-SELECTION should be initialized prior to use. WINPRINT-SET-PRINTER-EX has all the same values as WINPRINT-SET-PRINTER plus the following additional values:

WINPRINT-CURR-DUPLEX -- If WINPRINT-DUPLEX is set to a positive value greater than one, you may set this to one of the following values:

WPRT-SIMPLEX
WPRT-DUPLEX-VERTICAL
WPRT-DUPLEX-HORIZONTAL

WINPRINT-CURR-COLLATE -- If WINPRINT-COLLATE is set to a positive value greater than one, you may use WINPRINT-CURR-COLLATE to turn collating on or off.

WINPRINT-CURR-PAPERSIZE -- Because of the huge variety of paper sizes supported by different printers, this can be set to any value. Values less than 42 should correspond to the PAPER-SIZES table in “winprint.def”.

Values greater than 41 and less than 69 are defined by version 4.x of Windows NT. Values greater than 68 and less than 119 are defined in Windows 2000. Values greater than 118 and less than 256 are undefined. Values greater than 255 are user defined. The runtime accepts any value, no validation is performed.

WINPRINT-CURR-TRAY -- Because of the huge variety of paper trays supported by different printers, this can be set to any value. Values less than 16 should correspond to the PAPER-TRAYS table in "winprint.def". Values greater than 15 and less than 256 are undefined. Values greater than 255 are device specific. The runtime accepts any value, no validation is performed.

Values 12 and 13 are not defined in the PAPER-TRAYS table in "winprint.def". This matches a similar gap in the Windows API. Refer to "prndemox.cbl" for an example of how to compensate for these undefined values.

WINPRINT-CURR-COLOR -- If WINPRINT-COLOR is set to a positive value greater than one, you may use this to turn color printing on or off.

Note: When using one of these extended operations, it is best to pair it with a corresponding extended operation. For example, use WINPRINT-GET-CURRENT-INFO-EX with WINPRINT-SET-PRINTER-EX, instead of with WINPRINT-SET-PRINTER.

WINPRINT-SETUP-EX

Usage

```
CALL "WIN$PRINTER"
     USING WINPRINT-SETUP-EX WPRTDATA-SETUP-EX-FLAGS.
or
        CALL "WIN$PRINTER" USING WINPRINT-SETUP-EX.
```

Parameters

This op-code has an optional parameter (described in winprint.def):

WPRTDATA-SETUP-EX-FLAGS

This parameter may be set to one of these values:

WPRT-PRINTTOFILE (32) — By setting this, the Print To File checkbox is checked when the dialog is shown, which means when you open the print, a file save dialog will automatically show. You can also see this if, after this call, you use the WINPRINT-GET-CURRENT-INFO-EX to update the content of WINPRINT-SELECTION, in which case the WINPRINT-PORT will contain the string "FILE:". Note that if you want to print to file and do not want to have the dialog show, set the environment variable WIN-SPOOLER-PORT to a filename of your choice prior to the OPEN statement. Note that this file does not have to exist, but it must be a valid filename. If it does exist, it will be overwritten. This flag may be combined with WPRT-DISABLEPRINTTOFILE, for example:

```
ADD 32 524288 GIVING WPRTDATA-SETUP-EX-FLAGS
```

WPRT-DISABLEPRINTTOFILE (524288) — By setting this, the Print to File checkbox will show, but appears disabled so the user cannot change it. This flag may be combined with WPRT-PRINTTOFILE, for example:

```
ADD 32 524288 GIVING WPRTDATA-SETUP-EX-FLAGS
```

WPRT-HIDEPRINTTOFILE (1048576) — This will remove the Print to File checkbox, so it will not appear on the printer selection dialog.

Description

This op-code is used to invoke the Microsoft SDK PrintDlgEx printer dialog. Note that this printer dialog is considered a more modern and feature-rich function than its predecessors (PrintDlg, PageSetup) and is fully supported on Windows Vista.

Comments

This op-code requires Windows 2000 or later; if you try to execute this call on a machine equipped with Win9x, WinME, or Win NT of any version, it will return the error code:

WPRTERR-UNSUPPORTED (0)

It also requires that the application display a window prior to the call of this op-code. If your application has not displayed a window and you call this op-code, the following error code is returned:

WPRTERR-WINDOW-REQUIRED (-14)

This works also in thin client environments. Note that in thin client environments, Windows print will occur on the client side with the resources available to the client.

If you experience problems with execution, use the A-TRACE environment variable. By setting this and a trace file, both the return code from the dialog and the COM error that may have happened will be written to the trace file.

WINPRINT-COLUMN op-codes

WINPRINT-COLUMN takes the standard print line supplied by the COBOL program and breaks it up into "zones" of data, where each zone is printed at a specified column position on the page. Initially, there is only one column that starts at the left margin, has no "separation zone" between columns and has an alignment of WPRTALIGN-NONE. When printing, if there are more input columns than output columns, the unmatched input columns are not printed. If there are more output columns than input columns, the unmatched output columns are printed as if the data for those columns consisted of spaces.

Note: You should INITIALIZE WINPRINT-COLUMN first to ensure compatibility with future versions of the runtime.

WINPRINT-COLUMN is not part of the WINPRINT-DATA structure. When using WINPRINT-SET-PAGE-COLUMN, pass the WINPRINT-COLUMN structure instead of the WINPRINT-DATA structure. This will ensure that changes to WINPRINT-COLUMN can be detected dynamically by the routine.

Two op-codes control the "input" by specifying the zones in the print line. (This is similar to the DATA-COLUMNS property of the LIST-BOX control.):

- WINPRINT-SET-DATA-COLUMNS

- WINPRINT-CLEAR-DATA-COLUMNS

Three op-codes control the "output" by specifying the page layout. (This is similar to the DISPLAY-COLUMNS property of the LIST-BOX control.)

- WINPRINT-SET-PAGE-COLUMN
- WINPRINT-CLEAR-PAGE-COLUMNS
- WINPRINT-GET-PAGE-COLUMN

These operation codes use the data item WINPRINT-COLUMN (defined in "winprint.def").

WINPRINT-SET-DATA-COLUMNS

This operation code defines columns in the data when printing with a proportionally spaced font.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-DATA-C0LUMNS, CHARACTER-POSITION
    GIVING RESULT
```

Parameters

CHARACTER-POSITION Numeric value

One or more numeric values indicating each position that starts a new column of data.

Description

This is one of two op-codes that control the "input" by specifying the zones in the print line. (This is similar to the DATA-COLUMNS property of the LIST-BOX control.)

Specify one or more numeric values that represent the character position from the left side of the print record that begins a new column of data. Data starting at this column and extending to the beginning of the next column (or end of the print record) will be printed together as a single column on the

page. The first column always starts at position "1" of the print record. Initially, there is only one data column, which starts from the beginning of the print record and extends to the end of the print record.

Column specifications are additive. If you call WIN$PRINTER with this operation multiple times, all the columns specified are combined and appear in the resulting printout. Column specifications last until they are cleared, or the process that is running shuts down. See WINPRINT-CLEAR-DATA-COLUMNS for information on how to clear column specifications.

Note: The current column definitions apply to any report being printed. One advantage to this is that you can change the column definitions mid-report. To do this, simply change them before writing the print record.

Example

In the following example, the print line of a three column report is made ready for printing using a proportional font:

```
01  PRINT-RECORD.
    03  CUST-NAME     PIC X(30).
    03  FILLER        PIC X.
    03  CUST-PHONE    PIC X(15).
    03  FILLER        PIC X.
    03  CUST-BALANCE  PIC ZZZ,ZZZ,ZZZ.99-.

CALL "WIN$PRINTER" USING WINPRINT-SET-DATA-COLUMNS, 31, 47.
```

Note that this is the same as the following:

```
CALL "WIN$PRINTER" USING WINPRINT-SET-DATA-COLUMNS,
     RECORD-POSITION OF CUST-PHONE,
     RECORD-POSITION OF CUST-BALANCE.
```

WINPRINT-CLEAR-DATA-COLUMNS

This operation code clears all columns settings that have been specified in the print line.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-CLEAR-DATA-COLUMNS
    GIVING RESULT
```

Description

This is one of two op-codes that control the “input” by specifying the zones in the print line. (This is similar to the DATA-COLUMNS property of the LIST-BOX control.) This operation takes no parameters.

When executed, the printer setup returns to the default print record for columns. This is one column starting at the beginning of the print record and ending at the end of the print record.

WINPRINT-SET-PAGE-COLUMN

This operation code describes how the columns appear when printed.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-PAGE-COLUMN, WINPRINT-COLUMN
    GIVING RESULT
```

Parameters

WINPRINT-COLUMN Group item defined in “winprint.def” as follows:

```
01  WINPRINT-COLUMN, SYNC.
    03  WINPRINT-COL-START             PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-INDENT            PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-SEPARATION        PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-FONT              HANDLE OF FONT.
    03  WINPRINT-COL-UNITS             PIC 99 COMP-X.
    03  WINPRINT-COL-ALIGNMENT         PIC X.
    03  WINPRINT-TRANSPARENCY          PIC 99 COMP-X.
        88  WINPRINT-TRANSPARENT       VALUE 1, FALSE 0.
    03  WINPRINT-COL-FONTCOLOR         PIC 9(9) COMP-5 SYNC.
    03  WINPRINT-COL-FONTCOLOR-NEG     PIC 9(9) COMP-5 SYNC.

        78  WPRTUNITS-CELLS                    VALUE 0.
        78  WPRTUNITS-INCHES                   VALUE 1.
```

```
78  WPRTUNITS-CENTIMETERS              VALUE 2.
78  WPRTUNITS-PIXELS                   VALUE 3.
78  WPRTALIGN-NONE                     VALUE SPACE.
78  WPRTALIGN-LEFT                     VALUE "L".
78  WPRTALIGN-RIGHT                    VALUE "R".
78  WPRTALIGN-CENTER                   VALUE "C".
78  WPRTALIGN-DECIMAL                  VALUE "D".
78  WPRTALIGN-DECIMAL-SUPPRESS         VALUE "S".
```

Description

This is one of three op-codes that control the "output" by specifying the page layout. (This is similar to the DISPLAY-COLUMNS property of the LIST-BOX control.)

Each column of data is mapped to an output column in the print record: the first data column maps to the leftmost output column, the second data column to the next output column to the right, and so on. Each time WINPRINT-SET-PAGE-COLUMN is used, a new output column is defined. To reset the output columns, use WINPRINT-CLEAR-PAGE-COLUMNS as described below. Once set, output columns remain in effect until explicitly cleared or the runtime process shuts down.

Note: Changing the output device will also reset the columns (this occurs if you use of any of these op-codes: WINPRINT-SETUP, WINPRINT-SETUP-USE-MARGINS, WINPRINT-SET-SETTINGS, WINPRINT-SET-PRINTER).

If you describe a new column that starts in exactly the same position as a previously described column, then the new column replaces the previous column definition (replacement detection is calculated using output device units).

The fields in WINPRINT-COLUMN define the output column. The fields have the following meaning:

WINPRINT-COL-START -- Sets the leftmost point of the column on the page. The units of measurement are defined by WINPRINT-COL-UNITS. The measurement is made with respect to the left margin of the page. This position is calculated at the time that the column is defined. However, it is

always relative to the left margin, so changing the left margin will shift the columns. The column ends at the beginning of the next column or the right margin if there is no next column.

You may use this with WPRTUNITS-CELLS-ABS, WPRT-CENTIMETERS-ABS, or WPRTUNITS-INCHES-ABS to set the start position using an absolute value from the left edge of the paper.

WINPRINT-COL-INDENT -- Modifies the left edge of the column by adding its value to the WINPRINT-COL-START value. The units of measurement are defined by WINPRINT-COL-UNITS. The indent is normally set to zero. You can use a non-zero value to specify an indented column in a convenient fashion. You would typically use this when you wanted to indent a column for a particular set of output lines. Otherwise, you would have to clear all the columns and redefine them in order to change the left edge of one column.

Note: The values of WPRTUNITS-CENTIMETERS-ABS, WPRTUNITS-INCHES-ABS, and WPRTUNITS-CELLS-ABS do not affect this field because the field is always calculated as the given value.

WINPRINT-COL-SEPARATION -- Defines the width of the separation zone. This zone appears at the rightmost edge of the column. This zone is generally kept blank, but see WINPRINT-COL-ALIGNMENT for exceptions. The value specified is the width of this zone (which must be less than the width of the column). It is expressed in the units defined by WINPRINT-COL-UNITS.

Note: The values of WPRTUNITS-CENTIMETERS-ABS, WPRTUNITS-INCHES-ABS, and WPRTUNITS-CELLS-ABS do not affect this field because the field is always calculated as the given value.

WINPRINT-COL-FONT -- Sets the handle of the font to be used when printing the column. Set to NULL to use the font currently selected for the printer (this is the default). If you place a valid printer font handle in this field, then that font is used when printing this column regardless of the printer's font. Note that the printer's font still defines the height of the line.

WINPRINT-COL-UNITS -- Defines the measurement units used for WINPRINT-COL-START, WINPRINT-COL-INDENT and WINPRINT-COL-SEPARATION.

The following values are valid:

WPRTUNITS-CELLS	Values are measured using the "cell size" of the currently selected font. A font's "cell size" is the size of the '0' digit in the font. This is roughly equivalent to measuring in "characters". If you use a proportional font, it is common for uppercase characters to be wider than this measurement. If a column contains mostly uppercase data, you will need to make it wider than the number of characters in the data if you do not want to truncate the text. If a column contains numbers or mixed-case data, you can usually just set the column width to be the same as the number of characters in the data when measuring in cells. Non-integer values are allowed in the measurements.
WPRTUNITS-INCHES	Values are measured using inches.
WPRTUNITS-CENTIMETERS	Values are measured using centimeters.
WPRTUNITS-PIXELS	Values are measured using the resolution of the output device. Only integer values are allowed in the measurements. Note that the device resolution varies from device to device, and so these units are rarely used.

To measure units using an absolute value from the left edge of the page, you use WPRTUNITS-CELLS-ABS. To specify an absolute value from the left edge of the page for WINPRINT-COL-START only, you can use the following counterparts:

WPRTUNITS-CELLS-ABS
WPRTUNITS-INCHES-ABS
WPRTUNITS-CENTIMETERS-ABS

Other settings of WINPRINT-COL-UNITS are invalid.

WINPRINT-COL-ALIGNMENT -- Describes how data should be aligned in the column. The following values are allowed:

WPRTALIGN-NONE	No alignment is performed on the data, it is printed “as is”. In addition, the data is not truncated to fit the column. Any data that extends into the next column will be visible if you are printing with transparent text background, otherwise it may not be visible, as it will be overwritten when the following column is written.
WPRTALIGN-LEFT	Leading and trailing spaces are removed from the data and it is printed left aligned in the column. The text is truncated so that it does not extend into the separation zone.
WPRTALIGN-CENTER	Leading and trailing spaces are removed from the data and it is printed centered between the start of the column and the start of the column’s separation zone. Text is truncated so that it does not extend into the separation zone.

WPRTALIGN-RIGHT	Leading and trailing spaces are removed from the data and it is right aligned with respect to the beginning of the separation zone. Leading text is truncated so that it does not extend past the left edge of the column.
WPRTALIGN-RIGHT-SIGN	This is identical to WPRTALIGN-RIGHT, with the additional trait that space padding is automatically added to accommodate a trailing negative sign (“-”). For example, when printing a variable defined as “PIC ZZZ9-”, WPRTALIGN-RIGHT would align the column as follows: 220 220- WPRTALIGN-RIGHT-SIGN would align the column as follows: 220 220-

WPRTALIGN-DECIMAL	Leading and trailing spaces are removed from the data. The data is then examined to find the leftmost occurrence of the runtime's current notion of the decimal point character. The rightmost edge of the decimal point is aligned with the beginning of the separation zone. If no decimal point is found, the right edge of the data is aligned there instead. Data may extend into the separation zone and is truncated at the beginning and end of the column.
WPRTALIGN-DECIMAL-SUPPRESS	This is identical to WPRTALIGN-DECIMAL, with the additional trait that the decimal point used to align the data is replaced by a space when the data is printed. Columns with this style are limited to 256 data characters.

Any other setting of WINPRINT-COL-ALIGNMENT is invalid.

WPRTDATA-TRANSPARENCY -- When the level 88 item WPRTDATA-TRANSARENT is set to "true", then the column's foreground text is printed, but its background is left alone. This allows you to print text over something else, such as a bitmap, without erasing it. When WPRTDATA-TRANSPARENT is set to "false", then the column's background is also printed, writing over anything else on the page. Note that only the background behind the actual text printed is affected. Suppressed leading and trailing spaces are not printed.

WINPRINT-COL-FONTCOLOR -- This member of WINPRINT-COLUMN is used to specify a column's font color. This member should be set to a COLORREF (real color) value. See the Color Reference section below for details. If this member is 0 (NULL), it defaults to the color black. If this color is set, it will only be applied to the print of the column that it is associated with. It is a foreground color only.

For example, to make the entire contents of a column blue, set all other WINPRINT-COLUMN members first then code the following:

```
INITIALIZE WINPRINT-COL-FONTCOLOR-NEG.
MOVE16711680 TO WINPRINT-COL-FONTCOLOR.
CALL"WIN$PRINTER"    USING
WINPRINT-SET-PAGE-COLUMN
WINPRINT-COLUMN.
```

See the **Columns with Color** code example for a detailed demonstration of printing columns and values in color.

WINPRINT-COL-FONTCOLOR-NEG -- This member of WINPRINT-COLUMN enables you to specify an alternate color for negative numbers in a column. where the text terminates with the negative symbol, as defined on the host computer. This member should be set to a COLORREF (real color) value. See the **Color Reference** section below for details. If this member is 0 (NULL), it defaults to the color black. If this color is set and the last symbol of the column text equals the computer negative sign, it will overrule a possible use of WINPRINT-COL-FONTCOLOR and be applied only to the text that is about to be printed. It is a foreground color only.

For example, to make negative values red, set all other WINPRINT-COLUMN members first then code the following:

```
INITIALIZE WINPRINT-COL-FONTCOLOR.
MOVE X#000000FF TO
WINPRINT-COL-FONTCOLOR-NEG.
CALL"WIN$PRINTER" USING
WINPRINT-SET-PAGE-COLUMN
WINPRINT-COLUMN.
```

See the **Columns with Color** code example for a detailed demonstration of printing columns and values in color.

Real Colors (COLORREF)

COLORREF is a Windows native data item and should be declared in working storage as a PIC X(4) COMP-N item. You may also apply the SYNC clause when used internal to a group.

COLORREF is a value that can be created from the RGB (See WPAL-RED, WPAL-GREEN and WPAL-BLUE in palette.def) colors returned from the palette dialog. You can do this by using the following COMPUTE statement:

```
COMPUTE COLORREF-VAR=
WPAL-RED)+
WPAL-GREEN * 256)+
WPAL-BLUE * 65536).
```

You can also create it yourself. For instance, to create a blue color:

```
MOVE X#00FF0000 TO COLORREF-VAR.
```

To get a green color:

```
MOVE X#0000FF00 TO COLORREF-VAR.
```

To get a red color:

```
MOVE X#000000FF TO COLORREF-VAR.
```

If you want colors in between, just mix between the three values. Remember you only use 3 byte colors, so the most significant byte should be NULL.

Columns with Colors Code Example

The following code example demonstrates the use of several WIN$PRINTER operation codes and their members including: WINPRINT-SET-PAGE-COLUMN and its members: WINPRINT-COL-FONTCOLOR; WINPRINT-COL-FONTCOLOR-NEG.

The output of the program is three columns of data where the first column is blue, the second column is black, and the third column is negative numbers red, positive numbers black. Like this:

Amount 1:	500.00	**1,500.00-**
Amount 2:	2,500.00-	**2,500.00**
Amount 3:	33,500.00	**33,500.00-**
Amount 4:	444,500.00-	**444,500.00**

```
PROGRAM-ID. ColumnWithColors.
```

```
FILE-CONTROL.
SELECT      PRINT-FILE       ASSIGN TO "-P SPOOLER"
            ORGANIZATION     IS LINE SEQUENTIAL.

FILE SECTION.
FD PRINT-FILE.
01 PRINT-LINE                PIC X(80).

WORKING-STORAGE SECTION.
COPY "WINPRINT.DEF".
COPY "FONTS.DEF".
77  COLUMN-FONT             HANDLE OF FONT.
77  STANDARD-FONT           HANDLE OF FONT.

PROCEDURE DIVISION.
MAIN.

    INITIALIZE               WINPRINT-SELECTION.
    CALL    "WIN$PRINTER"    USING
            WINPRINT-GET-CURRENT-INFO-EX
            WINPRINT-SELECTION.
    SET     WPRT-COLOR       TO TRUE.
    CALL    "WIN$PRINTER"    USING
            WINPRINT-SET-PRINTER-EX
            WINPRINT-SELECTION.

    OPEN    OUTPUT           PRINT-FILE.
    INITIALIZE               WFONT-DATA
        STANDARD-FONT.
    MOVE    "Courier New"    TO WFONT-NAME.
    MOVE    12               TO WFONT-SIZE.
    SET     WFONT-BOLD       TO FALSE.
    SET     WFDEVICE-WIN-PRINTER TO TRUE.
    CALL    "W$FONT"         USING
            WFONT-GET-FONT
            STANDARD-FONT
            WFONT-DATA.
    INITIALIZE               WINPRINT-DATA
    MOVE    STANDARD-FONT    TO WPRTDATA-FONT
    CALL    "WIN$PRINTER"    USING
            WINPRINT-SET-FONT
            WINPRINT-DATA.

    INITIALIZE               WFONT-DATA
```

```
                                COLUMN-FONT.
           MOVE    "Arial"          TO WFONT-NAME.
           MOVE    12               TO WFONT-SIZE.
           SET     WFONT-BOLD       TO TRUE.
           SET     WFDEVICE-WIN-PRINTER TO TRUE.
           CALL    "W$FONT"         USING
                   WFONT-GET-FONT
                   COLUMN-FONT
                   WFONT-DATA.

           WRITE   PRINT-LINE       FROM
                  "This demo requires a COLOR printer to show
colors."
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                "It is also required with 8.1 runtime version"
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                   "The first column should be blue."
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                  "The second column should print the regular
black."
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                   "The third column should print the regular
black,"
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                  "except of negative numbers, they should be
in red."
                   BEFORE ADVANCING 1 LINE.
           WRITE   PRINT-LINE       FROM
                   "Column 3 is using ARIAL font, the rest is
Courier."
                   BEFORE ADVANCING 2 LINES.

           CALL    "WIN$PRINTER"    USING
                   WINPRINT-SET-DATA-COLUMNS
                   1
                   12
                   24.

           INITIALIZE               WINPRINT-COLUMN
                                    WINPRINT-COL-UNITS
```

```
                                    WINPRINT-COL-FONTCOLOR
                                   WINPRINT-COL-FONTCOLOR-NEG.
                MOVE    1            TO WINPRINT-COL-START.
                MOVE    0.2         TO WINPRINT-COL-SEPARATION.
                MOVE    WPRTALIGN-LEFT   TO WINPRINT-COL-ALIGNMENT.
                MOVE    1            TO WINPRINT-TRANSPARENCY.
                MOVE    ZEROS        TO WINPRINT-COL-INDENT.

                MOVE    16711680     TO WINPRINT-COL-FONTCOLOR.
                CALL    "WIN$PRINTER"    USING
                        WINPRINT-SET-PAGE-COLUMN
                        WINPRINT-COLUMN.

               MOVE    5            TO WINPRINT-COL-SEPARATION.
               MOVE    WPRTALIGN-DECIMAL TO WINPRINT-COL-ALIGNMENT.
             MOVE   0           TO WINPRINT-COL-FONTCOLOR.
                MOVE    12           TO WINPRINT-COL-START.

                CALL    "WIN$PRINTER"    USING
                        WINPRINT-SET-PAGE-COLUMN
                        WINPRINT-COLUMN.

                MOVE    COLUMN-FONT      TO WINPRINT-COL-FONT.
                MOVE    WPRTALIGN-RIGHT-SIGN TO
                        WINPRINT-COL-ALIGNMENT.
                MOVE    28           TO WINPRINT-COL-START.
                MOVE    0            TO WINPRINT-COL-FONTCOLOR.
              MOVE   255         TO WINPRINT-COL-FONTCOLOR-NEG.
                CALL    "WIN$PRINTER"    USING
                        WINPRINT-SET-PAGE-COLUMN
                        WINPRINT-COLUMN.

                MOVE    50           TO WINPRINT-COL-START.
                INITIALIZE           WINPRINT-COL-FONTCOLOR
                                    WINPRINT-COL-FONTCOLOR-NEG.
                CALL    "WIN$PRINTER"    USING
                        WINPRINT-SET-PAGE-COLUMN
                        WINPRINT-COLUMN.

                MOVE "Amount 1:      500.00     1,500.00- " TO
PRINT-LINE.
                WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.
                MOVE "Amount 2:    2,500.00-    2,500.00  " TO
PRINT-LINE.
                WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.
```

```
           MOVE "Amount 3:   33,500.00    33,500.00- " TO
PRINT-LINE.
           WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.
           MOVE "Amount 4:  444,500.00-  444,500.00  " TO
PRINT-LINE.
           WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.

         CALL "WIN$PRINTER" USING WINPRINT-CLEAR-DATA-COLUMNS.

           CLOSE PRINT-FILE.
```

WINPRINT-CLEAR-PAGE-COLUMNS

This operation code clears all columns settings that have been specified in the page layout.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-CLEAR-PAGE-COLUMNS,
    GIVING RESULT
```

Description

This is one of three op-codes that control the “output” by specifying the page layout. (This is similar to the DISPLAY-COLUMNS property of the LIST-BOX control.) This operation takes no additional parameters.

The operation removes all the output column definitions and restore to the initial state of one output column that starts at the left margin, has no separation zone and has an alignment of WPRTALIGN-NONE.

WINPRINT-GET-PAGE-COLUMN

This operation code retrieves information about a particular column on the page.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PAGE-COLUMN, COLUMN-POSITION,
```

```
           WINPRINT-COLUMN
       GIVING RESULT
```

Parameters

COLUMN-POSITION Numeric value

A numeric value indicating which column to retrieve.

WINPRINT-COLUMN Group item defined in "winprint.def" as follows:

```
01  WINPRINT-COLUMN, SYNC.
    03  WINPRINT-COL-START              PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-INDENT             PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-SEPARATION         PIC 9(7)V99 COMP-5.
    03  WINPRINT-COL-FONT               HANDLE OF FONT.
    03  WINPRINT-COL-UNITS              PIC 99 COMP-X.
    03  WINPRINT-COL-ALIGNMENT          PIC X.
    03  WINPRINT-TRANSPARENCY           PIC 99 COMP-X.
        88  WINPRINT-TRANSPARENT        VALUE 1, FALSE 0.

        78  WPRTUNITS-CELLS                    VALUE 0.
        78  WPRTUNITS-INCHES                   VALUE 1.
        78  WPRTUNITS-CENTIMETERS              VALUE 2.
        78  WPRTUNITS-PIXELS                   VALUE 3.
        78  WPRTALIGN-NONE                     VALUE SPACE.
        78  WPRTALIGN-LEFT                     VALUE "L".
        78  WPRTALIGN-RIGHT                    VALUE "R".
        78  WPRTALIGN-CENTER                   VALUE "C".
        78  WPRTALIGN-DECIMAL                  VALUE "D".
        78  WPRTALIGN-DECIMAL-SUPPRESS         VALUE "S".
```

Description

This is one of three op-codes that control the "output" by specifying the page layout. (This is similar to the DISPLAY-COLUMNS property of the LIST-BOX control.)

This operation takes two additional parameters. The first is a numeric parameter that indicates which column to retrieve. Columns are numbered from left-to-right on the printed page, starting with "1". The second parameter retrieves the current definition of the specified column and stores it in WINPRINT-COLUMN. This is useful if you want to change the

characteristics of a particular column. You can use this operation to get the current settings, change the ones you want, and use WINPRINT-SET-PAGE-COLUMN to apply the changed settings.

The units used for the various measurements are determined by setting WINPRINT-COL-UNITS in the WINPRINT-COLUMN structure passed into the call. You should set this to the desired units before calling "WIN$PRINTER". If WINPRINT-COL-UNITS contains an invalid setting, the units used by default are WPRTUNITS-CELLS.

Example

In the following example, the first column is retrieved and its current indent is set to 3 characters.

```
MOVE WPRTUNITS-CELLS TO WINPRINT-COL-UNITS
CALL "WIN$PRINTER"
   USING WINPRINT-GET-PAGE-COLUMN, 1, WINPRINT-COLUMN
MOVE 3 TO WINPRINT-COL-INDENT
CALL "WIN$PRINTER"
   USING WINPRINT-SET-PAGE-COLUMN, WINPRINT-COLUMN
```

WINPRINT-COLUMN-ALIGN-VERT

Usage

```
CALL "WIN$PRINTER"
USING WINPRINT-COLUMN-ALIGN-VERT
GIVING result
```

Return Values

This op-code returns one of the following statuses:

Positive value	Success
WPRTERR-SPOOLER-CLOSED (-6)	The op-code has been called before the print job has been started (before OPEN has been executed). This is not possible.
WPRTERR-NO-COLUMNS (-13)	The op-code has been called before any columns have been set. This is not possible.

Description

WINPRINT-COLUMN-ALIGN-VERT enables your COBOL application to support printing requests that contain fonts of alternate heights. This op-code tells the runtime to find the tallest font, and align the print so that all the print on one line comes out without any overlap of print from another line that immediately precedes or follows. Using this op-code only has meaning if you use alternate height fonts within the same line in conjunction with the WINPRINT-COLUMN feature. If you do not use alternate fonts, you do not need to call WINPRINT-COLUMN-ALIGN-VERT.

Do not call WINPRINT-COLUMN-ALIGN-VERT until all columns are set; if you do, the alignment calculations will be incorrect and the result unpredictable.

The new op-code WINPRINT-COLUMN-ALIGN-VERT is fully supported in thin client environments.

Comments

When using alternate fonts simultaneously, the runtime's internal line counter does not stay accurate with the actual line height. If you are using a variety of font heights you should take measures to control how much space there is left on the page.

WINPRINT-JOB-STATUS op-codes

The following operation codes use the data item WINPRINT-JOB-STATUS (defined in "winprint.def"). These operations are used check and modify the status of a particular printer.

WINPRINT-GET-JOB-STATUS
WINPRINT-SET-JOB-STATUS

WINPRINT-GET-JOB-STATUS

This operation code allows you to check the current status of a print job. This is useful for determining if a printer is paused or out of paper.

Note: Due to a limitation in the Windows API, computers that run Windows 9x (Windows 98, and Windows ME) do not return the spooler job ID when opening a print job. This means that you cannot use the WINPRINT-GET-JOB-STATUS operation on these machines

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-JOB-STATUS, WINPRINT-JOB-STATUS
    GIVING RESULT
```

Parameters

WINPRINT-JOB-STATUS Group item defined in “winprint.def” as follows:

```
01  WINPRINT-JOB-STATUS.
   03  WINPRINT-JOB-PRINTER           PIC X(80).
   03  WINPRINT-JOB-ID                SIGNED-INT.
   03  WINPRINT-JOB-STATUS-NO         PIC 9(9) COMP-5.
      88 WPRT-JOB-PAUSE               VALUE 1.
      88 WPRT-JOB-RESUME              VALUE 2.
      88 WPRT-JOB-CANCEL              VALUE 3.
      88 WPRT-JOB-RESTART             VALUE 4.
   03  WINPRINT-JOB-POSITION          SIGNED-INT.
   03  WINPRINT-JOB-PAGE-TOTAL        SIGNED-INT.
   03  WINPRINT-JOB-PAGE-PRINTED      SIGNED-INT.
   03  WINPRINT-JOB-STATUS-TEXT       PIC X(80).
```

Return Values

This operation returns the printer status as defined in the Windows API.

A great variety of conditions can affect a single print job and printer status may be the result of a combination of values. This makes it impossible to catalog all possible status settings in “winprint.def”. Refer to the Windows API documentation for a description of any status not covered in that file.

Description

This operation may be called at any time a print job has started, or has started and closed. There is no need to reset this function.
WINPRINT-JOB-STATUS should be initialized prior to use.

Note: If this function is executed on a networked printer with a missing or malfunctioning network, your application may appear to hang. Once the timeout has completed, your application will resume. This is a feature of the Windows API, not an effect of the runtime.

WINPRINT-GET-JOB-STATUS has the following values:

WINPRINT-JOB-PRINTER -- Should be set to the value of WINPRINT-NAME as obtained through a call to WINPRINT-GET-PRINTER-INFO(-EX) or WINPRINT-GET-CURRENT-INFO(-EX).

WINPRINT-JOB-ID -- Returns the Windows Job ID of the last print job. The printer must be open. The Job ID may be used for subsequent calls to the same printer, even if multiple jobs are printing.

WINPRINT-JOB-STATUS-NO -- Specifies the current condition of the printer, which may be one or more of the JOB-CONDITIONS defined in "winprint.def".

WINPRINT-JOB-POSITION -- Specifies a print job's current position in the queue of a particular printer. For example, if your job is third in the queue, this value is 3. This does not necessarily mean that the print job will wait until the two prior jobs in the queue have printed.

WINPRINT-JOB-PAGE-TOTAL -- Specifies the total number of pages to print.

WINPRINT-JOB-PAGE-PRINTED -- Specifies the total number of pages printed at the time of inquiry.

WINPRINT-JOB-STATUS-TEXT -- Specifies the status of the printer as a text string. Depending on the error condition, this string may be empty. This is a feature of the Windows API. Use both this parameter and WINPRINT-JOB-STATUS-NO when checking job status to be sure that you have determined the correct error condition.

WINPRINT-SET-JOB-STATUS

This operation code allows you to modify the current status of a print job.

Note: Due to a limitation in the Windows API, computers that run Windows 9x (Windows 98, and Windows ME) do not return the spooler job ID when opening a print job. This means that you cannot use the WINPRINT-SET-JOB-STATUS operation on these machines.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-JOB-STATUS, WINPRINT-JOB-STATUS
    GIVING RESULT
```

Parameters

WINPRINT-JOB-STATUS Group item defined in “winprint.def” as follows:

```
01  WINPRINT-JOB-STATUS.
   03  WINPRINT-JOB-PRINTER           PIC X(80).
   03  WINPRINT-JOB-ID                SIGNED-INT.
   03  WINPRINT-JOB-STATUS-NO         PIC 9(9) COMP-5.
      88 WPRT-JOB-PAUSE                VALUE 1.
      88 WPRT-JOB-RESUME               VALUE 2.
      88 WPRT-JOB-CANCEL               VALUE 3.
      88 WPRT-JOB-RESTART              VALUE 4.
   03  WINPRINT-JOB-POSITION          SIGNED-INT.
   03  WINPRINT-JOB-PAGE-TOTAL        SIGNED-INT.
   03  WINPRINT-JOB-PAGE-PRINTED      SIGNED-INT.
   03  WINPRINT-JOB-STATUS-TEXT       PIC X(80).
```

Return Values

This operation returns the printer status as defined in the Windows API.

A great variety of conditions can affect a single print job and printer status may be the result of a combination of values. This makes it impossible to catalog all possible status settings in "winprint.def". Refer to the Windows API documentation for a description of any status not covered in that file.

Description

This operation may not be called while the printer is open. WINPRINT-JOB-STATUS should be initialized prior to use.

Note: If this function is executed on a networked printer with a missing or malfunctioning network, your application may appear to hang. Once the timeout has completed, your application will resume. This is a feature of the Windows API, not an effect of the runtime.

WINPRINT-GET-JOB-STATUS has the following values:

WINPRINT-JOB-PRINTER -- Should be set to the value of WINPRINT-NAME as obtained through a call to WINPRINT-GET-PRINTER-INFO(-EX) or WINPRINT-GET-CURRENT-INFO(-EX).

WINPRINT-JOB-ID -- Specifies the print job to be modified. You must get the job ID number with WINPRINT-GET-JOB-STATUS, before you can set this value. If set to "0", the runtime will automatically look up the most recent print job. If a job is currently printing, that is the job that will be modified.

WINPRINT-JOB-STATUS-NO -- Modify the current print job by setting one of the following values: WPRT-JOB-PAUSE, WPRT-JOB-RESUME, WPRT-JOB-CANCEL, or WPRT-JOB-RESTART.

WINPRINT-JOB-PAGE-TOTAL -- Specifies the total number of pages to print.

WINPRINT-JOB-STATUS-TEXT -- Specifies the status of the printer as a text string. Depending on the error condition, this string may be empty. This is a feature of the Windows API. Use both this parameter and WINPRINT-JOB-STATUS-NO when checking job status to be sure that you have determined the correct error condition.

WINPRINT-MEDIA op-codes

The following operation code uses the data item WINPRINT-MEDIA (defined in “winprint.def”) to access the paper sizes and paper trays supported by a printer driver.

WINPRINT-GET-PRINTER-MEDIA

WINPRINT-GET-PRINTER-MEDIA

This operation code allows you to access the paper sizes and paper trays supported by the printer driver.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-PRINTER-MEDIA, WINPRINT-MEDIA
    GIVING RESULT
```

Parameters

WINPRINT-MEDIA Group item defined in “winprint.def” as follows:

```
01  WINPRINT-MEDIA.
   03  WINPRINT-MEDIA-PRINTER       PIC X(80).
   03  WINPRINT-MEDIA-PORT          PIC X(80).
   03  WINPRINT-MEDIA-PAPERCOUNT    SIGNED-SHORT.
   03  WINPRINT-MEDIA-TRAYCOUNT     SIGNED-SHORT.
   03  WINPRINT-MEDIA-PAPER         SIGNED-SHORT
                                    OCCURS MAX-PAPER-SIZES.
   03  WINPRINT-MEDIA-TRAYS         SIGNED-SHORT
                                    OCCURS MAX-PAPER-TRAYS.
```

Description

This operation may be called at any time a print job has started, or has started and closed. There is no need to reset this function. WINPRINT-MEDIA should be initialized prior to use.

WINPRINT-MEDIA has the following values:

WINPRINT-MEDIA-PRINTER -- Should be set to the value of WINPRINT-NAME as obtained through a call to WINPRINT-GET-PRINTER-INFO(-EX) or WINPRINT-GET-CURRENT-INFO(-EX).

WINPRINT-MEDIA-PORT -- Should be set to the value of WINPRINT-PORT from the WINPRINT-SELECTION op-code.

WINPRINT-MEDIA-PAPERCOUNT -- Returns the total number of paper sizes supported by the selected printer driver. This number varies from printer to printer. The maximum value is 41, even if the printer driver actually supports more sizes of paper.

WINPRINT-MEDIA-TRAYCOUNT -- Returns the total number of paper trays supported by the printer driver. This number varies from printer to printer. The maximum value is 13, even if the printer driver actually supports more paper trays.

WINPRINT-MEDIA-PAPER -- Returns an array of supported paper sizes. The array is limited to a maximum of 41 possible sizes. Each number in the array corresponds to a paper size defined by WINPRINT-CURR-PAPERSIZE in "winprint.def". The numbers in the array may not appear in sequential order (1,2,3...). If WINPRINT-MEDIA-PAPERCOUNT returns a value less than 41, values between the returned count and 41 are undefined. Values over 41 are not defined by the Windows API, and may be undefined, device specific or user-defined.

WINPRINT-MEDIA-TRAYS -- Returns an array of supported paper trays. The array is limited to a maximum of 13 possible trays. Each number in the array corresponds to a paper tray defined by WINPRINT-CURR-TRAY in "winprint.def". The numbers in the array may not appear in sequential order (1,2,3...). If WINPRINT-MEDIA-TRAYS returns a value less than 13, values between the returned count and 13 are undefined. Values over 13 are not defined by the Windows API, and may be undefined, device specific or user-defined.

USER-DATA op-codes

The following operation codes use data items defined by the user in Working-Storage.

WINPRINT-GET-SETTINGS
WINPRINT-SET-SETTINGS

WINPRINT-GET-SETTINGS

This operation code retrieves information about the destination device.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-GET-SETTINGS, BUFFER
    GIVING RESULT
```

Return Values

This operation returns the number of bytes used in the buffer to hold the spooler's configuration settings.

The number of bytes needed to hold the configuration can change when new settings are selected by the user, often by several hundred bytes. You should allow for a wide range of configuration sizes. Experiments suggest that 1000 bytes is adequate to hold typical configurations, but there is no guaranteed upper boundary.

Comments

The parameter, BUFFER, is a PIC X(n) data item containing the spooler's current configuration. BUFFER should be the second argument to WIN$PRINTER. The spooler's configuration includes information about the destination device, its paper size, and page orientation. It does not include information about the current font selection. The information stored in the buffer is binary data that corresponds to some internal structures used by Windows. This information should be left unchanged. This operation should be called before calling WINPRINT_SET_SETTINGS.

Note: This operation is not supported in our Thin Client environment. Use the WINPRINT-GET-PRINTER-INFO-EX operation instead.

WINPRINT-SET-SETTINGS

This operation code sets the spooler's configuration to match information stored in the specified buffer.

Usage

```
CALL "WIN$PRINTER"
    USING WINPRINT-SET-SETTINGS, BUFFER
    GIVING RESULT
```

Description

BUFFER must contain the same spooler configuration data returned by an earlier call to WINPRINT-GET-SETTINGS. Passing other data is an error that can cause a variety of problems, including unexpected printout results or a General Protection Fault in Windows.

Changing the output device with this operation will reset any columns you have set using WINPRINT-COLUMN op-codes.

Note: This operation is not supported in our Thin Client environment. Use the WINPRINT-SET-PRINTER operation instead.

WIN$VERSION

The WIN$VERSION routine returns version information for Windows and Windows NT host platforms. It provides more information about the system than is returned by the ACCEPT FROM SYSTEM-INFO statement.

Usage

```
CALL "WIN$VERSION"
    USING WINVERSION-DATA
```

Parameters

WINVERSION-DATA Group item as follows:

```
01  WINVERSION-DATA.
    03  WIN-MAJOR-VERSION     PIC X COMP-X.
```

```
03  WIN-MINOR-VERSION     PIC X COMP-X.
03  WIN-PLATFORM          PIC X COMP-X.
    88  PLATFORM-WIN-31   VALUE 1.
    88  PLATFORM-WIN-95   VALUE 2.
    88  PLATFORM-WIN-9X   VALUE 2.
    88  PLATFORM-WIN-NT   VALUE 3.
03  WIN-WORDSIZE          PIC X COMP-X.
    88  WIN-WORDSIZE-16   VALUE 1.
    88  WIN-WORDSIZE-32   VALUE 2.
```

WINVERSION-DATA is found in the COPY library “winvers.def”.

Comments

Upon return from WIN$VERSION, all of the data elements contained in WINVERSION-DATA are filled in. If you call WIN$VERSION and the host machine is not a Windows or Windows NT system, the fields are set to zero.

The WINVERSION-DATA fields have the following meaning:

WIN-MAJOR-VERSION — The major version number reported by Windows. See table below for possible values.

WIN-MINOR-VERSION — The minor version number reported by Windows. See table below for possible values.

Windows Version	WIN-MAJOR-VERSION	WIN-MINOR-VERSION
Windows 98	4	10
Windows ME	4	90
Windows XP	5	1
Windows NT	4	0
Windows 2000	5	0
Windows Vista	6	0

WIN-PLATFORM — Provides a general description of the host system. If the host is Windows NT/Windows 2000, the value is set to PLATFORM-WIN-NT. If the host is Windows 98, the value is set to PLATFORM-WIN-9X.

WIN-WORDSIZE — This item is set to WIN-WORDSIZE-32 for a 32-bit runtime.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X