© Copyright 2000 Sterling Software, Inc.
All Rights Reserved
Sterling Software
Information Management Division
5900 Canoga Avenue
P.O. Box 4237
Woodland Hills, CA 91365-4237

Notice to U.S. Government Users
Under Contracts Subject to DFARS Restricted Rights Legend
Sterling Software, Inc. makes no warranty, either expressed or implied, with respect to VISION:Builder, VISION:Two, and VISION:Workbench, their merchantability or fitness for a particular purpose, other than as expressly provided in the license agreement. This document and the software programs described herein are the proprietary and confidential information of Sterling Software, Inc. No use may be made of these software programs or this document without the express written consent of Sterling Software, Inc. or its authorized licensors.

Use, duplication, or disclosure is subject to restrictions stated in the Rights in Technical Data and Computer Software clause at DFAR 252.227-7202.1, or subparagraphs (c) (1) and (2) of the Commercial Computer Software Restricted Rights clause of 48 CFR 52.227-19, as applicable.

Notice - Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction, and disclosure are as set forth in the applicable Government Contract or Purchase Order.

Trademarks
Sterling Software, VISION:Builder, VISION:Two, and VISION:Workbench are trademarks or registered trademarks of Sterling Software, Inc. in the United States, other countries, or both.
IBM, DB2, MVS, and OS/390 are registered trademarks of IBM Corporation.
Adobe and Acrobat are registered trademarks of Adobe Systems Incorporated.
Some of the other designations used in this document are trademarks of other companies not associated with Sterling Software, Inc. Such use of these trademarks is for information purposes only and is not intended to be generic in nature nor to imply that such trademarks are owned by or licensed to Sterling Software, Inc.
## Contents

Chapter 1  
Introduction ......................................................................................................................... 1

Chapter 2  
Subroutines ......................................................................................................................... 3  
  M4TKBIT ........................................................................................................................................................... 3  
  M4TKBYTE ....................................................................................................................................................... 5  
  M4TKDATG....................................................................................................................................................... 6  
  M4TKDATJ........................................................................................................................................................ 8  
  M4TKHEXA ...................................................................................................................................................... 9

Chapter 3  
Comma Separated Values ............................................................................................................ 12  
  API Description ........................................................................................................................................... 15  
  Parameters for Converting a Field to CSV Format ................................................................................. 16
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Designated Symbols</td>
<td>21</td>
</tr>
<tr>
<td>Parameters for Terminating M4TKCSV</td>
<td>24</td>
</tr>
<tr>
<td>Using the API</td>
<td>24</td>
</tr>
<tr>
<td>Return Codes and Reason Codes</td>
<td>25</td>
</tr>
<tr>
<td>Sample Program Using M4TKCSV</td>
<td>30</td>
</tr>
<tr>
<td>Chapter 4</td>
<td></td>
</tr>
<tr>
<td>COMLIB Access API</td>
<td>34</td>
</tr>
<tr>
<td>API Description</td>
<td>34</td>
</tr>
<tr>
<td>OPEN</td>
<td>36</td>
</tr>
<tr>
<td>GETATTR</td>
<td>38</td>
</tr>
<tr>
<td>GETTABLE</td>
<td>41</td>
</tr>
<tr>
<td>CLOSE</td>
<td>43</td>
</tr>
<tr>
<td>Sample Program</td>
<td>46</td>
</tr>
<tr>
<td>Chapter 5</td>
<td></td>
</tr>
<tr>
<td>PDS User I/O Routine</td>
<td>52</td>
</tr>
<tr>
<td>Contacting Sterling Software</td>
<td>56</td>
</tr>
<tr>
<td>Index</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Chapter 1 Introduction describes this manual.

Chapter 2 Subroutines describes subroutines that enable you to perform useful functions that are not directly supported by VISION:Builder® native commands, including:

- **M4TKBIT** Translates the zeros and ones in an 8-byte field to zero and one bits in a 1-byte field.
- **M4TKBYTE** Translates the zero and one bits in a 1-byte field to zeros and ones in an 8-byte field.
- **M4TKDATG** Translates a date from Julian format (YY/DDD) to Gregorian format (MM/DD/YY, DD/MM/YY, or YY/MM/DD).
- **M4TKDATJ** Translates a date from Gregorian format (MM/DD/YY, DD/MM/YY, or YY/MM/DD) to Julian format (YY/DDD).
Chapter 3 Comma Separated Values describes M4TKCSV, an application programmer interface that converts a field to a comma-separated values (CSV) format.

Chapter 4 COMLIB Access API describes CLACCESS, an application programmer interface that accesses tables stored in the common library for use in user programs written in Assembler, COBOL, C, FORTRAN, or PL/I.

Chapter 5 PDS User I/O Routine describes how the M4PDSIN User I/O routine allows VISION:Builder applications to read a Partition Data Set (PDS) as a sequential file.

M4TKHEXA Translates a field to hexadecimal format (for example, A to C1).
The following sections describe the individual toolkit subroutines in detail. Although some subroutines may expect certain parameter fields to be of a specific type, no type checking is performed. Each subroutine will set a return code in the flag field RETURNCD when it returns. A return code of 0 indicates successful completion in all cases.

**M4TKBIT**

M4TKBIT translates the zeros and ones in an 8-byte field to zero and one bits in a 1-byte (8-bit) field. The correspondence of the bytes and bits is from left to right.

**Syntax**

```
CALL MODULE M4TKBIT USING byte-field, bit-field
```
The parameters are as follows:

**byte-field**  The field whose first eight bytes are to be translated to corresponding zero and one bits. If the field is longer than eight bytes, the trailing bytes are ignored. The field is assumed to be character-string field although no type checking is performed.

**bit-field**  The field where the zero and one bits are stored. The bits are set in order from left to right corresponding to the eight characters in the byte field. If the corresponding byte in the byte field contains a zero (0), then a zero bit is set; otherwise, a one bit is set. Only the first byte of the bit field receives the translated bits. Any remaining bytes in the field are not touched.

**Return Codes**
M4TKBIT returns the following return codes in F.RETURNCD:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>1</td>
<td>Insufficient parameters</td>
</tr>
</tbody>
</table>

**Example**

```
;Replace a byte with a specific bit pattern
LET T.FROM = '11111111'
CALL MODULE M4TKBIT USING T.FROM CODEX
;
; < Code to examine F.RETURNCD goes here >
;
;The bits of field CODEX will now be set to all ones
```
**M4TKBYTE**

M4TKBYTE translates the zero and one bits in a 1-byte field into a string of zero and one characters in an 8-byte field. The correspondence of the bits and bytes is from left to right.

**Syntax**

```
CALL MODULE M4TKBYTE USING bit-field, byte-field
```

The parameters are as follows:

- **bit-field**  The field whose first byte is translated to corresponding zero and one bytes. If the field is longer than one byte, the remaining bytes after the first byte are ignored. The field may be of any type.

- **byte-field** The field where the zero and one characters are stored. If the corresponding bit is zero, a `0' is stored; otherwise, a `1' is stored. The field should be a character type field; otherwise, unexpected results will occur. The field must be at least eight characters long. Any bytes after the first eight bytes are ignored.

**Return Codes**

M4TKBYTE returns the following return codes in F.RETURNCD:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>1</td>
<td>Insufficient parameters</td>
</tr>
</tbody>
</table>
Example

; Determine bit pattern of field
CALL MODULE M4TKBYTE USING NAME, T.CODE
;
; < Code to examine F.RETURNCD goes here >
;
; If first char of NAME was an 'A', T.CODE will now
; contain '11000001'.

M4TKDATG

M4TKDATG converts a Julian date in the format of YY/DDD to a Gregorian date format of either MM/DD/YY, DD/MM/YY, or YY/MM/DD depending upon a pattern parameter.

Syntax

CALL MODULE M4TKDATG USING julian-date, gregorian-date, pattern

julian-date

The character format field containing the Julian date in the format of YY/DDD. The subroutine is not sensitive to the separator character (it may be any character).

gregorian-date

The field where the Gregorian format date is to be stored. The field must be at least 8 characters long. The separator character for the date field components will always be a '/' character.
**Return Codes**
MK4TKDATG returns the following return codes in F.RETURNCD:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>1</td>
<td>Insufficient parameters</td>
</tr>
<tr>
<td>2</td>
<td>Julian date is not numeric</td>
</tr>
<tr>
<td>3</td>
<td>Number of days in Julian date is invalid</td>
</tr>
<tr>
<td>4</td>
<td>Pattern parameter is invalid</td>
</tr>
</tbody>
</table>

**Example**

```
; Convert Julian date to Gregorian date of format MM/DD/YY
CALL MODULE M4TKDATG USING JULDATE T.MMDDYY `MDY'

; ; Code to examine F.RETURNCD goes here
; ; T.MMDDYY will now contain a date in MM/DD/YY format
```
M4TKDATJ

M4TKDATJ converts a Gregorian date field in the format of either MM/DD/YY, DD/MM/YY, or YY/MM/DD into a Julian date field in the format of YY/DDD.

Syntax

CALL MODULE M4TKDATJ USING gregorian-date, julian-date, pattern

gregorian-date  The field that contains the Gregorian format date field. The format must correspond to the pattern specified using the pattern parameter. The subroutine is not sensitive to the separator character (it may be any character).

julian-date     The field where the Julian date of the format YY/DDD is to be stored. The field must be at least 6 characters long. The separator character will always be a `/` character.

pattern         The field that specifies how the gregorian-date parameter is to be interpreted.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Date format</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDY</td>
<td>MM/DD/YY</td>
</tr>
<tr>
<td>DMY</td>
<td>DD/MM/YY</td>
</tr>
<tr>
<td>YMD</td>
<td>YY/MM/DD</td>
</tr>
</tbody>
</table>
Return Codes
M4TKDATJ returns the following return codes in F.RETURNCD:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>1</td>
<td>Insufficient parameters</td>
</tr>
<tr>
<td>2</td>
<td>Gregorian date is not numeric</td>
</tr>
<tr>
<td>3</td>
<td>Invalid month specified in Gregorian date</td>
</tr>
<tr>
<td>4</td>
<td>Invalid day specified in Gregorian date</td>
</tr>
<tr>
<td>5</td>
<td>Invalid pattern parameter</td>
</tr>
</tbody>
</table>

Example

; Convert date of format YY/MM/DD to Julian date format
CALL MODULE M4TKDATJ USING GREGDATE T.JULIAN 'YMD'
;
; < Code to examine F.RETURNCD goes here >
;
; T.JULIAN will now contain the Julian date

M4TKHEXA

M4TKHEXA translates a field to hexadecimal so that the exact contents can be examined or printed.
Chapter 2  Subroutines

Syntax

CALL MODULE M4TKHEXA USING from-field, to-field, length

from-field  The field that contains the data to be translated to hexadecimal format. The field may be of any type and length. You can translate a maximum of 127 bytes per call of M4TKHEXA.

to-field  The field which is to receive the hexadecimal representation of the translated bytes. This must be a character type field and must be at least twice as long as the number of bytes to be translated as specified in the length parameter.

length  This must be a 4-byte integer field that specifies the number of bytes in the from-field for translation. The maximum value that this parameter may contain is 127.

Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>1</td>
<td>Insufficient parameters</td>
</tr>
<tr>
<td>2</td>
<td>Length parameter value is larger than 127</td>
</tr>
</tbody>
</table>
Example

LEN4: FIELD TYPE F LENGTH 4 INIT 4
;Translate NAME field to hex
CALL MODULE M4TKHEXA USING NAME T.HEXNAME T.LEN4
;
; < Code to examine F.RETURNCD goes here >
;
; T.HEXSAL will now contain the hexadecimal representation
; of the first 4 bytes of NAME. For example, if NAME was
; 'GRAY', T.HEXNAME will now contain 'C7D9C1E8'.
The M4TKCSV subroutine translates a field of any type and length into a comma-separated values (CSV) format. The CSV format consists of the field value, enclosed by a pair of quotation mark characters (usually double quotation marks), with a delimiting character (usually a comma) appended to the end. For example:

<table>
<thead>
<tr>
<th>Numeric value</th>
<th>48000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV format</td>
<td>“48000”,</td>
</tr>
</tbody>
</table>

Any symbol can be used to delimit, enclose, or punctuate the field. For this discussion, the most commonly specified values are used.

The converted field may contain a character that duplicates either the quotation mark character or the escape character. If this duplication occurs, M4TKCSV precedes the
character in the converted field with an escape character, usually a backslash (\). For example:

Field value  Reply with a “Y” to continue.
CSV format  “Reply with a \“Y\” to continue.”

Field value  Precede the directory name with a \\.
CSV format  “Precede the directory name with a \\,”

A numeric field may be specified as binary, packed, or zoned. When these values are converted to character, all leading zeros are suppressed. The decimal point symbol is inserted if the specified scale is greater than zero. A leading sign symbol is inserted if the user-designated symbol for the appropriate sign is not blank. If it is blank, no sign symbol is inserted. If the value of the numeric field is zero, then the converted field is a single character of zero. For example:

Field value  000059
CSV format  “59;”

Field value  45008 scale=3
CSV format  “45.008;”

Field value  0092 scale=3 negative value (symbol = -)
CSV format  “-.092;”
A floating point value is always converted to character in the familiar exponential format: \text{n.nnnnnnE+xx}.

The numeric portion is always represented in normalized form, one integer followed by six decimal place integers. The numeric portion contains a leading sign if the value is negative. Otherwise, no sign is appended. The exponent follows the character E. The exponent is always two digits and is always preceded by a sign for both positive and negative exponents.

The user-designated symbols have no effect on the format of a floating point value. Here are some examples of floating point values in CSV format.

\begin{itemize}
  \item Field value \text{20000 scale=1 positive value (symbol = +)}
  \item CSV format \text{“+2000.0”,}
  \item Field value \text{0}
  \item CSV format \text{“0”,}
  \item Field value \text{0 scale=2}
  \item CSV format \text{“.00,”}
\end{itemize}
API Description

All access to M4TKCSV is handled through the API defined in this section. The API consists of a subroutine (or function in C terminology), called M4TKCSV, with a set of required parameters.

The first parameter (required) is a function code indicating the type of operation. The remaining parameters depend on the value of this function code. The parameters used with each particular function code are described separately in the following sections.

You may request the following functions when calling M4TKCSV.

- **TOCSV**: Converts a field from a specified type and length to character format and places it in the buffer in CSV format.
- **TERM**: Indicates that this is the final call to M4TKCSV and the C environment is terminated.

The parameters follow normal calling conventions for assembler language programs and are compatible with most high-level languages. To code calls to M4TKCSV in C, COBOL, FORTRAN, or PL/I, follow the individual language's rules for making calls to assembler routines. Specify the return code and reason code parameters in the parameter list for each M4TKCSV call.

M4TKCSV register and parameter list conventions for assembler language are described in the following table. These conventions also conform to the standard conventions used by most high-level languages.

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Parameter list pointer</td>
</tr>
</tbody>
</table>
M4TKCSV runs in 31-bit mode, above the 16 megabyte line. The addressing mode (AMODE) is 31. The residency mode (RMODE) is ANY.

### Parameters for Converting a Field to CSV Format

The first parameter of the M4TKCSV subroutine specifies a function code with a value of TOCSV. You provide the characteristics of the field to be converted and the address of a buffer for the converted value in the additional parameters. The result of this call is the requested field, converted to character and in CSV format, placed in the buffer. The field is delimited by a user-designated symbol (usually a comma) and enclosed within a pair of user-designated symbols (usually double quotation marks). M4TKCSV returns the length of the converted field, a return code, and a reason code.

There are 12 required parameters for the function code TOCSV. The first nine are input parameters that you define at the time of the call. The remaining three are return parameters.

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>R13</td>
<td>Address of caller's save area</td>
</tr>
<tr>
<td>R14</td>
<td>Caller's return address</td>
</tr>
<tr>
<td>R15</td>
<td>M4TKCSV entry point address</td>
</tr>
</tbody>
</table>

### Register Usage

- R13: Address of caller’s save area
- R14: Caller’s return address
- R15: M4TKCSV entry point address
Chapter 3  Comma Separated Values

Parameters for Converting a Field to CSV Format

**Syntax**

```
CALL M4TKCSV  fcncode, fldtype, fldlength, fldscale, fldvalue,
              bufptr, bufsize, bufsizet, usersymbols,
              cnvlength, retcode, reascode
```

The following input parameters are 4-byte addresses.

- **fcncode** Points to an 8-byte area, in character format, containing the value TOCSV. Note this value must be padded with blanks to formulate an 8-byte value.
- **fldtype** Points to a 1-byte area, in character format, containing the field type.
  - F  binary
  - C  character
  - E  floating point
  - P  packed
  - Z  zoned
fldlength

Points to a 4-byte area, in binary format, containing the field length. Allowable lengths depend on the field type.

C  0 to 255
F  1 to 4
E  4
P  1 to 16
Z  1 to 16

Note that a length of zero is allowed for a character field. This creates a null value in the buffer consisting of a pair of quotation marks followed by the delimiter symbol (" ").
**fldscale** Points to a 4-byte area, in binary format, containing the scale of the field (number of decimal places). Allowable values depend on the field type.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Length</th>
<th>Minimum Scale</th>
<th>Maximum Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
<td>(fldlength * 2) - 1 to a maximum of 16</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>fldlength</td>
</tr>
</tbody>
</table>

**fldvalue** Points to the area containing the field value. The type of the field value pointed to by fldvalue is contained in fldtype. The length of the field value pointed to by fldvalue is contained in fldlength.
Chapter 3  Comma Separated Values

Parameters for Converting a Field to CSV Format

*bufptr* Points to the buffer which holds the converted field value. The length of the buffer is contained in bufsize. The offset for the buffer is contained in bufoffset.

Note that if the converted value does not fit within the confines described by the buffer size and offset, the buffer is cleared to blanks starting from the offset to the end of the buffer. Any data that may have been at that location upon input is cleared to blanks.

*bufsize* Points to a 4-byte area, in binary format, containing the length of the buffer for the converted field value. The buffer size must be greater than zero.

*bufoffset* Points to a 4-byte area, in binary format, containing the offset for the buffer, relative to zero, at which to place the converted value. The buffer offset must be greater than or equal to zero and less than the buffer size.

*usersymbols* Points to an 8-byte area, in character format, containing user-designated symbols. These symbols are used when quoting, delimiting, and punctuating fields when converting them into CSV format.

The symbols are defined in the following order: quotation mark, delimiter, escape, decimal point, positive sign, negative sign, unused, unused. See User-Designated Symbols on page 21 for more information about user-designated symbols.
The following return parameters are 4-byte addresses.

- **cnvlength**: Points to a 4-byte area, in binary format, containing the length of the converted field. The length of the converted field includes the length of the field (in character format with all necessary punctuation) plus the delimiting symbol (usually a comma) and the pair of symbols used to enclose the value (usually double quotation marks). It reflects the actual amount of space occupied by the converted field in the buffer.

- **retcode**: Points to a 4-byte area, in binary format, containing the return code. See Return Codes and Reason Codes on page 25 for more information about return codes.

- **reascode**: Points to a 4-byte area, in binary format, containing the reason code. See Return Codes and Reason Codes on page 25 for more information about reason codes.

### User-Designated Symbols

User-designated symbols are used when quoting, delimiting, and punctuating fields. You define them as input parameters to M4TKCSV. There are no default values; each symbol must be specified.

These symbols are inserted into the field value after it has been converted to character format. All references to the field value in this section refer to the value after it has been converted to character format.

The following symbols are defined and discussed:

- Quotation Mark
- Delimiter
- Escape
- Decimal Point
- Positive Sign
- Negative Sign
**Quotation Mark**
The quotation mark symbol is used to enclose the value. It is inserted at the beginning and the end of the value. It typically is specified as double quotation marks (") but can be any symbol.

Example: “900.3”

**Delimiter**
The delimiter symbol is used to delimit the value from any value that may follow it in the buffer. It is appended to the end of the value after the added quotation marks. It typically is specified as a comma (,) but can be any symbol.

Example: “California”,

**Escape**
The escape symbol is used when a value contains a character that is the same as the quotation mark symbol or the escape symbol. When this duplication occurs, the escape symbol is inserted into the value preceding the character. It typically is specified as a backslash (\) but can be any symbol.

Example: “Reply with a \”Y\” to continue.”,

**Decimal Point**
The decimal point symbol is used when a numeric field (type F, P, or Z) has a non-zero scale. The decimal point symbol is inserted into the value at the relevant position to indicate its scale. It typically is specified as a period (.) but can be any symbol.

Example: “4100.50”,
Example if symbol is , (comma): “4100,50”,


This specification has no effect on the format of a floating point number (type E). A decimal point is always indicated with a period (.), regardless of the symbol you designated.

Example if symbol is , (comma): “5.999712E+02”,

Positive Sign
The positive sign symbol is used when a numeric field (type F, P, or Z) has a value greater than or equal to zero. If the symbol is not blank, it is inserted into the value prior to the first digit. If the symbol is blank, nothing is inserted. It typically is specified as a blank but can be any symbol. If not blank, it typically is specified as a plus sign (+).

Example if symbol is blank: “625”,
Example if symbol is +: “+625”,

This specification has no effect on the format of a floating point number (type E). A positive sign is always indicated with a plus sign (+), regardless of the symbol you designated.

Example if symbol is # (pound sign): “5.999712E+02”,

Negative Sign
The negative sign symbol is used when a numeric field (type F, P, or Z) has a value less than zero. If the symbol is not blank, it is inserted into the value prior to the first digit. If the symbol is blank, nothing is inserted. If the symbol is either a left parenthesis (()) or a right parenthesis ()), the value is bracketed with a pair of parentheses. It typically is specified as a minus sign (-) but can be any symbol.

Example if symbol is blank: “625”,
Example if symbol is - : “-625”,
Example if symbol is ( : “(625)”,
Example if symbol is (): “(625)”,
This specification has no effect on the format of a floating point number (type E). A negative sign is always indicated with a minus sign (-) regardless of the symbol you designated.

Example if symbol is # (pound sign): “-5.999712E-02”,

**Parameters for Terminating M4TKCSV**

A function code with a value of TERM terminates the C environment when no more calls are to be made to M4TKCSV.

This function code requires the following parameter to be passed to M4TKCSV.

**Syntax**

```
CALL M4TKCSV fcncode
```

The following input parameter is a 4-byte address.

`fcncode` Points to an 8-byte area, in character format, containing the value TERM. Note this value must be padded with blanks to formulate an 8-byte value.

**Using the API**

An application typically calls M4TKCSV repeatedly for each converted field. With each call, you want the field to be appended to the end of the data that already resides in the buffer from previous calls. To accomplish this, elect to increment either the buffer address or the buffer offset by the value passed back in the converted field length return parameter (cnvlength). Each subsequent call to M4TKCSV uses this incremented value as an input parameter, effectively causing the field to be appended to the buffer.
When the application has all the desired fields in the buffer, it typically writes the buffer to an output file. Prior to this, some final processing may be necessary. Every field placed in the buffer, including the last field, is followed by a delimiter (usually a comma). It is the application’s responsibility to remove the delimiter from the last field, if it is not wanted. If you want the buffer to terminate with a carriage return/line feed (CR/LF), it is your responsibility to append the appropriate characters to the end of the buffer.

You may want a null value to be placed in the buffer under certain conditions. This is accomplished by calling M4TKCSV with parameters specifying a character field with a length of zero. This causes a pair of quotation mark symbols and a delimiter symbol to be placed in the buffer.

When all processing is complete, make a final call to M4TKCSV with a function code of TERM. This terminates the C environment. Although this is not a requirement, it is recommended in order for M4TKCSV to perform all necessary memory management, cleanup, and file closing. Failure to make the termination call may result in missing error and diagnostic messages, or other unpredictable results.

### Return Codes and Reason Codes

The following table lists the return codes that M4TKCSV passes back in the retcode parameter. Each return code has a set of reason codes associated with it. The reason codes are set in the reascode parameter.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The field converted successfully.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An error occurred.

1. The function code was not recognized. The function code must be either TOCSV or TERM. Note that each value must be padded with blanks to formulate an 8-byte value.

2. The field type was not recognized. The field type must be E, F, G, P, or Z.

3. The length of the binary field was not within the allowable limits. The length of a binary field must be between 1 and 4.

4. The scale of the binary field was not within the allowable limits. The scale of a binary field must be between 0 and the largest scale allowed for the specified field length.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Reason Code</th>
<th>Description (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>An error occurred.</td>
<td>1</td>
<td>The function code was not recognized. The function code must be either TOCSV or TERM. Note that each value must be padded with blanks to formulate an 8-byte value.</td>
</tr>
<tr>
<td>2</td>
<td>The field type was not recognized. The field type must be E, F, G, P, or Z.</td>
<td>3</td>
<td>The length of the binary field was not within the allowable limits. The length of a binary field must be between 1 and 4.</td>
</tr>
<tr>
<td>4</td>
<td>The scale of the binary field was not within the allowable limits. The scale of a binary field must be between 0 and the largest scale allowed for the specified field length.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>Maximum Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>The binary field did not convert to character successfully.</td>
</tr>
<tr>
<td>6</td>
<td>The length of the character field was not within the allowable limits. The length of a character field must be between 0 and 32767.</td>
</tr>
<tr>
<td>7</td>
<td>The scale of the character field was not zero.</td>
</tr>
<tr>
<td>8</td>
<td>The length of the floating point field was not within the allowable limits. The length of a floating point field must be 4.</td>
</tr>
<tr>
<td>9</td>
<td>The scale of the floating point field was not zero.</td>
</tr>
<tr>
<td>10</td>
<td>The floating point field did not convert to character successfully.</td>
</tr>
<tr>
<td>11</td>
<td>The length of the packed field was not within the allowable limits. The length of a packed field must be between 1 and 16.</td>
</tr>
<tr>
<td>12</td>
<td>The scale of the packed field was not within the allowable limits. The scale of a packed field must be between 0 and twice the specified field length minus 1, up to a maximum scale of 16.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>13</td>
<td>The packed field did not convert to character</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>The length of the zoned field was not within the</td>
</tr>
<tr>
<td></td>
<td>allowable limits. The length of a zoned field</td>
</tr>
<tr>
<td></td>
<td>must be between 1 and 16.</td>
</tr>
<tr>
<td>15</td>
<td>The scale of the zoned field was not within the</td>
</tr>
<tr>
<td></td>
<td>allowable limits. The scale of a zoned field</td>
</tr>
<tr>
<td></td>
<td>must be between 0 and the specified field length.</td>
</tr>
<tr>
<td>16</td>
<td>The zoned field did not convert to character</td>
</tr>
<tr>
<td></td>
<td>successfully. It did not consist of valid digits</td>
</tr>
<tr>
<td></td>
<td>or did not contain a valid sign and digit</td>
</tr>
<tr>
<td></td>
<td>combination in the last position.</td>
</tr>
<tr>
<td>8</td>
<td>A severe error occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The length of the user buffer was exceeded. The converted field, including all punctuation, did not fit within the buffer provided.

Note that if the converted value does not fit within the confines described by the buffer size and offset, the buffer is cleared to blanks starting from the offset to the end of the buffer. Any data that may have been at that location upon input is cleared to blanks.

An error condition occurs when an incorrect number of parameters is specified. There must be exactly 12 parameters specified. Or, if the function code is ‘TERM’, then either 1 or 12 parameters must be specified. If the CSV System detects that the number of parameters is incorrect, it sets a value of 12 into register 15 and returns immediately, without performing any processing. Because there was an error in the parameter list itself, the return code and reason code parameters cannot be set by the CSV Systems. The only indication of this error is the return value in register 15.
## Sample Program Using M4TKCSV

<table>
<thead>
<tr>
<th>RUN</th>
<th>STMT</th>
<th>FILE</th>
<th>ONSDUTRAB</th>
<th>PSDSL</th>
<th>RB</th>
<th>RSMRS</th>
<th>START</th>
<th>END</th>
<th>NAME</th>
<th>TYPE</th>
<th>LEEIPREUURLCS</th>
<th>JOFS</th>
<th>SEARCH</th>
<th>SEARCH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>DWRQDNPDFTMTN</td>
<td>TK</td>
<td>TPPOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1 Sample Program Using M4TKCSV (Page 1 of 4)
***** ASL Procedure Follows - Using Translator Version x.xx

**PROC

; EACH MASTER-FILE-RECORD CONTROL PROCEDURE
;
FUNCTION: FIELD C 8 INIT 'TOSCSV'

USERSYM: FIELD C 8 INIT '.*\._ -'

FULLNAME: FIELD C 25

CUTOFF: FIELD C 80

OFFSET: FIELD F 4

OUTLEN: FIELD F 4

RETCODE: FIELD F 4

REASCODE: FIELD F 4

RUNSIZE: FIELD F 4 INIT 80

NUMSZ: FIELD F 4 INIT 25

TRN: FIELD F 4 INIT 10

FIVE: FIELD F 4 INIT 5

THREE: FIELD F 4 INIT 3

TWO: FIELD F 4 INIT 2

ONE: FIELD F 4 INIT 1

ZERO: FIELD F 4 INIT 0

LET T.CUTOFF = ' '

LET T.FULLNAME = FIRSTNAME
IF MIDINIT ME ' '

COMBINE FIRSTNAME MIDINIT STORE T.FULLNAME BLANKS 1

COMBINE T.FULLNAME '.' STORE T.FULLNAME BLANKS 0

END

COMBINE T.FULLNAME LASTNAME STORE T.FULLNAME BLANKS 1

LET PP(T.USERSYM,2,1) = ',', ; Set separator to comma

LET T.OFFSET = 0

CALL MODULE M4TKCSV USING T.FUNCTION, 'C', T.NUMSZ, T.ZERO, T.FULLNAME,

T.CUTOFF, T.RUNSIZE, T.OFFSET, T.USERSYM,

T.OUTLEN, T.RETCODE, T.REASCODE

LET T.OFFSET = T.OFFSET + T.OUTLEN
CALL MODULE M4TKCSV USING T.FUNCTION, 'C', T.ONE, T.ZERO, SEL,
T.OUTBUFF, T.HFSIZE, T.OFFSET, T.USERSYM,
T.OUTLEN, T.RETCODE, T.REACODE
;
LET T.OFFSET = T.OFFSET + T.OUTLEN
CALL MODULE M4TKCSV USING T.FUNCTION, 'C', T.TWO, T.ZERO, WORDSPT,
T.OUTBUFF, T.HFSIZE, T.OFFSET, T.USERSYM,
T.OUTLEN, T.RETCODE, T.REACODE
;
LET T.OFFSET = T.OFFSET + T.OUTLEN
CALL MODULE M4TKCSV USING T.FUNCTION, 'C', T.TEN, T.ZERO, HIREDATE,
T.OUTBUFF, T.HFSIZE, T.OFFSET, T.USERSYM,
T.OUTLEN, T.RETCODE, T.REACODE
;
LET PF(T.USERSYM,2,1) = ' ' ;Set separator to blank for last field
LET T.OFFSET = T.OFFSET + T.OUTLEN
CALL MODULE M4TKCSV USING T.FUNCTION, 'P', T.FIVE, T.TWO, SALARY,
T.OUTBUFF, T.HFSIZE, T.OFFSET, T.USERSYM,
T.OUTLEN, T.RETCODE, T.REACODE
;
CALL SUBFILE CSVOUT
#VEND
***** End of ASL Procedure

**************************************************************************************
*  REQUEST NAME = CSVOUT  *
*  INPUT STREAM REQUEST  *
**************************************************************************************

<table>
<thead>
<tr>
<th>STMT REPORT</th>
<th>MAX SEL SUM VERT FORMS PAGE PAGE LINE REQ</th>
<th>BACK RMT AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE REPORT</td>
<td>DATE REQUEST ID ITEMS CTL RST SP CNTRL WITH RMT NOS TYPE SET NAME CONTROL IMP FM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SR) (SG) (SN)</td>
<td></td>
</tr>
<tr>
<td>ST SLM V 8 PG PG 1 SP MAXIMUM TB COLUMN DT PG STRT LIN SL RH SUBFILE........ C E A S R S N C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TY RST S LPI WID HST M FHM LFP PGS TO HEADING PGS POS PAGE NUM UB FN RMT NAME BLM/ FRM H F G T S U D O</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP F G</td>
<td>LI TYP POS NUM ML TD REC BUF C C S L T M S N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SR) (CG) (CSVOUT) (F)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1 Sample Program Using M4TKCSV (Page 3 of 4)
### Figure 1
Sample Program Using M4TKCSV (Page 4 of 4)
Chapter 4

COMLIB Access API

VISION:Builder uses a component called COMLIB to perform all access to the common library. The COMLIB component is a shared component used by the VISION:Builder, VISION:Workbench™ for ISPF, and other Sterling Software products.

The COMLIB component modules retrieve table definitions for use in user programs written in Assembler, COBOL, C, FORTRAN, or PL/I. With this API, you can access tables stored in the common library using the same stable and consistent access provided by COMLIB.

API Description

The COMLIB Access API consists of a subroutine (or function in C terminology) with a set of required parameters that includes a function code, indicating the operation to be performed. The subroutine is called CLACCESS and the first parameter (required) is always a function that describes the action you want CLACCESS to take.
Summary of CLACCESS Functions

You can use the following functions with CALL CLACCESS:

OPEN       Establishes the connection to COMLIB and performs the initialization required to begin retrieving tables.

GETATTR    Returns the size of (the memory required for) a table and its attributes (such as number of rows, argument field type and size, result field type and size).

GETTABLE   Retrieves a table and stores it in a specified area.

CLOSE      Terminates the connection to COMLIB and performs any necessary cleanup activities.

The parameters used with each particular function are described with the syntax and usage of the function. The parameters follow typical calling conventions for assembler language programs and are compatible with most high-level languages.

To code CLACCESS calls in C, COBOL, FORTRAN, or PL/I, follow the individual language's rules for making calls to assembler routines. Specify the return code and reason code parameters in the parameter list for each CLACCESS call.

Register Conventions

If you do not specify the return code and reason code parameters in your CLACCESS calls, CLACCESS puts a return code in register 15 and a reason code in register 0. CLACCESS supports high-level languages that cannot interrogate individual registers. The contents of registers 2 through 14 are preserved across calls. CLACCESS register and
parameter list conventions for assembler language are described in the following table. These conventions also conform to the standard conventions used by most high-level languages.

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Parameter list pointer</td>
</tr>
<tr>
<td>R13</td>
<td>Address of caller’s save area</td>
</tr>
<tr>
<td>R14</td>
<td>Caller’s return address</td>
</tr>
<tr>
<td>R15</td>
<td>M4TKCSV entry point address</td>
</tr>
</tbody>
</table>

**OPEN**

OPEN establishes a connection to COMLIB and performs the necessary initialization.

**Syntax**

```
CALL CLACCESS (function, ribptr, worksize, ddname<, retcode<, reascode>>)
```

The parameters are as follows:

- **function**: An 8-byte area containing the word OPEN followed by four blanks.
- **ribptr**: Points to an 8-byte area where CLACCESS places the address of the release information block (RIB) after the call. For example, `nn.naaax`, where `nn.n` is the COMLIB release number, `aaa` is either the BDM (for BDAM only), VSM (for VSAM only), or B/V (for BDAM/VSAM combination) version of COMLIB, and `x` is the unused byte containing a zero (X’00’).
Usage
OPEN allocates the necessary resources needed to access the common library and performs the related operating system data set access initialization for each of the common library data sets.

Once a successful OPEN has been performed, as many tables as needed may be retrieved. OPEN should not be called a second time until an intervening CLOSE has been performed.
Example

In Assembler:  CALL CLACCESS, (OPENFN, RIBPTR, WORKSIZE, DDNAME, RETCODE, X
X REASCODE), VL

In C:  claccess ( &openfn[0], &ribptr[0], &worksize, &ddname, &retcode, &reascode );

In COBOL:  CALL 'CLACCESS' USING OPENFN RIBPTR WORKSIZE DDNAME RETCODE REASCODE.

In FORTRAN:  CALL CLACCESS (OPENFN, RIBPTR, WORKSIZE, DDNAME, RETCODE, REASCODE)

In PL/I:  CALL CLACCESS (OPENFN, RIBPTR, WORKSIZE, DDNAME, RETCODE, REASCODE);

GETATTR

GETATTR returns the table attribute information to the caller. This information includes the memory requirements needed to retrieve and store the entire table.

Syntax

CALL CLACCESS (function, tablenam, attrptr<, retcode<, reascode>>)

The parameters are as follows:

function  Points to an 8-byte area containing the word GETATTR followed by one blank.

tablenam  Points to an 8-byte area containing the name of the table (padded with trailing blanks if necessary) to be retrieved.
attrptr Points to an area where CLACCESS places the table attribute information. The attribute information area is described in Figure 2 on page 40. The area must be as large as the block size (BDAM) or control interval size (VSAM) of the COMLIB data set.

retcode Points to a 4-byte area where CLACCESS places the return code. The field is optional. If not provided, CLACCESS places the return code in register 15 and the reason code in register 0.

reascode Points to a 4-byte area where CLACCESS places a reason code. If not provided, CLACCESS places the reason code in register 0. When the return code is less than 12, the reason code is an integer. When the return code is 12, the reason code is a COMLIB message ID code. The field is optional. If provided, retcode must also be provided.
**Attribute Information Area**

Figure 2 shows the attribute information area description in assembler language.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLENAM</td>
<td>CL8</td>
<td>NAME OF TABLE</td>
</tr>
<tr>
<td>TABLSIZE</td>
<td>F</td>
<td>TOTAL SIZE OF TABLE IN BYTES</td>
</tr>
<tr>
<td>TABLROWS</td>
<td>H</td>
<td>NUMBER OF ROWS IN TABLE</td>
</tr>
<tr>
<td>TABLLEN</td>
<td>H</td>
<td>SIZE OF ONE TABLE ROW</td>
</tr>
<tr>
<td>TABLTYPE</td>
<td>H</td>
<td>TABLE TYPE CODE - * 0=DISPLACEMENT, 4=SEQ, 8=BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>TABLATYP</td>
<td>H</td>
<td>ARGUMENT FIELD TYPE - * 0=CHAR, 4=ZONED DEC, 8=PACKED DEC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12=INTEGER, 16=FLOAT</td>
</tr>
<tr>
<td>TABLLEN</td>
<td>H</td>
<td>ARGUMENT FIELD LENGTH (1-255)</td>
</tr>
<tr>
<td>TABLDEC</td>
<td>H</td>
<td>ARGUMENT FIELD DECIMAL PLACES (0-9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESULT FIELD TYPE (SEE ABOVE)</td>
</tr>
<tr>
<td>TABLREN</td>
<td>H</td>
<td>RESULT FIELD LENGTH (1-255)</td>
</tr>
<tr>
<td>TABLRDEC</td>
<td>H</td>
<td>RESULT FIELD DECIMAL PLACES (0-9)</td>
</tr>
<tr>
<td></td>
<td>CL8</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

Zoned type arguments are converted to packed type by COMLIB when the table is stored in the common library. As a result, when the table is retrieved, the table attribute information area indicates that the argument field is of type packed instead of zoned. This is an operating characteristic of COMLIB and cannot be overridden. If packed type argument fields are not acceptable, define the table with character type arguments.

**Usage**

GETATTR returns the table attribute information to the caller. If a successful OPEN has not been previously performed or if the table name, as specified in the tablenam parameter, does not exist in the common library, the entire table attribute information is cleared to zeros (every byte contains X’00’).
GETTABLE retrieves the table attribute information along with all of the table entries into a storage area provided by the caller. If dynamic storage allocation for the table is being used, GETATTR must be called first to determine the size of the table. The TABLSIZE value in the attribute information includes the attribute area size, plus space for one dummy table row, plus TABLROWS rows of size TABLRLEN. If static storage allocation is being used, the size of the defined static area must be at least as large as TABLSIZE.

**Syntax**

```c
CALL CLACCESS (function, tablenam, areaptr<, retcode<, reascode>>);
```

The parameters are as follows:

- **function** Points to an 8-byte area containing the word GETTABLE.
- **tablenam** Points to an 8-byte area containing the name of the table (padded with trailing blanks if necessary) to be retrieved.
areaptr

Points to a 4-byte area containing a pointer to the target area where the table attribute information along with the table entries are stored. This target area must be at least as large as the space needed by the table attribute information in addition to all of the table entries. If the target area is not large enough, unpredictable results occur.

The table entries area begins with space for a dummy table row followed by the actual table rows for as many rows as the TABLROWS value in the table attribute information area indicates. This dummy entry can store the most recent results of a table search, optimizing the next search.

Note that DISPLACEMENT tables do not contain an argument field because these tables are intended to be accessed using an index. The first row returned for a DISPLACEMENT table (following the dummy entry) is an index value of 0. Result values for rows in a DISPLACEMENT table which were not defined when the table was cataloged (that is, rows for unspecified intermediate argument values less than the high value) will contain all binary zeros regardless of the result data type. (See the VISION:Builder User's Guide for more information regarding these tables.)

retcode

Points to a 4-byte area where CLACCESS places the return code.

The field is optional. If not provided, CLACCESS places the return code in register 15 and the reason code in register 0.

reascode

Points to a 4-byte area where CLACCESS places a reason code. If not provided, CLACCESS places the reason code in register 0. When the return code is less than 12, the reason code is an integer. When the return code is 12, the reason code is a COMLIB message ID code.

The field is optional. If provided, retcode must also be provided.
Usage
GETTABLE retrieves the table specified in the tablenam parameter into the area specified in the areaptr parameter. If a successful OPEN has not been previously performed or if the table name, as specified in the tablenam parameter, does not exist in the common library, the table is not retrieved.

CLOSE
CLOSE terminates the connection to COMLIB and releases any resources obtained by the OPEN call.

Syntax
CALL CLACCESS (function<,retcode<,reascode>>)

The parameters are as follows:

function Points to an 8-byte area containing the word CLOSE followed by three blanks.

retcode Points to a 4-byte area where CLACCESS places the return code.

The field is optional. If not provided, CLACCESS places the return code in register 15 and the reason code in register 0.

reascode Points to a 4-byte area where CLACCESS places a reason code. If not provided, CLACCESS places the reason code in register 0. When the return code is less than 12, the reason code is an integer. When the return code is 12, the reason code is a COMLIB message ID code.

The field is optional. If provided, retcode must also be provided.
Usage
Once CLOSE is called, no more GET calls can be made until OPEN is successfully called again.

Return and Reason Codes
The following table lists the return codes that CLACCESS passes back in the retcode parameter. Each return code has a set of reason codes associated with it. The codes are returned either to the variables named in the return and reason code parameters or, if those parameters are not used, to registers 15 and 0.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>4</td>
<td>Not found</td>
<td>1</td>
<td>Named table is not present in the library.</td>
</tr>
<tr>
<td>8</td>
<td>Programming error</td>
<td>1</td>
<td>Unknown function call was received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Wrong number of parameters or the end of list bit was off.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Received a second OPEN without a CLOSE.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>CLOSE received before a successful OPEN.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>GETATTR received before a successful OPEN.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>GETTABLE received before a successful OPEN.</td>
</tr>
</tbody>
</table>
Unable to obtain the required resources.

Internal COMLIB error. Reason code is the low order 4 bytes of the COMLIB message ID code. See the COMLIB System Messages manual for an explanation of the message. CLACCESS will perform a CLOSE, if possible. Do not attempt to call CLACCESS.

The most common codes are `CV00' indicating that the working storage size in the OPEN call was too small and `CG00' indicating that the DD statement corresponding to the ddname parameter on the OPEN call was missing.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Reason Code</th>
<th>Description (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Unable to obtain the required resources.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Internal error</td>
<td>Message ID</td>
<td>Internal COMLIB error. Reason code is the low order 4 bytes of the COMLIB message ID code. See the COMLIB System Messages manual for an explanation of the message. CLACCESS will perform a CLOSE, if possible. Do not attempt to call CLACCESS. The most common codes are <code>CV00' indicating that the working storage size in the OPEN call was too small and </code>CG00' indicating that the DD statement corresponding to the ddname parameter on the OPEN call was missing.</td>
</tr>
</tbody>
</table>
Sample Program

Figure 3 lists a sample COBOL program retrieving a table and displaying its attributes and table entries.

IDENTIFICATION DIVISION.
PROGRAM-ID. CLATCOB.
AUTHOR. STERLING SOFTWARE.
DATE-WRITTEN. TODAY.
DATE-COMPILED. TODAY.
*
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
*
*
DATA DIVISION.
WORKING-Storage SECTION.
*
01 CLAFUNC PIC X(8).
*
01 CLA TABLE PIC X(8).
*
01 CLA WORK-SIZE PIC 9(9) COMP.
*
01 DDNAME PIC X(8).
*
01 RETURN-VALUES.
05  ERROR-CODE          PIC 9(9) COMP.
05  REASON-CODE         PIC 9(9) COMP.
05  MSG-ID REDEFINES REASON-CODE PIC X(4).

* 01  RELEASE-INFO         PIC X(8).

* 01  SAMPLE-TABLE.
   05  TABLE-ATTR.
      10  TABLE-NAME       PIC X(8).
      10  TABLE-SIZE       PIC 9(9) COMP.
      10  NUM-ENTRIES      PIC 9(4) COMP.
      10  ENTRY-SIZE       PIC 9(4) COMP.
      10  TABLE-TYPE       PIC 9(4) COMP.
      10  FILLER           PIC X(2).
      10  ARG-TYPE         PIC 9(4) COMP.
      10  ARG-SIZE         PIC 9(4) COMP.
      10  ARG-DIGITALS      PIC 9(4) COMP.
      10  RES-TYPE         PIC 9(4) COMP.
      10  RES-SIZE         PIC 9(4) COMP.
      10  RES-DIGITALS      PIC 9(4) COMP.
      10  FILLER           PIC X(8).

* THE FOLLOWING TABLE ENTRIES ARE TABLE DEPENDENT.
* THEY MUST MATCH THE TABLE ARGUMENT AND RESULT DATA
* CODED IN THE TABLE DEFINITION.
*
  05  DUMMY-ENTRY.
      10  DUMMY-ARGUMENT  PIC X(4).
      10  DUMMY-RESULT    PIC X(30).

Figure 3 Sample COBOL Program Using CLACCESS (Page 2 of 5)
* 05 TABLE-ENTRY OCCURS 100 TIMES
   ASCENDING KEY ARGUMENT
   INDEXED BY IX1.
   10 ARGUMENT PIC X(4).
   10 RESULT PIC X(30).
*
PROCEDURE DIVISION.
*
000-MAINLINE.
   MOVE 'OPEN ' TO CLAFUNC.
   MOVE 65536 TO CLAWORK-SIZE.
   MOVE 'LIBDD ' TO DDNAME.
   CALL 'CLACCESS' USING CLAFUNC, RELEASE-INFO, CLAWORK-SIZE,
              DDNAME, ERROR-CODE, REASON-CODE.
   IF ERROR-CODE IS NOT = 0
      GO TO 900-ERROR.
   DISPLAY 'COMLIB RELEASE INFO: ' RELEASE-INFO.
   MOVE 'GETATTR ' TO CLAFUNC.
   MOVE 'VENDORS ' TO CLATABLE.
   CALL 'CLACCESS' USING CLAFUNC, CLATABLE, SAMPLE-TABLE,
              ERROR-CODE, REASON-CODE.
   IF ERROR-CODE IS NOT = 0
      GO TO 900-ERROR.
   DISPLAY 'TABLE ATTRIBUTES FOLLOW:'.
   DISPLAY ' TABLE NAME: ' TABLE-NAME.
   DISPLAY ' TABLE SIZE: ' TABLE-SIZE.
   DISPLAY ' NUM OF ENTRIES: ' NUM-ENTRIES.
   DISPLAY ' ENTRY SIZE: ' ENTRY-SIZE.

Figure 3  Sample COBOL Program Using CLACCESS (Page 3 of 5)
DISPLAY '   TABLE TYPE:     ' TABLE-TYPE.
DISPLAY '   ARGUMENT TYPE:  ' ARG-TYPE.
DISPLAY '   ARGUMENT SIZE:  ' ARG-SIZE.
DISPLAY '   ARGUMENT DEC:   ' ARG-DECIMALS.
DISPLAY '   RESULT TYPE:    ' RES-TYPE.
DISPLAY '   RESULT SIZE:    ' RES-SIZE.
DISPLAY '   RESULT DEC:     ' RES-DECIMALS.
DISPLAY '  '.

* IF NUM-ENTRIES > 100
  DISPLAY 'NUMBER OF TABLE ENTRIES EXCEEDS LIMIT OF 100'
  MOVE 16 TO RETURN-CODE
  STOP RUN.
*
  MOVE 'GETTABLE' TO CLAFUNC.
  CALL 'CLACCESS' USING CLAFUNC, CLATABLE, SAMPLE-TABLE,
       ERROR-CODE, REASON-CODE.
  IF ERROR-CODE IS NOT = 0
    GO TO 900-ERROR.
*
  MOVE 'CLOSE   ' TO CLAFUNC.
  CALL 'CLACCESS' USING CLAFUNC,
       ERROR-CODE, REASON-CODE.
  IF ERROR-CODE IS NOT = 0
    GO TO 900-ERROR.
*
  DISPLAY 'TABLE DATA FOLLOWS:'.
  PERFORM DISPLAY-DATA
    VARYING IX1 FROM 1 BY 1 UNTIL IX1 > NUM-ENTRIES.

Figure 3  Sample COBOL Program Using CLACCESS (Page 4 of 5)
Figure 4 displays sample JCL for compiling, linking, and executing the program in Figure 3. The link-edit step must include the data set containing the CLACCESS module in its SYSLIB concatenation. The go (application execution) step must include the COMLIB executable program library in the STEPLIB (or JOBLIB) concatenation and the

```
GOBACK.
*
900-ERROR.
  IF ERROR-CODE = 12
    DISPLAY 'ERROR RETURN - RETURN-CODE = ', ERROR-CODE,
    ', REASON-CODE = ', MSG-ID
  ELSE
    DISPLAY 'ERROR RETURN - RETURN-CODE = ', ERROR-CODE,
    ', REASON-CODE = ', REASON-CODE.
    MOVE ERROR-CODE TO RETURN-CODE.
    STOP RUN.
  *
  DISPLAY-DATA.
    DISPLAY ' ', ARGUMENT(IX1), ' ', RESULT(IX1).
```

Figure 3  Sample COBOL Program Using CLACCESS (Page 5 of 5)

Figure 4 displays sample JCL for compiling, linking, and executing the program in Figure 3. The link-edit step must include the data set containing the CLACCESS module in its SYSLIB concatenation. The go (application execution) step must include the COMLIB executable program library in the STEPLIB (or JOBLIB) concatenation and the
LIBDD DD statement for the common library where the table resides. If the LIBDD DD statement is not provided in this example, CLACCESS sets the return code to 12 and the reason code to `CG00'.

```
//jobname  JOB ...
//        EXEC IGYWCLG
//COBOL.SYSIN   DD DSN=your.source.program(member),DISP=SHR
//LKD.SYSLIB    DD
//               DD DSN=your.claccess.loadlib,DISP=SHR
//GO.STEPLIB    DD
//               DD DSN=your.comlib.loadlib,DISP=SHR
//LIBDD        DD DSN=your.common.library.dataset,DISP=SHR
```

**Figure 4** Sample JCL for Compile, Link, and Execution

When using COMLIB Release 3.1 and referencing a BDAM common library, you must use the BDAM version of the COMLIB software in your STEPLIB or JOBLIB concatenation. Likewise, when referencing a VSAM common library, you must use the VSAM version of the COMLIB software in your STEPLIB or JOBLIB concatenation. If you are using COMLIB Release 4.0, the same version can be used to access both BDAM and VSAM common libraries.
Chapter 5

PDS User I/O Routine

The M4PDSIN User I/O routine allows VISION:Builder applications to read a Partition Data Set (PDS) as a sequential file. Each member of the PDS will be read in sequence until all members have been read. A special delimiter record will be inserted as the first record of every member. This record will contain the text `MEMBER=member-name` beginning in position 1 with the rest of the record set to all blanks. In addition, the first eight position of the M4OWN flag field will contain the member name for the duration of all records belonging to the member.

A File Definition must be prepared that corresponds to the data contained in the PDS. The file record attributes (such as format, record size, block size) in the file definition must correspond to the related data set attributes.

The User I/O routine is used by specifying the name M4PDSIN in the User I/O Module Name entry of the RF statement for the associated input file as shown in the following example for an M4OLD file:

```
TESTRUN RFALL80F  SM4OLD                              M4PDSIN
```
M4PDSIN may cause a VISION:Builder application to terminate with message MK4UI00. The possible error codes are:

- **FF000001** Unexpected function code received from VISION:Builder.
- **FF000002** Internal M4PDSIN program error, bad state.
- **FF000003** Open failed. Possible missing DD statement.
- **FF000004** Incompatible record format between data set and file definition.
- **FF000005** Incompatible record size between data set and file definition.
- **FF000006** Incompatible access mode or processing mode. Typically, usage is not as a sequential input file.

The following sample application illustrates the use of M4PDSIN to read a partition data set containing 80-byte records using M4OLD and output all of the members as single sequential file using M4SUBF1:

```
(FDGREAD ) (RC) (ALL80F ) (S) (0) (Y,#,T) 1
(FDGREAD ) (SF) (ALL80F ) (S) (M4OLD ) 2
(FDGREAD ) (SF) (SUBOUT1 ) (S) (M4SUBF1 ) 3
```

**Figure 5** Sample M4PDSIN Application (Page 1 of 2)
Note that the source code for M4PDSIN is supplied on the VISION:Builder input tape. This is intended to serve two purposes. The first purpose is to provide users with a model of a working User I/O routine to better understand the interface to VISION:Builder. Second, the source code is provided so that users may modify or adapt the existing M4PDSIN program to meet particular user requirements.

Figure 5 Sample M4PDSIN Application (Page 2 of 2)
One possible variation to M4PDSIN is to provide the current member name using the OWN flag field rather than inserting the delimiter record as noted above. Another variation is to return the PDS member as one hierarchical record with the directory information as the root segment and the member records as child segments. This approach would however require the file definition record size to be large enough to contain the entire member.
Contacting Sterling Software

- For customers in the U.S. and Canada:

<table>
<thead>
<tr>
<th>Technical Support</th>
<th>Product Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone: 800.328.7463 or 818.703.8599</td>
<td>Phone: 800.587.1002 or 818.716.1616</td>
</tr>
<tr>
<td>FAX: 818.704.1102</td>
<td>FAX: 818.716.5998</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:IMD.Customer.Support@sterling.com">IMD.Customer.Support@sterling.com</a></td>
<td>Web: <a href="http://www.sterling.com/vision">www.sterling.com/vision</a></td>
</tr>
</tbody>
</table>

Telephone support available from 6:00 a.m. to 5:30 p.m. Pacific Time, Monday through Friday, with 24-hour emergency support provided seven days a week.

- For customers outside the U.S. and Canada:

<table>
<thead>
<tr>
<th>Technical Support</th>
<th>Product Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone: 44-1932-587-000</td>
<td>Phone: 44-1932-587-000</td>
</tr>
<tr>
<td>FAX: 44-1932-587-242</td>
<td>Web: <a href="http://www.sterling.com/vision">www.sterling.com/vision</a></td>
</tr>
<tr>
<td>E-mail: <a href="mailto:IBIDProduct-Services@sterling.com">IBIDProduct-Services@sterling.com</a></td>
<td></td>
</tr>
</tbody>
</table>

- To contact Consulting Services and for Regional and Customized On-Location Training Courses, call 800.998.0550.

- To receive information about the annual Sterling Software Worldwide Customer Conference, call 800.587.1003.
A
API description 15
API for M4TKCSV 24
areaptr 42
attrptr 39

B
bit to byte 5
bit-field 4, 5
bufoffset 20
bufptr 20
bufsize 20
byte to bit 3
byte-field 4, 5

C
CLACCESS 34
CLOSE 43
functions 35
GETATTR 38
GETTABLE 41
OPEN 36
register conventions 35
CLOSE 43
cnvlength 21
COMLIB
close connection 43
open connection 36
retrieve attributes 38
retrieve table 41
comma separated values 12
convert field to CSV format 16
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D</strong></td>
</tr>
<tr>
<td>ddname 37</td>
</tr>
<tr>
<td><strong>F</strong></td>
</tr>
<tr>
<td>fcncode 17, 24</td>
</tr>
<tr>
<td>fldlength 18</td>
</tr>
<tr>
<td>fldsacle 19</td>
</tr>
<tr>
<td>fldtype 17</td>
</tr>
<tr>
<td>fldvalue 19</td>
</tr>
<tr>
<td>from-field 10</td>
</tr>
<tr>
<td>function 36, 38, 41, 43</td>
</tr>
<tr>
<td><strong>G</strong></td>
</tr>
<tr>
<td>GETATTR 38</td>
</tr>
<tr>
<td>GETTABLE 41</td>
</tr>
<tr>
<td>Gregorian to Julian 8</td>
</tr>
<tr>
<td>Gregorian-date 6, 8</td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td>hexadecimal 9</td>
</tr>
<tr>
<td><strong>J</strong></td>
</tr>
<tr>
<td>M4TKBIT 3</td>
</tr>
<tr>
<td>M4TBYTE 5</td>
</tr>
<tr>
<td>M4TKCSV 12</td>
</tr>
<tr>
<td>terminating 24</td>
</tr>
<tr>
<td>M4TKDATG 6</td>
</tr>
<tr>
<td>M4TKHEXA 9</td>
</tr>
<tr>
<td>MK4TKDATJ 8</td>
</tr>
<tr>
<td><strong>O</strong></td>
</tr>
<tr>
<td>OPEN 36</td>
</tr>
<tr>
<td><strong>P</strong></td>
</tr>
<tr>
<td>pattern 7, 8</td>
</tr>
</tbody>
</table>
Index

**R**
reascode 21, 37, 39, 42, 43
retcode 21, 37, 39, 42, 43
ribptr 36

**S**
subroutines 3

**T**
tablenam 38, 41
TERM 15
to-field 10
TOCSV 15
translate
  bit to byte 5
  byte to bit 3
  comma separated values 12
  Gregorian to Julian 8
  hexadecimal 9
  Julian to Gregorian 6

**U**
usersymbols 20

**W**
worksize 37