IBM Communications Server for AIX or Linux

Common Service Verbs Programmer's Guide

V64

SC23-8589-00

IBM Communications Server for AIX or Linux

Common Service Verbs Programmer's Guide

V64

SC23-8589-00

Note:
Before using this information and the product it supports, be sure to read the general information under Appendix B, "Notices," on page 45.

First Edition (May 2009)

This edition applies to Version 6 Release 4 of Communications Server for AIX and Linux (5765-E51 and 5724-i33) and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. You may send your comments to the following address:
International Business Machines Corporation
Attn: Communications Server for AIX/Linux Information Development
Department AKCA, Building 501
P.O. Box 12195, 3039 Cornwallis Road
Research Triangle Park, North Carolina
27709-2195
U.S.A.

You can send us comments electronically by using one of the following methods:
- Fax (USA and Canada):
  - 1+919-254-4028
  - Send the fax to "Attn: Communications Server for AIX/Linux Information Development"
- Internet e-mail:
  - comsvrcf@us.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Tables

About This Book

This book is a guide for using the IBM Communications Server for AIX or Linux Common Service Verbs (CSVs) in C-language application programs.

This manual applies to IBM Communications Server, which is an IBM®software product that enables a server running AIX®, or a computer running Linux, to exchange information with other nodes on an SNA network.

There are three different installation variants of IBM Communications Server, depending on the hardware on which it operates:

IBM Communications Server for AIX (CS/AIX)
IBM Communications Server for AIX operates on a server running AIX Version 5.2, 5.3 or 6.1 base operating system.

IBM Communications Server for Linux (Communications Server for Linux)
IBM Communications Server for Linux, program product number 5724–i33, operates on the following:

- 32–bit Intel workstations running Linux (i686)
- 64–bit AMD64/Intel EM64T workstations running Linux (x86_64)
- IBM pSeries computers running Linux (ppc64)

IBM Communications Server for Linux on System z (Communications Server for Linux on System z)
IBM Communications Server for Linux on System z, program product number 5724–i34, operates on System z mainframes running Linux for System z (s390 or s390x).

In this book, the name Communications Server is used to indicate any of these variants, and the term "Communications Server computer" is used to indicate any type of computer running Communications Server, except where differences are described explicitly.

This book applies to V6.4 of Communications Server.

Who Should Use This Book

This book is intended for experienced C programmers who write Systems Network Architecture (SNA) transaction programs for systems with Communications Server. Programmers may or may not have prior experience with SNA or the communication facilities of Communications Server.

Application programmers design and code transaction and application programs that use the Communications Server programming interfaces to send and receive data over an SNA network. They should be thoroughly familiar with SNA, the remote program with which the transaction or application program communicates, and the AIX / Linux operating system programming and operating environments.

How to Use This Book

This section explains how information is organized and presented in this book.

Organization of This Book

This book is organized as follows:

- Chapter 1, "Concepts," on page 1, summarizes Common Service Verbs and explains how to use them in C programs.
- Chapter 2, "Common Service Verbs Reference," on page 15, describes each verb in detail. Each description includes the verb's purpose, verb control block (VCB), and supplied and returned parameters.
- Appendix A, "Code Pages," on page 41, lists the ASCII and EBCDIC code pages that are supported by the GET_CP_CONVERT_TABLE verb.

Typographic Conventions

Table 1 shows the typographic styles used in this document.

Table 1. Typographic Conventions

Special Element	Sample of Typography
Document title	*IBM Communications Server for AIX or Linux APPC Programmer's Guide*
File or path name	**acssvcc.h**
Program or application	**snamsgf**
Command or AIX / Linux utility	**kill**
Option or flag	**-I**
Parameter or Motif field	*wVersionRequired*; *primary_rc*
Literal value or selection that the user can enter (including default values)	`0x0001`; `0`
Constant or signal	`SIGPOLL`; `SV_ASCII_TO_EBCDIC`
Return value	`WCSVVERNOTSUPPORTED`; `0`; `AP_OK`
Variable representing a supplied value	*programname*
Environment variable	`LD_RUN_PATH`
Programming verb	CONVERT; TRANSFER_MS_DATA
Function, call, or entry point	`ACSSVC_P`
Data structure	`WCSVDATA`
Hexadecimal value	`0x20`

Graphic Conventions

AIX, LINUX

This symbol is used to indicate the start of a section of text that applies only to the AIX or Linux operating system. It applies to AIX / Linux servers and to the IBM Remote API Client running on AIX, Linux, Linux for pSeries or Linux for System z.

WINDOWS

This symbol is used to indicate the start of a section of text that applies to the IBM Remote API Client on Windows.

This symbol indicates the end of a section of operating system specific text. The information following this symbol applies regardless of the operating system.

Where to Find More Information

See the bibliography for other books in the Communications Server library, as well as books that contain additional information about topics related to SNA and AIX / Linux workstations.

Chapter 1. Concepts

This chapter provides information that you need to know when developing CSV application programs. It contains the following information:

- Summary of Common Service Verbs
- CSV entry points
- Issuing a verb

AIX, LINUX

- AIX or Linux considerations

WINDOWS

- Windows considerations

- Writing portable applications

Summary of Common Service Verbs

This section briefly describes Common Service Verbs. Chapter 2, "Common Service Verbs Reference," on page 15 contains a detailed description of each verb.

CONVERT
Converts a character string from ASCII to EBCDIC or from EBCDIC to ASCII.

COPY_TRACE_TO_FILE

AIX, LINUX

Copies the current contents of the trace file (or files) to another file for storage.

DEFINE_TRACE
Enables or disables tracing for specific APIs.

GET_CP_CONVERT_TABLE
Creates and returns a 256-byte conversion table to translate character strings from a source code page to a target code page.

LOG_MESSAGE

AIX, LINUX

Takes a message from a message file, adds specified data to it, and records the message in the error log file or the audit log file.

TRANSFER_MS_DATA

WINDOWS

Builds a Systems Network Architecture (SNA) request unit (RU) containing Network Management Vector Transport (NMVT) data. The verb can send the NMVT data to NetView for centralized problem diagnosis and resolution. The data can also be logged in the local error log file.

CSV Entry Points: AIX or Linux Systems

AIX, LINUX

A C program calls Common Service Verbs through the following entry point:

```
void ACSSVC_C (
               void * vcbptr
              );
```

The only parameter passed to the function is the address of a verb control block (VCB). The VCB is a structure made up of variables that identify the verb to be executed, supply information to be used by the verb, and contain information returned by the verb when execution is complete. Each verb has its own VCB structure, which is declared in the header file **/usr/include/sna/acssvcc.h** (AIX) or **/opt/ibm/sna/include/acssvcc.h** (Linux) delivered with Communications Server. Use `#include` to include this file in any application program that issues Common Service Verbs.

Note: The CSV VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

For compatibility with other CSV implementations, Communications Server also provides the entry points `ACSSVC_P` and `ACSSVC`, which can be used in the same way as `ACSSVC_C`.

The entry points are defined in the CSV header file **acssvcc.h**.

CSV Entry Points: Windows

WINDOWS

A Windows application accesses CSV using the following functions:

ACSSVC_C
Issues a verb. The verb blocks; that is, the application's thread is suspended until CSV has finished processing the verb and returned the results. This has the same effect as `WinCSV`.

WinCSVStartup
Registers the application as a Windows CSV user, and determines whether the CSV software supports the level of function required by the application.

WinCSV Issues a verb. The verb blocks; that is, the application's thread is suspended until CSV has finished processing the verb and returned the results. This has the same effect as `ACSSVC_C`.

WinAsyncCSV
Issues a verb. With the exception of TRANSFER_MS_DATA, the verb blocks; processing is the same as for the `WinCSV` entry point. The TRANSFER_MS_DATA verb normally completes asynchronously and does not block; CSV indicates the completion by posting a message to the application window.

WinCSVCleanup
Unregisters the application when it has finished using CSV.

GetCsvReturnCode
Generates a printable character string for the primary and secondary return codes obtained on a CSV verb.

The entry points are defined in the Windows CSV header file **wincsv.h**. This file is installed in the subdirectory **\sdk** for 32–bit applications, or **\sdk64** for 64–bit applications, within the directory where you installed the Windows Client software.

The application must call `WinCSVStartup` before attempting to issue any verbs using the `WinCSV` or `WinAsyncCSV` calls. It then issues verbs using either `WinAsyncCSV` (asynchronous) or `WinCSV` (synchronous). If a verb returns with return codes that indicate an error, the application can use `GetCsvReturnCode` to obtain a text string representation of these return codes, which can be used to generate standard error messages.

When the application has finished issuing verbs using the `WinCSV` or `WinAsyncCSV` calls, it must call `WinCSVCleanup` before terminating; it must not attempt to issue any more verbs after calling `WinCSVCleanup`.

The following sections describe these functions.

ACSSVC_C

The application uses this function to issue a verb. The verb blocks; that is, the application's thread is suspended until CSV has finished processing the verb and returned the results.

For compatibility with other CSV implementations, Communications Server also provides the entry points ACSSVC_P and ACSSVC, which can be used in the same way as ACSSVC_C. The entry points are defined in the CSV header file **sdk/wincsv.h**.

Function Call

```
void ASCCVC_C (
              void * vcbptr
             )
```

Supplied Parameters

The only parameter passed to the function is the address of a verb control block (VCB). The VCB is a structure made up of variables that identify the verb to be executed, supply information to be used by the verb, and contain information returned by the verb when execution is complete. Each verb has its own VCB structure, which is declared in the header file **sdk/wincsv.h** delivered with the Remote API Client on Windows. Use #include to include this file in any application program that issues Common Service Verbs.

Note: The CSV VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use memset:

memset(vcb, 0, sizeof(vcb));

Returned Values

The function does not return a value.

WinCSVStartup

The application uses this function to register as a Windows CSV user, and to determine whether the CSV software supports the Windows CSV version that it requires.

Function Call

```
int WINAPI WinCSVStartup (
                           WORD              wVersionRequired,
                           WCSVDATA far *    lpData
                         );

typedef struct
{
 WORD       wVersion;
 char       szDescription[128];
} WCSVDATA;
```

Supplied Parameters

The supplied parameter is:

wVersionRequired
The version of Windows CSV that the application requires. Communications Server supports version 1.0.

The low-order byte of this parameter specifies the major version number, and the high-order byte specifies the minor version number. For example:

Version	wVersionRequired
1.0	0x0001
1.1	0x0101
2.0	0x0002

If the application can use more than one version, it should specify the highest version that it can use.

Returned Values

The return value from the function is one of the following:

0 (zero)
The application was registered successfully, and the Windows CSV software supports either the version number specified by the application or a lower version. The application should check the version number in the WCSVDATA structure to ensure that it is high enough.

WCSVVERNOTSUPPORTED
The version number specified by the application was lower than the lowest version supported by the Windows CSV software. The application was not registered.

WCSVSYSNOTREADY
The Remote API Client software has not been started, or the local node is not active. The application was not registered.

If the return value from WinCSVStartup is zero, the WCSVDATA structure contains information about the support provided by the Windows CSV software. If the return value is nonzero, the contents of this structure are undefined and the application should not check them. The parameters in this structure are as follows:

wVersion
The Windows CSV version number that the software supports, in the same format as the *wVersionRequired* parameter (defined previously). Communications Server supports version 1.0.

If the software supports the version number requested by the application, this parameter is set to the same value as the *wVersionRequired* parameter; otherwise it is set to the highest version that the software supports, which will be lower than the version number supplied by the application. The application must check the returned value and take action as follows:

- If the returned version number is the same as the requested version number, the application can use this Windows CSV implementation.
- If the returned version number is lower than the requested version number, the application can use this Windows CSV implementation but must not attempt to use features that are not supported by the returned version number. If it cannot do this because it requires features not available in the lower version, it should fail its initialization and not attempt to issue any CSV verbs.

szDescription
A text string describing the Windows CSV software.

WinCSV

The application uses this function to issue a verb, which blocks until verb processing is completed.

Function Call

```
void WINAPI WinCSV (
                    long  vcbptr
                   );
```

Supplied Parameters

The only parameter to the function is a pointer to the VCB structure for the verb. This is defined as a long integer, and so needs to be cast from a pointer to a long integer. For the definition of the VCB structure for each verb, see Chapter 2, "Common Service Verbs Reference," on page 15.

Note: The CSV VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

Returned Values

The function does not return a value. When the call returns, the application should check the *primary_rc* and *secondary_rc* parameters in the VCB structure to determine whether the verb completed successfully. For information about the parameters returned in the VCB structure, see the descriptions of individual verbs in Chapter 2, "Common Service Verbs Reference," on page 15.

WinAsyncCSV

The application uses this function to issue a verb.

For TRANSFER_MS_DATA, the verb may complete asynchronously; CSV will indicate the completion by posting a message to the application's window handle. All other verbs complete synchronously.

Before using the `WinAsyncCSV` call for the first time, the application must use RegisterWindowMessage to obtain the message identifier that CSV will use for messages indicating asynchronous verb completion. For more information, see "Windows Considerations" on page 12.

Function Call

```
HANDLE WINAPI WinAsyncCSV (
                           HWND  hWnd,
                           long  vcbptr
                          );
```

Supplied Parameters

The supplied parameters are:

hWnd A window handle that CSV will use to post a message indicating asynchronous verb completion.

vcbptr A pointer to the VCB structure for the verb. This parameter is defined as a long integer, and so needs to be cast from a pointer to a long integer. For more information about the VCB structure and on its usage for individual verbs, see Chapter 2, "Common Service Verbs Reference," on page 15.

Note: The CSV VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

```
memset(vcb, 0, sizeof(vcb));
```

Returned Values: TRANSFER_MS_DATA

If the function was successful, the return value is a handle. When the verb later completes, CSV uses this handle as an identifier in the message passed to the application's window procedure (for more information, see "Usage").

A return value of `0` indicates that the function call was not accepted.

Returned Values: Other Verbs

For all verbs other than TRANSFER_MS_DATA, the function operates in the same way as the `WinCSV` entry point (described in the previous section), and does not return a value. When the call returns, the application should check the *primary_rc* and *secondary_rc* parameters in the VCB structure to determine whether the verb completed successfully.

Usage

Before using `WinAsyncCSV` for the first time, the application must use the `RegisterWindowMessage` call to obtain the message identifier that CSV will use for messages indicating asynchronous verb completion. `RegisterWindowMessage` is a standard Windows function call, not specific to CSV; refer to your Windows documentation for more information about the function. (There is no need to issue the call again before subsequent verbs; the returned value will be the same for all calls issued by the application.)

The application must pass the string "WinAsyncCSV" to the function; the returned value is a message identifier, as described below.

Each time a verb that was issued using the `WinAsyncCSV` entry point completes asynchronously, CSV posts a message to the window handle specified on the `WinAsyncCSV` call. The format of the message is as follows:

- The message identifier is the value returned from the RegisterWindowMessage call.
- The *lParam* argument contains the address of the VCB that was supplied to the original WinAsyncCSV call; the application can use this address to access the returned parameters in the VCB structure.
- The *wParam* argument contains the handle that was returned to the original WinAsyncCSV call.

WinCSVCleanup

The application uses this function to unregister as a Windows CSV user, after it has finished issuing verbs.

Function Call

```
BOOL WINAPI WinCSVCleanup (void);
```

Supplied Parameters

No parameters are supplied for this function.

Returned Values

The return value from the function is one of the following:

TRUE The application was unregistered successfully.

FALSE An error occurred during processing of the call, and the application was not unregistered. Check the log files for messages indicating the cause of the failure.

GetCsvReturnCode

This call returns a printable character string interpreting the return codes from a supplied VCB. The string can be used to generate application error messages for return codes other than AP_OK.

This call is designed to provide strings for display to the end user of an application. For return codes indicating configuration problems or user errors (for example if a required component is not configured or not started), the string should provide sufficient information to help the user correct the problem. For return codes indicating application errors (for example, if the application has issued a verb that is not valid or failed to supply a required parameter), the user will not generally be able to correct the problem; in these cases, the string may be meaningful only to an application developer.

Function Call

```
int WINAPI GetCsvReturnCode (
                             struct svc_hdr FAR * vcbbptr,
                             unsigned int         buffer_length,
                             unsigned char FAR *  buffer_addr
                            );

typedef struct svc_hdr
  {
  unsigned short     opcode;        /* Verb identifying operation code.      */
  unsigned char      opext;         /* Verb extension code - reserved.       */
  unsigned char      reserv2;       /* Reserved.                             */
  unsigned short     primary_rc;    /* Primary return code from verb.        */
  unsigned long      secondary_rc;  /* Secondary (qualifying) return code.   */
```

Supplied Parameters

The supplied parameters are:

vcbptr A pointer to the VCB structure for the verb. For more information about the VCB structure and on its usage for individual verbs, see Chapter 2, "Common Service Verbs Reference," on page 15.

buffer_length
The length (in bytes) of the buffer supplied by the application to hold the returned data string. The recommended length is 256 bytes.

buffer_addr
The address of the buffer supplied by the application to hold the returned data string.

Returned Values

The return value from the function is one of the following:

0x00000000
The function completed successfully.

0x20000001
CSV could not read from the supplied VCB, or could not write to the supplied data buffer.

0x20000002
The supplied data buffer is too small to hold the returned character string.

0x20000003
The dynamic link library (**CSVSTR32.DLL**) which generates the returned character strings for this function, could not be loaded.

If the return value is `0x00000000`, the returned character string is in the buffer identified by the *buffer_addr* parameter. This string is terminated by a null character (binary zero), but does not include a trailing new-line (`\n`) character.

Issuing a Verb

The major steps in issuing a Common Service Verb follow. Each step is illustrated by sample code pertaining to the CONVERT verb; for more information about this verb, see Chapter 2, "Common Service Verbs Reference," on page 15.

1. Create a structure variable from the VCB structure that applies to the verb to be issued.

   AIX, LINUX

   ```
   #include <acssvcc.h>
        .
        .
        .
   struct convert  conv_block;
   ```

   WINDOWS

   ```
   #include <wincsv.h>
        .
        .
        .
   struct convert  conv_block;
   ```

The VCB structures are declared in the CSV header file **acssvcc.h**(for AIX / Linux) or **wincsv.h** (for Windows). One of these structures is named *convert*.

2. Clear (set to zero) the variables within the structure.

```
memset(&conv_block, 0, sizeof( conv_block ) );
```

This step is important to ensure that the application can later be upgraded to work with future CSV versions (which can use fields that are reserved in the current version). It also helps in debugging and interpreting trace data.

3. Assign values to the required VCB variables.

```
conv_block.opcode = SV_CONVERT;
conv_block.direction = SV_ASCII_TO_EBCDIC;
conv_block.char_set = SV_AE;
conv_block.len = sizeof(tpstart_name);
conv_block.source = (unsigned char *) tpstart_name;
conv_block.target = (unsigned char *) tpstart.tp_name;
```

The fields SV_CONVERT, SV_ASCII_TO_EBCDIC, and SV_AE are symbolic constants representing integers. These constants are defined in the CSV header file.

The character array *tpstart_name* contains an ASCII string to be converted to EBCDIC and placed in the character array *tpstart.tp_name*.

4. Invoke the verb. The only parameter is a pointer to the structure containing the VCB for the verb.

AIX, LINUX

```
ACSSVC_C ((char *)&conv_block);
```

For compatibility with other CSV implementations, the entry points `ACSSVC_P` or `ACSSVC` can be used instead of `ACSSVC_C`.

WINDOWS

```
WinCSV ( (long) ( (char far *) &conv_block ) );
```

Use the values returned by the verb.

```
if (conv_block.primary_rc == SV_OK)
{
     /* other statements   */
          .
          .
          .
}
```

AIX or Linux Considerations

AIX, LINUX

This section summarizes the information you need to consider when developing applications for use in the AIX or Linux environment.

CSV Header File

The header file to be used with CSV applications is **acssvcc.h**. This file contains the definitions of the CSV entry points and verb control blocks. It also includes the common interface header file **values_c.h**, which contains the constants defined for supplied and returned parameter values at the CSV interface. Both of these files are stored in **/usr/include/sna** (AIX) or **/opt/ibm/sna/include** (Linux).

Multithreaded Applications

The Communications Server CSV library supports multithreaded applications. The only restrictions are as follows:

- Only one verb can be outstanding at any time. A verb will fail with the return codes `AP_STATE_CHECK` and `AP_SYNC_PENDING` if another verb is in progress.
- The application must perform any required clean-up processing before a thread terminates. The CSV library does not maintain any correlation between threads and verb usage, and will not perform this processing automatically when a thread terminates.

Do not attempt to use multithreaded applications with a version of the library that does not support DCE threads.

Compiling and Linking the CSV Application

AIX Applications

To compile and link 32–bit applications, use the following options:

-bimport:/usr/lib/sna/csv_r.exp -I /usr/include/sna

To compile and link 64–bit applications, use the following options:

-bimport:/usr/lib/sna/csv_r64_5.exp -I /usr/include/sna

Linux Applications

Before compiling and linking a CSV application, specify the directory where shared libraries are stored, so that the application can find them at run time. To do this, set the environment variable `LD_RUN_PATH` to **/opt/ibm/sna/lib**, or to **/opt/ibm/sna/lib64** if you are compiling a 64–bit application.

To compile and link 32–bit applications, use the following options:

-I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lcsv -lsna_r -lpthread -lpLiS

To compile and link 64–bit applications, use the following options:

-I /opt/ibm/sna/include -L /opt/ibm/sna/lib64 -lcsv -lsna_r -lpthread -lpLiS

The option `-lpLiS` is required only if you will be running the application on a Communications Server server; you do not need to use it if you are building the application on an IBM Remote API Client and it will run only on the client. As an alternative to using this option, you can set the the environment variable `LD_PRELOAD` to **/usr/lib/libpLiS.so** before compiling and linking the application.

Windows Considerations

WINDOWS

This section summarizes processing considerations you need to be aware of when developing applications on a Windows client.

Compiling and Linking a CSV Application

This section provides information about compiling and linking CSV programs on Windows.

Compiler Options for Structure Packing

The VCB structures for CSV are not packed. Do not use compiler options that change this packing method.

DWORD parameters are on DWORD boundaries, *WORD* parameters are on WORD boundaries, and *BYTE* parameters are on BYTE boundaries.

Header Files

The CSV header file to be included in Windows CSV applications is named **wincsv.h**. This file is installed in the subdirectory **\sdk** for 32–bit applications, or **\sdk64** for 64–bit applications, within the directory where you installed the Remote API Client on Windows software.

Load-Time Linking

To link the TP to CSV at load time, link the TP to the API library file **\sdk\wincsv32.lib** for 32–bit applications, or **\sdk64\wincsv32.lib** for 64–bit applications.

Run-Time Linking

To link the TP to CSV at run-time, include the following calls in the TP:

- `LoadLibrary` to load the CSV dynamic link library **wincsv32.dll**
- `GetProcAddress` to specify CSV on each of the CSV entry points required (such as `WinAsyncCSV`, `WinCSVStartup`, and `WinCSVCleanup`)
- `FreeLibrary` when the library is no longer required

Writing Portable Applications

The following guidelines are provided for writing Communications Server applications so that they will be portable to other environments:

- Include the CSV header file without any path name prefix. Use include options on the compiler to locate the file (refer to the appropriate section for your operating system, earlier in this chapter) This enables the application to be used in an environment with a different file system.
- Use the symbolic constant names for parameter values and return codes, not the numeric values shown in the header file; this ensures that the correct value will be used regardless of the way these values are stored in memory.

- Include a check for return codes other than those applicable to your current operating system (for example using a “default” case in a switch statement), and provide appropriate diagnostics.
- Ensure that any parameters shown as reserved are set to `0`.

Chapter 2. Common Service Verbs Reference

This chapter contains a description of each of the Common Service Verbs. The following information is provided for each verb:

- Definition of the verb.
- Structure defining the verb control block (VCB) used by the verb. The structure is declared in the CSV header file.
- Parameters (VCB fields) supplied for and returned by the verb. For each parameter, the following information is provided:
  - Description
  - Possible values
  - Additional information
- Additional information describing the use of the verb.

Most parameters supplied with and returned by Common Service Verbs are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants defined in the header file **values_c.h**, which is included by the CSV header file **acssvcc.h**. For example, the *opcode* (operation code) parameter for the CONVERT verb is the hexadecimal value represented by the symbolic constant `SV_CONVERT`. The file **values_c.h** also includes definitions of parameter types such as `AP_UINT16` that are used in the CSV VCBs.

It is important that you use the symbolic constant and not the hexadecimal value when setting values for supplied parameters, or when testing values of returned parameters. This is because different systems store these values differently in memory, so the value shown may not be in the format recognized by your system.

If you are writing applications for use in other environments as well as Communications Server, see "Writing Portable Applications" on page 12.

Note: The CSV VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

```
memset(vcb, 0, sizeof(vcb));
```

CONVERT

The CONVERT verb translates an ASCII character string to EBCDIC or an EBCDIC character string to ASCII.

The string to be converted is called the source string. The converted string is called the target string.

VCB Structure

AIX, LINUX

```
typedef struct convert
  {
  AP_UINT16        opcode;          /* Verb identifying operation code.      */
  unsigned char    opext;           /* Verb extension code - reserved.       */
  unsigned char    reserv2;         /* Reserved.                             */
  AP_UINT16        primary_rc;      /* Primary return code from verb.        */
  AP_UINT32        secondary_rc;    /* Secondary (qualifying) return code.   */
  unsigned char    direction;       /* Direction of conversion - ASCII to    */
                                    /*  EBCDIC or vice-versa.                */
  unsigned char    char_set;        /* Character set to use for the          */
                                    /*  conversion A, AE, or user-defined G. */
  AP_UINT16        len;             /* Length of string to be converted.     */
  unsigned char    *source;         /* Pointer to string to be converted.    */
  unsigned char    *target;         /* Address to put converted string at.   */
  };
```

WINDOWS

```
typedef struct convert
  {
  unsigned short     opcode;        /* Verb identifying operation code.      */
  unsigned char      opext;         /* Verb extension code - reserved.       */
  unsigned char      reserv2;       /* Reserved.                             */
  unsigned short     primary_rc;    /* Primary return code from verb.        */
  unsigned long      secondary_rc;  /* Secondary (qualifying) return code.   */
  unsigned char      direction;     /* Direction of conversion - ASCII to    */
                                    /*  EBCDIC or vice-versa.                */
  unsigned char      char_set;      /* Character set to use for the          */
                                    /*  conversion A, AE, or user-defined G. */
  unsigned short     len;           /* Length of string to be converted.     */
  unsigned char      *source;       /* Pointer to string to be converted.    */
  unsigned char      *target;       /* Address to put converted string at.   */
  };
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_CONVERT

direction

Possible values are:

SV_ASCII_TO_EBCDIC
Convert from ASCII to EBCDIC characters.

SV_EBCDIC_TO_ASCII
Convert from EBCDIC to ASCII characters.

char_set

Specifies which character set to use in converting the source string. Possible values are:

SV_A The type-A character set consists of the following:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, @, and space

This character set is supported by a system-supplied type-A conversion table.

The first character of the source string must be an uppercase letter or the special character $, #, or @. Spaces are allowed only in trailing positions. Lowercase letters can be supplied in positions other than the first character, but will be translated to uppercase.

SV_AE The type-AE character set consists of the following:

- Uppercase letters
- Lowercase letters
- Numerals 0–9
- Special characters $, #, @, and space

This character set is supported by a system-supplied type-AE conversion table.

The first character of the source string can be any character in the character set. Spaces are allowed only in trailing positions, unless the string consists entirely of spaces. No case conversion is performed.

SV_G The type-G character set is defined by a user-written conversion table. This table is described in detail under "Creating a Type-G Conversion Table" on page 20.

AIX, LINUX

The file containing the table must be specified by the environment variable SNATBLG; set this variable to the full path name of the file. (If the environment variable is not set or the file is not found, the system returns the SV_TABLE_ERROR return code.)

WINDOWS

For Windows clients, the file containing the table must be specified by the *CSVTBLG* value Registry Key as follows:

\\HKEY_LOCAL_MACHINE\SOFTWARE\SNA Client\SxClient\Parameters\CSV_data

The *CSVTBLG* parameter is described in the *IBM Communications Server for Linux Administration Guide* or *IBM Communications Server for AIX Administration Guide*. Set this parameter to the full path name of the file. (If the file is not found, the system returns the SV_TABLE_ERROR return code.)

len The number of characters to be converted.

source Address of buffer containing character string to be converted.

target Address of buffer to contain the converted character string.

This buffer can overlap or coincide with the buffer pointed to by the source parameter. In this case, the converted data string overwrites the source data string.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameter:

primary_rc
SV_OK

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
Possible values are:

SV_CONVERSION_ERROR
One or more characters in the source string were not found in the conversion table, or embedded spaces were found in a type-A or type-AE string. These characters or spaces were converted to nulls (0x00). The verb still executed.

SV_INVALID_CHARACTER_SET
The *char_set* parameter contained a value that is not valid.

WINDOWS

SV_INVALID_DATA_SEGMENT
The supplied source or target string extended beyond the boundary of a data segment, or the target data segment was not a read/write segment.

SV_INVALID_DIRECTION
The *direction* parameter contained a value that is not valid.

SV_INVALID_FIRST_CHARACTER
The first character of a type-A source string is not a valid value.

SV_TABLE_ERROR
The file containing the user-written type-G conversion table was not defined correctly, could not be accessed, or was not in the correct format.

AIX, LINUX

The file containing the table must be specified by the environment variable SNATBLG; set this variable to the full path name of the file.

WINDOWS

For Windows clients, the file containing the table must be specified by the *CSVTBLG* value Registry Key as follows:

\\HKEY_LOCAL_MACHINE\SOFTWARE\SNA Client\SxClient\Parameters\CSV_data

The *CSVTBLG* parameter is described in the *IBM Communications Server for AIX Administration Guide* or the *IBM Communications Server for Linux Administration Guide*. Set this parameter to the full path name of the file.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*).

WINDOWS

SV_COMM_SUBSYSTEM_NOT_LOADED
The Remote API Client software has not been started. Consult the System Administrator for corrective action.

SV_INVALID_VERB_SEGMENT
The supplied VCB extended beyond the boundary of a data segment.

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

SV_UNEXPECTED_DOS_ERROR
The operating system has encountered an error while processing the verb. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

AIX, LINUX

For the meaning of the operating system return code, see the file **/usr/include/errno.h**.

WINDOWS

For the meaning of the operating system return code, refer to your operating system documentation.

Creating a Type-G Conversion Table

You can use the GET_CP_CONVERT_TABLE verb to build a type-G, user-written conversion table. The GET_CP_CONVERT_TABLE verb is described in detail later in this chapter.

The table must be an ASCII file 32 lines long. Each line must consist of 32 hexadecimal digits, representing 16 characters. The first 16 lines (256 characters) specify the EBCDIC characters to which ASCII characters are converted; the remaining 16 lines specify the ASCII characters to which EBCDIC characters are converted.

For Communications Server, the hexadecimal digits A–F can be either uppercase or lowercase. However, you may want to make these digits uppercase to ensure compatibility with the CSV implementation provided in the IBM OS/2®Extended Edition.

The file **/usr/lib/sna/samples/snatblg.dat** (AIX) or **/opt/ibm/sna/samples/snatblg.dat** (Linux) delivered with Communications Server contains a sample type-G conversion table which converts the first 127 characters of an ASCII code page to EBCDIC. Here is a listing of that file:

```
00010203372D2E2F1605250B0C0D0E0F
101112133C3D322618193F27221D351F
405A7F7B5B6C507D4D5D5C4E6B604B61
F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F
7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D
79818283848586878889919293949596
979899A2A3A4A5A6A7A8A9C06AD0A107
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00010203000900 7F0000000B0C0D0E0F
10111213000008001819000000 1D001F
00001C00000A171B0000000000050607
00001600001E000400000000 1415001A
20000000000000000000002E3C282B00
26000000000000000000021242A293B5E
2D2F00000000000000007C2C255F3E3F
00000000000000000603A2340273D22
00616263646566676869000000000000
006A6B6C6D6E6F707172000000000000
007E737475767778797A0000005B0000
00000000000000000000000005D0000
7B41424344454647484900000000000
7D4A4B4C4D4E4F505152000000000000
5C00535455565758595A000000000000
30313233343536373839000000000000
```

COPY_TRACE_TO_FILE

AIX, LINUX

The COPY_TRACE_TO_FILE verb copies the current contents of the API trace file or files to a new file, and clears the trace files. This enables you to save a copy of the current trace data for this application. For more information about API tracing, refer to the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide*.

All API tracing on this application (for any of the Communications Server APIs) must be stopped before you issue COPY_TRACE_TO_FILE. If any tracing is active, use the DEFINE_TRACE verb to stop it before using this verb.

VCB Structure

```
typedef struct copy_trace_to_file
  {
  AP_UINT16       opcode;             /* Verb identifying operation code.    */
  unsigned char   opext;              /* Verb extension code - reserved.     */
  unsigned char   reserv2;            /* Reserved.                           */
  AP_UINT16       primary_rc;         /* Primary return code from verb.      */
  AP_UINT32       secondary_rc;       /* Secondary (qualifying) return code. */
  unsigned char   reserv3[8];         /* Reserved.                           */
  unsigned char   file_name[64];      /* File name to write to.              */
  unsigned char   file_option;        /* File options. New or overwrite.     */
  unsigned char   reserv4[12];        /* Reserved.                           */
  };
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_COPY_TRACE_TO_FILE

file_name

The name (and optionally the path) of the file to hold the trace information. This name can be up to 64 characters. If the file is not in the current directory, specify the full path; ensure that it is a valid path on any computer to which this verb is issued.

If you set the *file_option* parameter to SV_NEW, the file name specified must not be the name of an existing file.

file_option

Possible values are:

SV_NEW Create a new file with the name specified in *file_name*. An error is returned if this file already exists.

SV_OVERWRITE

Overwrite the file if it exists, or create the file if it does not exist.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameter:

primary_rc
SV_OK

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc

SV_INVALID_FILE_OPTION
The *file_option* parameter contained a value that was not valid.

State Check: If the verb does not execute successfully because of a state error, the following parameters are returned:

primary_rc
SV_STATE_CHECK

secondary_rc
Possible values:

SV_TRACE_BUFFER_EMPTY
There was no trace information to copy to file. Either the trace files were empty, or the SNATRC environment variable was not set up. This environment variable must be set up before the application is started. For information about how to control API tracing, refer to the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide*.

SV_TRACE_NOT_STOPPED
Tracing was still active when the verb was issued. Before issuing COPY_TRACE_TO_FILE, tracing for the CSV, APPC, CPI-C, and RUI interfaces must be turned off. Use DEFINE_TRACE to turn off any active tracing before issuing COPY_TRACE_TO_FILE; for more information, see "DEFINE_TRACE" on page 23.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*).

SV_FILE_ALREADY_EXISTS
You specified the value SV_NEW for the *file_option* parameter (to create a new output file), but a file with the specified name already exists.

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

SV_OUTPUT_DEVICE_FULL
There was insufficient space in the output file's disk or directory to hold the trace information. The trace files were not reset; the output file may contain some of the available trace information, but is not complete.

SV_UNEXPECTED_DOS_ERROR

The operating system has encountered an error while processing the verb. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

For the meaning of the operating system return code, refer to the file **/usr/include/errno.h**.

DEFINE_TRACE

The DEFINE_TRACE verb enables or disables tracing for specified Application Program Interfaces (APIs).

The trace files must be set up before the application which issues this verb is started, using the SNATRC environment variable. For information about how to control API tracing, refer to the *IBM Communications Server for AIX Diagnostics Guide* or *IBM Communications Server for Linux Diagnostics Guide*.

The operation of this verb is affected by the SNACTL environment variable (for more information, see "SNACTL Environment Variable" on page 26).

VCB Structure

```
typedef struct define_trace
  {
  AP_UINT16        opcode;            /* Verb identifying operation code.   */
  unsigned char    opext;             /* Verb extension code - reserved.    */
  unsigned char    reserv2;           /* Reserved.                          */
  AP_UINT16        primary_rc;        /* Primary return code from verb.     */
  AP_UINT32        secondary_rc;      /* Secondary (qualifying) return code.*/
  unsigned char    reserv3[8];        /* Reserved.                          */
  unsigned char    dt_set;            /* Trace state to be set (on/off).    */
  unsigned char    appc;              /* Tracing for APPC.                  */
  unsigned char    nof;               /* Tracing for NOF.                   */
  unsigned char    srpi;              /* Reserved.                          */
  unsigned char    sdlc;              /* Reserved.                          */
  unsigned char    tkn_rng_dlc;       /* Reserved.                          */
  unsigned char    pcnet_dlc;         /* Reserved.                          */
  unsigned char    dft;               /* Reserved.                          */
  unsigned char    acdi;              /* Reserved.                          */
  unsigned char    reserv5;           /* Reserved.                          */
  unsigned char    comm_serv;         /* Tracing for Comm_Serv_API.         */
  unsigned char    ehllapi;           /* Reserved.                          */
  unsigned char    x25_api;           /* Reserved.                          */
  unsigned char    x25_dlc;           /* Reserved.                          */
  unsigned char    twinax;            /* Reserved.                          */
  unsigned char    ms;                /* Tracing for MS.                    */
  unsigned char    rui;               /* Tracing for RUI interface of LUA.  */
  unsigned char    etherand;          /* Reserved.                          */
  unsigned char    subsym;            /* Reserved.                          */
  unsigned char    reserv7[8];        /* Reserved.                          */
  unsigned char    reset_trc;         /* Flag to reset the trace files.     */
  AP_UINT16        trunc;             /* Truncation size for trace records. */
  AP_UINT16        strg_size;         /* Reserved.                          */
  unsigned char    reserv8[1];        /* Reserved.                          */
  unsigned char    phys_link[8];      /* Reserved.                          */
  unsigned char    reserv9[56];       /* Reserved.                          */
  };
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_DEFINE_TRACE

dt_set Specifies whether the DEFINE_TRACE verb is being used to turn tracing on or to turn tracing off.

Possible values are:

SV_ON Enable tracing for a particular API if the parameter for that API (*appc*, *nof*, *comm_serv*, *ms* or *rui*) has bit 0 set to 1; do not modify tracing for the API if the parameter has bit 0 set to 0.

SV_OFF Disable tracing for a particular API if the parameter for that API has bit 0 set to 1; do not modify tracing for the API if the parameter has bit 0 set to 0.

appc Specifies whether the state of APPC and CPI-C tracing (on or off) is to be changed. This option controls both APPC and CPI-C tracing; they cannot be controlled independently.

Communications Server checks only the most significant bit (bit 0) of this byte; other bits are ignored.

To enable or disable tracing for APPC and CPI-C, depending on the *dt_set* parameter, set the most significant bit of this byte to 1.

To leave tracing in its current state for APPC and CPI-C, set the most significant bit of this byte to zero.

nof Specifies whether the state of NOF tracing (on or off) is to be changed.

Communications Server checks only the most significant bit (bit 0) of this byte; other bits are ignored.

To enable or disable NOF tracing, depending on the *dt_set* parameter, set the most significant bit of this byte to 1.

To leave tracing in its current state for NOF, set the most significant bit of this byte to zero.

comm_serv
Specifies whether the state of tracing for the Common Service Verbs (on or off) is to be changed.

Communications Server checks only the most significant bit (bit 0) of this byte; other bits are ignored.

To enable or disable tracing for Common Service Verbs, depending on the *dt_set* parameter, set the most significant bit of this byte to 1.

To leave tracing in its current state for Common Service Verbs, set the most significant bit of this byte to zero.

ms Specifies whether the state of MS tracing (on or off) is to be changed.

Communications Server checks only the most significant bit (bit 0) of this byte; other bits are ignored.

To enable or disable MS tracing, depending on the *dt_set* parameter, set the most significant bit of this byte to 1.

To leave tracing in its current state for MS, set the most significant bit of this byte to zero.

rui Specifies whether the state of tracing for the RUI interface of LUA (on or off) is to be changed.

Communications Server checks only the most significant bit (bit 0) of this byte; other bits are ignored.

To enable or disable tracing for the RUI interface, depending on the *dt_set* parameter, set the most significant bit of this byte to 1.

To leave tracing in its current state for the RUI interface, set the most significant bit of this byte to zero.

reset_trc
Specifies whether to reset the trace file or files. Possible values are:

SV_YES Reset the trace file or files; empty the files and discard their current contents.

SV_NO Do not reset the trace files.

trunc The length at which each trace record is to be truncated. Specify zero if you do not want truncation.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameter:

primary_rc
SV_OK

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
Possible values are:

SV_INVALID_SET
The *dt_set* parameter contained a value that was not valid.

SV_INVALID_RESET_TRACE
The *reset_trc* parameter contained a value that was not valid.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*):

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

SV_UNEXPECTED_DOS_ERROR
The operating system has encountered an error while processing the verb.

The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

For the meaning of the operating system return code, refer to the file **/usr/include/errno.h**.

SNACTL Environment Variable

The SNACTL environment variable is provided by Communications Server for debugging application programs which use the DEFINE_TRACE verb. If this variable is set, DEFINE_TRACE verbs issued by the program will have no effect on tracing (although they will still return SV_OK unless an error occurs). This can be used to force tracing of a program which normally turns tracing off, or to suppress tracing of a program which normally uses it. For more information about tracing and on this environment variable, refer to the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide*.

GET_CP_CONVERT_TABLE

The GET_CP_CONVERT_TABLE verb creates and returns a 256-byte conversion table to translate character strings from a source code page to a target code page. If a character from the source code page does not exist in the target code page, the translated (target) string differs from the original (source) string.

A code page is a table that associates specific ASCII or EBCDIC values with specific characters. It is used to provide a national language variant of ASCII or EBCDIC which supports characters specific to that language. For a list of code pages supported by Communications Server and the national languages for which they are used, see Appendix A, “Code Pages,” on page 41.

VCB Structure

AIX, LINUX

```
typedef struct get_cp_convert_table
  {
  AP_UINT16        opcode;          /* Verb identifying operation code.      */
  unsigned char    opext;           /* Verb extension code - reserved.       */
  unsigned char    reserv2;         /* Reserved.                             */
  AP_UINT16        primary_rc;      /* Primary return code from verb.        */
  AP_UINT32        secondary_rc;    /* Secondary (qualifying) return code.   */
  AP_UINT16        source_cp;       /* Source code page for conversion table.*/
  AP_UINT16        target_cp;       /* Target code page for conversion table.*/
  unsigned char    *conv_tbl_addr;  /* Address to put conversion table at.   */
  unsigned char    char_not_fnd;    /* Character not found option: either    */
                                    /*  substitute character or round trip.  */
  unsigned char    substitute_char; /* Substitute character to use.          */
  };
```

WINDOWS

```
typedef struct get_cp_convert_table
  {
  unsigned short    opcode;          /* Verb identifying operation code.      */
```

```
unsigned char     opext;           /* Verb extension code - reserved.       */
unsigned char     reserv2;         /* Reserved.                             */
unsigned short    primary_rc;      /* Primary return code from verb.        */
unsigned long     secondary_rc;    /* Secondary (qualifying) return code.   */
unsigned short    source_cp;       /* Source code page for conversion table.*/
unsigned short    target_cp;       /* Target code page for conversion table.*/
unsigned char     *conv_tbl_addr;  /* Address to put conversion table at.   */
unsigned char     char_not_fnd;    /* Character not found option: either    */
                                   /*  substitute character or round trip.  */
unsigned char     substitute_char; /* Substitute character to use.          */
};
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_GET_CP_CONVERT_TABLE

source_cp

Source code page (from which characters are converted).

A decimal number which identifies the code page to be used. For a list of valid code page numbers, see Appendix A, "Code Pages," on page 41.

target_cp

Target code page (to which characters are converted).

A decimal number which identifies the code page to be used. For a list of valid code page numbers, see Appendix A, "Code Pages," on page 41.

conv_tbl_addr

Address of buffer to contain the 256-byte conversion table.

char_not_fnd

Specifies the action to take if a character in the source code page does not exist in the target code page.

Possible values are:

SV_ROUND_TRIP

Store a unique value in the conversion table for each source code-page character. This value is useful only if you build a second conversion table to convert between the same two code pages in the reverse direction. If you specify the SV_ROUND_TRIP value in building both conversion tables, any character translated from one code page to the other and then back will be unchanged.

SV_SUBSTITUTE

Store a substitute character (specified by the *substitute_char* parameter) in the conversion table. Converting the translated character string back to the original code page will not necessarily recreate the original character string.

substitute_char

Specifies the character to store in the conversion table when a character from the source code page has no equivalent in the target code page.

Use this parameter only if the *char_not_fnd* parameter is set to SV_SUBSTITUTE.

When the target code page is an EBCDIC code page, this parameter should be set to the EBCDIC value of the character you want to use, not to the actual character. For example, to use the – character as the substitute character in an ASCII to EBCDIC conversion table, supply the value 60 (the value associated with the character – in EBCDIC), and not the actual character –. When the target code page is an ASCII code page, you can specify either the character or its ASCII value.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameter:

primary_rc
SV_OK

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
Possible values are:

SV_INVALID_CHAR_NOT_FOUND
The *char_not_fnd* parameter contained a value that was not valid.

SV_INVALID_SOURCE_CODE_PAGE
The code page specified by the *source_cp* parameter is not supported.

SV_INVALID_TARGET_CODE_PAGE
The code page specified by the *target_cp* parameter is not supported.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*):

WINDOWS

SV_COMM_SUBSYSTEM_NOT_LOADED
The Remote API Client software has not been started. Consult the System Administrator for corrective action.

SV_INVALID_VERB_SEGMENT
The supplied VCB extended beyond the boundary of a data segment.

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

SV_UNEXPECTED_DOS_ERROR
The operating system has encountered an error while processing the verb. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

AIX, LINUX

For the meaning of the operating system return code, refer to the file **/usr/include/errno.h**.

WINDOWS

For the meaning of the operating system return code, refer to your operating system documentation.

LOG_MESSAGE

AIX, LINUX

The LOG_MESSAGE verb records a message in the Communications Server error or audit log file. The text for the message is taken from a user-defined message file; the verb can also supply parameters to be inserted in the message.

If you use this verb, you will need to supply an appropriate message file for use with the application. For more information, see "Creating a Log Message File" on page 32.

For more information about the Communications Server audit and error log files and the format of the logged messages, refer to the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide*.

VCB Structure

```
typedef struct log_message
  {
  AP_UINT16        opcode;           /* Verb identifying operation code.     */
  unsigned char    opext;            /* Verb extension code - reserved.      */
  unsigned char    reserv2;          /* Reserved.                            */
  AP_UINT16        primary_rc;       /* Primary return code from verb.       */
  AP_UINT32        secondary_rc;     /* Secondary (qualifying) return code.  */
  AP_UINT16        msg_num;          /* Number of message to log.            */
  unsigned char    orignтr_id[8];    /* ID of the originator of the message. */
  unsigned char    msg_file_name[3]; /* Message file to search for the       */
```

```
                                    /*  required message number.           */
unsigned char    msg_act;           /* Message action - how to log the msg. */
AP_UINT16        msg_ins_len;       /* Length of data for insertion in msg. */
unsigned char    *msg_ins_ptr;      /* Address of data for insertion in msg.*/
};
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_LOG_MESSAGE

msg_num
Number of the message in the message file specified by *msg_file_name*.

The message identifier shown in the Communications Server log file consists of two parts: the Communications Server component identifier and the message number. The *msg_num* parameter gives the message number; the component identifier for a message logged by this verb is always 32,767.

orignt_id
Name of the component issuing the LOG_MESSAGE verb; a string of up to eight characters. This parameter is optional; set the first byte to 0x00 if you do not want to include it.

If you specify this name, Communications Server uses it as the first parameter inserted into the message text; that is, this name replaces "%1" in the message text. For further information, see "Creating a Log Message File" on page 32.

msg_file_name
Name of the file containing the text for the message to be logged. For information about how to create this message file, see "Creating a Log Message File" on page 32.

The message file must have a name consisting of three characters followed by the **.msg** extension. This parameter specifies only the base file name; the **.msg** extension is added automatically.

The message file must be stored in the directory **/usr/lib/sna** (AIX) or **/opt/ibm/sna/lib** (Linux) on the computer where the application is running. If Communications Server is set up to use centralized logging on a single server, the same message file must also be in **/usr/lib/sna** on the server that holds the log file.

msg_act
Action to be taken when processing the message. This defines the log category (problem, exception, or audit) of the logged message; refer to the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide* for more information about log categories. Possible values are:

SV_PROBLEM
Log as a problem message.

SV_EXCEPTION
Log as an exception message.

SV_AUDIT
Log as an audit message.

For compatibility with other CSV implementations, the following values are also supported. These are provided for migration only, because the

mapping between these values and the Communications Server log categories is only approximate and may not always give the most appropriate category; use the values SV_PROBLEM, SV_EXCEPTION, or SV_AUDIT when writing new applications.

SV_INTRV, SV_INTRV_16, SV_NO_INTRV_16
Equivalent to SV_PROBLEM

SV_NO_INTRV, SV_NO_INTRV_10
Equivalent to SV_EXCEPTION

SV_NO_INTRV_8, SV_NO_INTRV_6
Equivalent to SV_AUDIT

A message of type SV_EXCEPTION or SV_AUDIT, or equivalent, will be logged only if Communications Server is currently configured to log messages of the appropriate type (exception or audit); otherwise the message will be ignored (although the verb will still return SV_OK). Values other than SV_INTRV and SV_NO_INTRV may not be supported by other CSV implementations.

msg_ins_len
Length of data to be inserted into the message (0–1000 characters). Specify a length of 0 (zero) if no data is to be inserted.

msg_ins_ptr
Address of the data to be inserted into the message. This parameter is ignored if *msg_ins_len* is 0 (zero).

The data consists of 1–19 null-terminated strings. The total length of the inserted data must not exceed 1000 characters.

When you create a log message file, you specify the positions in the message text where these data strings are to be inserted. For further information, see "Creating a Log Message File" on page 32. The data supplied to this verb must include a string for each parameter required by the message text; the first string may be supplied in the *orignt_id* parameter instead of in this data string.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameter:

primary_rc

SV_OK Either the message was logged successfully, or the message was ignored because Communications Server is not currently configured to log messages of the specified type (exception or audit).

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
One of the following:

SV_INVALID_FIRST_CHARACTER
The first character of the *msg_file_name* parameter was zero or a space character.

SV_INVALID_MESSAGE_ACTION
The *msg_act* parameter contained a value that was not valid.

There is no SV_PARAMETER_CHECK secondary return code indicating that the specified message file was not found or could not be opened; this error will cause a return code of SV_UNEXPECTED_DOS_ERROR.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*):

SV_COMM_SUBSYSTEM_NOT_LOADED
The Remote API Client software has not been started. Contact the System Administrator for corrective action.

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

SV_UNEXPECTED_DOS_ERROR
The operating system has encountered an error while processing the verb. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

For the meaning of the operating system return code, refer to the file **/usr/include/errno.h**.

Creating a Log Message File

The **snamsgf** program provided with Communications Server, enables you to produce your own message files to be used with the LOG_MESSAGE verb.

To use this facility, you must first create a text file containing the message numbers and text, and then use **snamsgf** to convert it into a message file.

Message Source File Format

A message source file is a plain ASCII text file. You can include a comment line anywhere in the file by using an asterisk (*) as the first character of the line. Communications Server ignores all the remaining text on this line.

The first line in the source file must be “ID:”, followed by a character string of 1–8 characters identifying the component logging the message. This string is printed out at the start of each message in the log file. Specify a string that identifies the user of this message file; for example, the name of the application if only one application uses this message file, or a string identifying a group of applications that use the same message file.

The rest of the message source file consists of entries for individual messages. Each message is defined as a series of fields, as shown in the example that follows.

```
ID:MYAPPL
Message:       1
Type:          PROBLEM
Cause Type:    CSV
Cause:         The specified file could not be opened.
Action:        Check the reason shown on this message for more information.
Flags:         NONE
String:        Could not open the file.$
Filename = %1\nReason   = %2
```

The fields are as follows:

Message
A unique identifier for the message (a decimal number in the range 1–65,535). The messages in the file must be listed in ascending order of message number. Numbers do not need to be consecutive; however, large ranges of unused message numbers will increase the size of the message file.

Type The category of log message. Specify PROBLEM, EXCEPTION, or AUDIT. The actual category that Communications Server uses when logging the message is determined by the *msg_act* parameter of the LOG_MESSAGE verb. In the source file, this information is included for readability, but Communications Server ignores it.

Cause type
A summary of the cause of the message. Specify CSV (to indicate that the message was logged using the CSV LOG_MESSAGE verb), or one of the following values:

Internal
Internal error in the application.

Resource
Resource shortage (for example, insufficient memory on the AIX / Linux computer).

User User error (for example, parameters that are not valid supplied on the command line to an application program).

SNA SNA protocol violation by a remote system.

Config
Configuration mismatch.

Audit A normal event, reported for information only.

Cause The cause of the condition being logged.

Action
Any action that the local System Administrator should take as a result of the message. For audit messages, which provide accounting and progress information instead of reporting error conditions, there is generally no action required.

Flags Specify CONSOLE to indicate that the message should be written to the AIX / Linux computer's system console as well as to the log file, or NONE to indicate that the message should be written only to the log file.

String The text of the message (1–256 characters). To include parameters supplied to the LOG_MESSAGE verb, use %1, %2, and so on to indicate the position

of each parameter. When logging the message, Communications Server replaces %1 with the first parameter supplied to LOG_MESSAGE, %2 with the second parameter, and so on.

The *origintr_id* parameter supplied to LOG_MESSAGE, if any, replaces %1. The first parameter in the data string supplied to LOG_MESSAGE replaces %2 (if *orignt_id* was used) or %1 (if *orignt_id* was not used); the second parameter in the data string replaces %3 or %2, and so on.

The following also applies to these fields:

- Each field name (such as Message) must be at the start of a line, followed by a colon. Spaces or tabs following the colon are ignored. All the text associated with the field name must be in a single line (except when lines are concatenated using the $ character, as described below); there is no limit on the length of the line.
- In the Cause, Action, and String fields, the following characters can be used to control the format of the text written to the log file:

  \t Insert a tab character in the output text.

  $ (followed by a new-line character in the source text)
  Insert a new-line character in the output text, and continue with the following line of the source file. This enables you to specify a text field that extends over more than one line. The last line of the text field must not end with a $ character.

  \n Insert a new-line character in the output text, and continue with the following character of the source file. This enables you to specify a text field as a single line in the source file, and to specify where line breaks will appear in the output. However, it is recommended that you split long text fields into multiple lines using the $ character, as described above, for readability.

  \$ Insert a $ character in the output text.

  %n (in the String parameter only)
  Insert the nth parameter supplied to the log call in the output text.

  The logging code does not insert new-line characters into text strings except where \n or $ characters are included in the source text. To ensure that the output text is easily readable on an 80-column screen, use these characters to force line breaks.
- The fields Message, Type, Cause Type, Flags, and String must be specified. The fields Cause and Action are optional; to indicate that one of these fields is not used, specify the following string, with capitalization as shown:

  @!* Not Used

  For example, if the message is an audit message and no action is required, use the following line:

  ```
  Action:   @!* Not Used
  ```

  In this case, Communications Server will not include the Action field when writing the message to the log file.
- The total length of the Cause and Action fields must not exceed 2048 characters.

Sample Log Message Output

The previous section shows a sample entry in the message source file. If you build a message file from a source file containing this entry, an application can call LOG_MESSAGE specifying message number 1 in this message file. The

application's supplied data must contain two null-terminated strings, one specifying the file name (for example, **/usr/jim/myfile**) and one specifying the reason for the failure (for example, "File not found"). The output will then be as follows:

```
------------------12:17:28 BST 05/13/1994 ------------------
MYAPPL          Message 32767-1, Subcode: 0
Log category:   PROBLEM   Cause Type: CSV
System:         jimsbox
Process ID:     12345
Could not open the file.
Filename = /usr/jim/myfile
Reason   = File not found
Cause:          The specified file could not be opened.
Action:         Check the reason shown on this message for more information.
```

This sample output assumes that verbose logging (not succinct logging) is being used. For more information about succinct logging, and the format of entries in the log file if it is being used, refer to the chapter on log messages in the *IBM Communications Server for AIX Diagnostics Guide* or the *IBM Communications Server for Linux Diagnostics Guide*.

Creating the Message File from the Text File

To convert the text file into a message file, use the **snamsgf** program as follows:

snamsgf *infile outfile*

The name of the input text file is *infile*, including a path if it is not in the current directory.

The name of the output message file is *outfile*, as specified by the *msg_file_name* parameter on LOG_MESSAGE. The output file must have a name consisting of 1–3 characters with the extension **.msg**; you need not specify the extension on the command line.

The output file is created in the current directory. It must be stored in the directory **/usr/lib/sna** (AIX) or **/opt/ibm/sna** (Linux) in order for Communications Server to find it when it is specified by a LOG_MESSAGE verb.

For example, the following command creates the message file **new.msg** from the source text file **/usr/fred/myfile.text**:

snamsgf /usr/fred/myfile.text new

The **snamsgf** program writes error messages to standard error if it detects errors in the input file format.

TRANSFER_MS_DATA

WINDOWS

The TRANSFER_MS_DATA verb builds a request unit (RU) containing Network Management Vector Transport (NMVT) data. The verb can send the NMVT data to NetView for centralized problem diagnosis and resolution. The data can also be logged in the local error log file.

The application can supply a complete NMVT to be sent, or it can supply some of the required subvectors and request Communications Server to add header information or additional subvectors. For more information about the format of NMVTs, including the format of the headers and subvectors that Communications Server adds, refer to *IBM Systems Network Architecture: Formats*.

VCB Structure

```
typedef struct transfer_ms_data
{
   unsigned short      opcode;             /* Verb operation code           */
   unsigned char       data_type;          /* Type of data supplied by appl */
   unsigned char       reserv2;            /* reserved                      */
   unsigned short      primary_rc;         /* Primary return code           */
   unsigned long       secondary_rc;       /* Secondary return code         */
   unsigned char       options;            /* Verb options                  */
   unsigned char       reserv3;            /* reserved                      */
   unsigned char       originator_id[8];   /* Originator ID                 */
   unsigned short      dlen;               /* Length of data                */
   unsigned char       *dptr;              /* Data                          */
} TRANSFER_MS_DATA;
```

Supplied Parameters

The program using this verb supplies the following parameters:

opcode SV_TRANSFER_MS_DATA

data_type

Possible values are:

SV_NMVT
The data contains a complete NMVT.

SV_ALERT_SUBVECTORS
The data contains MS subvectors in the SNA-defined format for an Alert major vector. Communications Server adds an NMVT header and an alert major vector header.

SV_USER_DEFINED
The data contains a complete NMVT request unit. Communications Server always logs the data, and does not send it to NetView.

SV_PDSTATS_SUBVECTORS
The data contains problem determination statistics. Communications Server always logs the data, and does not send it to NetView.

options This parameter is a one-byte value, with individual bits indicating the options selected. Bit 0 is the most significant and bit 7 is the least significant bit. For compatibility with other implementations, the bit values for bits 0–3 are defined so that a value of 1 indicates no action and a value of 0 indicates an action. (Bits 1–3 are ignored if *data_type* is set to SV_USER_DEFINED.)

Bit 0—Add Date/Time (0x01) subvector to the data.

- To request Communications Server to add the subvector, set this bit to 0.

- To request Communications Server not to add the subvector, set this bit to 1.

Bit 1—Add Product Set ID (0x10) subvector to the data. If the application supplies data that already contains a Product Set ID subvector, Communications Server adds its own Product Set ID subvector immediately preceding the existing one.

- To request Communications Server to add the subvector, set this bit to 0.
- To request Communications Server not to add the subvector, set this bit to 1.

Bit 2—Send the data to NetView.

- To request Communications Server to send the data, set this bit to 0.
- To request Communications Server not to send the data, set this bit to 1.

If *data_type* is set to SV_USER_DEFINED or SV_PDSTATS_SUBVECTORS, this bit is ignored; the data cannot be sent to NetView.

Bit 3—Log the data in the Communications Server error log file.

- To request Communications Server to log the data, set this bit to 0.
- To request Communications Server not to log the data, set this bit to 1.

If *data_type* is set to SV_USER_DEFINED or SV_PDSTATS_SUBVECTORS, this bit is ignored; the data is always logged.

Bits 4–7 are reserved, and must be set to 0.

originator_id
Name of the component that issued the verb. If the data is being logged in the Communications Server error log file, this name is used to identify the originator of the log message; otherwise it is not used.

This is an ASCII string of up to eight characters, using any locally displayable characters. The parameter is optional; set the first character to 0x00 if you do not want to include it.

dlen Length of the data supplied by the application.

The maximum length of an NMVT is 512 bytes. If the application is supplying a complete NMVT, the data length must not exceed 512 bytes. If the application is supplying alert subvectors, or requesting Communications Server to add one or more subvectors to the supplied data, the total length after addition of the required headers and/or subvectors must not exceed 512 bytes.

dptr A pointer to the data string supplied by the application. The data must be in the valid format for an NMVT, alert subvectors, or problem determination statistics, as specified by the *data_type* parameter.

Returned Parameters

After the verb executes, Communications Server returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was unsuccessful.

Successful Execution

If the verb executes successfully, Communications Server returns the following parameters:

primary_rc
SV_OK

secondary_rc
Not used.

Unsuccessful Execution

When a verb does not execute successfully, Communications Server returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for unsuccessful execution.

Parameter Check: If the verb does not execute because of a parameter error, Communications Server returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
Possible values are:

SV_INVALID_DATA_TYPE
The supplied *data_type* parameter was not one of the valid values.

SV_INVALID_DATA_SEGMENT
The supplied data string extended beyond the boundary of a data segment.

SV_DATA_EXCEEDS_RU_SIZE
One of the following occurred:

- The application supplied a data string longer than the maximum NMVT size of 512 bytes.
- The application supplied data as alert subvectors, or specified that Communications Server should add one or more subvectors to it, but the added headers and/or subvectors increased the data size beyond 512 bytes.

State Check: If the verb does not execute because of a state error, Communications Server returns the following parameters:

primary_rc
SV_STATE_CHECK

secondary_rc

SV_SSCP_PU_SESSION_NOT_ACTIVE
The application specified SV_SEND in the *options* parameter, but the session to the appropriate PU was not active.

Other Conditions: Other conditions can result in the following primary return codes (*primary_rc*):

primary_rc

SV_CANCELLED
The WinCSVCleanup call was issued while this verb (issued using the asynchronous entry point) was still outstanding. This verb has been cancelled; the data may not have been sent.

primary_rc

SV_COMM_SUBSYSTEM_NOT_LOADED
The Remote API Client software has not been started, or has been stopped.

primary_rc

SV_INVALID_VERB
The *opcode* parameter did not match the operation code of any verb. No verb executed.

primary_rc

SV_INVALID_VERB_SEGMENT
The supplied VCB extended beyond the boundary of a data segment.

primary_rc

SV_SERVER_RESOURCE_NOT_FOUND
A required Communications Server component was not active; the data could not be sent.

primary_rc

SV_SERVER_RESOURCES_LOST
A required Communications Server resource was not available.

secondary_rc

SV_SERVER_COMM_FAILURE
The communications path to a required Communications Server component has failed; the data could not be sent.

primary_rc

SV_THREAD_BLOCKING
The verb was issued using the synchronous CSV entry point, but a synchronous verb is already in progress for this application. Only one synchronous verb can be in progress at any time.

primary_rc

SV_UNEXPECTED_DOS_ERROR
The operating system has encountered an error while processing the verb. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult the System Administrator for corrective action.

For the meaning of the operating system return code, refer to your operating system documentation.

This return code may also indicate that the application issuing the verb was invoked using the Windows function SendMessage instead of PostMessage; the application cannot issue any verbs in this state. For more information, see "Windows Considerations" on page 12.

Appendix A. Code Pages

This appendix lists the code pages supported by Communications Server for use with the GET_CP_CONVERT_TABLE verb, and the national language variants of ASCII or EBCDIC that use each code page.

ASCII Code Pages

AIX, LINUX

8859 Generalized ASCII code page defined by ISO 8859, used to support all language variants

[redacted]

437 US English

737 Greece

813 Greece

819 ANSI

850 International code page: US English, UK English, French, German, Italian, Spanish, Finnish, Netherlands, Swedish, Swiss, Belgian, Latin American

852 Poland, Hungary, Romania, Slovakia, Czech, Croatia, Slovenia

855 Bulgaria, Serbia-Montenegro, FYR Macedonia

857 Turkey

858 Multilingual

860 Portuguese

861 Iceland

862 Hebrew

863 Canadian French

864 Arabic

865 Danish, Norwegian

866 Russia

869 Greece

874 Thailand

897 Japan

903 People's Republic of China

912 Poland, Hungary, Romania, Slovakia, Czech, Croatia, Slovenia

915 Russia, Bulgaria, Serbia-Montenegro, FYR Macedonia

916 Hebrew

920 Turkey
921 Latvia, Lithuania
922 Estonia
923 ANSI
1008 Arabic
1041 Japan
1088 Korea
1089 Arabic
1114 Republic of China (Taiwan)
1115 People's Republic of China
1124 Ukraine
1125 Ukraine
1126 Korea
1127 Arabic
1129 Vietnam
1131 Belarus
1133 Laos
1250 Poland, Hungary, Romania, Slovakia, Czech, Croatia, Slovenia
1251 Russia, Bulgaria, Serbia-Montenegro, FYR Macedonia
1252 United States / Multilingual
1253 Greece
1254 Turkey
1255 Hebrew
1256 Arabic
1257 Baltic
1258 Vietnam

EBCDIC Code Pages

037 US English, Canadian Bilingual, Netherlands, Portuguese
273 German
275 Brazil
277 Danish, Norwegian
278 Finnish, Swedish
280 Italian
284 Spanish, Latin American
285 UK English
290 Japan
297 French

420 Arabic

424 Hebrew

500 Belgian (New), Swiss French, Swiss German

803 Arabic

833 Korea

836 People's Republic of China

838 Thailand

870 Poland, Hungary, Romania, Slovakia, Czech, Croatia, Slovenia

871 Iceland

875 Greece

924 USA, Canada (French), Netherlands, Portugal, France, Finland

1025 Russia, Bulgaria, Serbia-Montenegro, FYR Macedonia

1026 Turkey

1027 Japan

1047 USA, Canada (French), Netherlands, Portugal

1112 Latvia, Lithuania

1122 Estonia

1123 Baltic

1130 Vietnam

1132 Laos

1140 USA, Canada (French), Netherlands, Portugal

1141 Germany, Austria

1142 Denmark, Norway

1143 Finland, Sweden

1144 Italy

1145 Latin America, Spain

1146 United Kingdom

1147 France

1148 Belgium, Switzerland (French), Switzerland (German)

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE: This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows: ® (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ® Copyright IBM Corp. 2000, 2005, 2006, 2007, 2008, 2009. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countires, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Bibliography

The following IBM publications provide information about the topics discussed in this library. The publications are divided into the following broad topic areas:

- IBM Communications Server for AIX
- IBM Communications Server for Linux
- Systems Network Architecture (SNA)
- Advanced Program-to-Program Communication (APPC)
- Programming

For IBM Communications Server for AIX and IBM Communications Server for Linux books, brief descriptions are provided. For other books, only the titles and order numbers are shown here.

IBM Communications Server for AIX Publications

The IBM Communications Server for AIX library comprises the following books. In addition, softcopy versions of these documents are provided on the CD-ROM. See *IBM Communications Server for AIX Quick Beginnings* for information about accessing the softcopy files on the CD-ROM. To install these softcopy books on your system, you require 9–15 MB of hard disk space (depending on which national language versions you install).

- *IBM Communications Server for AIX Migration Guide* (SC31-8585)

  This book explains how to migrate from Communications Server for AIX Version 4 Release 2 or earlier to IBM Communications Server for AIX Version 6.
- *IBM Communications Server for AIX Quick Beginnings* (GC31-8583)

  This book is a general introduction to IBM Communications Server for AIX, including information about supported network characteristics, installation, configuration, and operation.
- *IBM Communications Server for AIX Administration Guide* (SC31-8586)

  This book provides an overview of SNA and IBM Communications Server for AIX, and information about IBM Communications Server for AIX configuration and operation.
- *IBM Communications Server for AIX Administration Command Reference* (SC31-8587)

  This book provides information about SNA and IBM Communications Server for AIX commands.
- *IBM Communications Server for AIX or Linux CPI-C Programmer's Guide* (SC23-8591)

  This book provides information for experienced "C" or Java™ programmers about writing SNA transaction programs using the IBM Communications Server CPI Communications API.
- *IBM Communications Server for AIX or Linux APPC Programmer's Guide* (SC23-8592)

  This book contains the information you need to write application programs using Advanced Program-to-Program Communication (APPC).
- *IBM Communications Server for AIX or Linux LUA Programmer's Guide* (SC23-8590)

  This book contains the information you need to write applications using the Conventional LU Application Programming Interface (LUA).

- *IBM Communications Server for AIX or Linux CSV Programmer's Guide* (SC23-8589)

  This book contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).
- *IBM Communications Server for AIX or Linux MS Programmer's Guide* (SC23-8596)

  This book contains the information you need to write applications using the Management Services (MS) API.
- *IBM Communications Server for AIX NOF Programmer's Guide* (SC31-8595)

  This book contains the information you need to write applications using the Node Operator Facility (NOF) API.
- *IBM Communications Server for AIX Diagnostics Guide* (SC31-8588)

  This book provides information about SNA network problem resolution.
- *IBM Communications Server for AIX or Linux APPC Application Suite User's Guide* (SC23-8595)

  This book provides information about APPC applications used with IBM Communications Server for AIX.
- *IBM Communications Server for AIX Glossary* (GC31-8589)

  This book provides a comprehensive list of terms and definitions used throughout the IBM Communications Server for AIX library.

IBM Communications Server for Linux Publications

The IBM Communications Server for Linux library comprises the following books. In addition, softcopy versions of these documents are provided on the CD-ROM. See *IBM Communications Server for Linux Quick Beginnings* for information about accessing the softcopy files on the CD-ROM. To install these softcopy books on your system, you require 9–15 MB of hard disk space (depending on which national language versions you install).

- *IBM Communications Server for Linux Quick Beginnings* (GC31-6768 and GC31-6769)

  This book is a general introduction to IBM Communications Server for Linux, including information about supported network characteristics, installation, configuration, and operation. There are two versions of this book:
  GC31-6768 is for IBM Communications Server for Linux on the i686, x86_64, and ppc64 platforms
  GC31-6769 is for IBM Communications Server for Linux on System z.
- *IBM Communications Server for Linux Administration Guide* (SC31-6771)

  This book provides an overview of SNA and IBM Communications Server for Linux, and information about IBM Communications Server for Linux configuration and operation.
- *IBM Communications Server for Linux Administration Command Reference* (SC31-6770)

  This book provides information about SNA and IBM Communications Server for Linux commands.
- *IBM Communications Server for AIX or Linux CPI-C Programmer's Guide* (SC23-8691)

  This book provides information for experienced "C" or Javaprogrammers about writing SNA transaction programs using the IBM Communications Server CPI Communications API.
- *IBM Communications Server for AIX or Linux APPC Programmer's Guide* (SC23-8692)

This book contains the information you need to write application programs using Advanced Program-to-Program Communication (APPC).

- *IBM Communications Server for AIX or Linux LUA Programmer's Guide* (SC23-8690)

  This book contains the information you need to write applications using the Conventional LU Application Programming Interface (LUA).
- *IBM Communications Server for AIX or Linux CSV Programmer's Guide* (SC23-8689)

  This book contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).
- *IBM Communications Server for AIX or Linux MS Programmer's Guide* (SC23-8596)

  This book contains the information you need to write applications using the Management Services (MS) API.
- *IBM Communications Server for Linux NOF Programmer's Guide* (SC31-6778)

  This book contains the information you need to write applications using the Node Operator Facility (NOF) API.
- *IBM Communications Server for Linux Diagnostics Guide* (SC31-6779)

  This book provides information about SNA network problem resolution.
- *IBM Communications Server for AIX or Linux APPC Application Suite User's Guide* (SC23-8595)

  This book provides information about APPC applications used with IBM Communications Server for Linux.
- *IBM Communications Server for Linux Glossary* (GC31-6780)

  This book provides a comprehensive list of terms and definitions used throughout the IBM Communications Server for Linux library.

Systems Network Architecture (SNA) Publications

The following books contain information about SNA networks:

- *Systems Network Architecture: Format and Protocol Reference Manual—Architecture Logic for LU Type 6.2* (SC30-3269)
- *Systems Network Architecture: Formats* (GA27-3136)
- *Systems Network Architecture: Guide to SNA Publications* (GC30-3438)
- *Systems Network Architecture: Network Product Formats* (LY43-0081)
- *Systems Network Architecture: Technical Overview* (GC30-3073)
- *Systems Network Architecture: APPN Architecture Reference* (SC30-3422)
- *Systems Network Architecture: Sessions between Logical Units* (GC20-1868)
- *Systems Network Architecture: LU 6.2 Reference—Peer Protocols* (SC31-6808)
- *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084)
- *Systems Network Architecture: 3270 Datastream Programmer's Reference* (GA23-0059)
- *Networking Blueprint Executive Overview* (GC31-7057)
- *Systems Network Architecture: Management Services Reference* (SC30-3346)

APPC Publications

The following books contain information about Advanced Program-to-Program Communication (APPC):

- *APPC Application Suite V1 User's Guide* (SC31-6532)
- *APPC Application Suite V1 Administration* (SC31-6533)
- *APPC Application Suite V1 Programming* (SC31-6534)

- *APPC Application Suite V1 Online Product Library* (SK2T-2680)
- *APPC Application Suite Licensed Program Specifications* (GC31-6535)
- *z/OS V1R2.0 Communications Server: APPC Application Suite User's Guide* (SC31-8809)

Programming Publications

The following books contain information about programming:

- *Common Programming Interface Communications CPI-C Reference* (SC26-4399)
- *Communications Server for OS/2 Version 4 Application Programming Guide* (SC31-8152)

Index

Program Number: 5765-E51 and 5724-i33

Printed in USA

SC23-8589-00