IBM Communications Server for Linux

MS Programmer's Guide

Version 6.2

SC31-6777-00

IBM Communications Server for Linux

MS Programmer's Guide

Version 6.2

SC31-6777-00

Note:
Before using this information and the product it supports, be sure to read the general information under Appendix C, "Notices," on page 49.

First Edition (May 2004)

This edition applies to Version 6 Release 2 of Communications Server for Linux (5724–i33 and 5724–i34) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You may send your comments to the following address.

International Business Machines Corporation
Attn: Globalization Services
P.O. Box 12195, 3039 Cornwallis Road
Department G7IA, Building 500/C10A
Research Triangle Park, North Carolina 27709-2195

You can send us comments electronically by using one of the following methods:

Fax (USA and Canada):
1+919-254-9823

Internet e-mail:

- comsvrcf@us.ibm.com

IBMLink:
CIBMORCF at RALVM17

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

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 writing Management Services (MS) applications to use the Communications Server for Linux MS application programming interface (API). Communications Server for Linux (hereafter referred to as CS Linux) is an IBM® software product that enables a computer running Linux to exchange information with other nodes on an SNA network.

Communications Server for Linux is designed to operate on either an Intel™ workstation running Linux (CS Linux, program product number 5724–i33) or a zSeries® mainframe running a 31–bit or 64–bit Linux for zSeries (CS Linux on zSeries, program product number 5724–i34). In this book, the name CS Linux is used to indicate either of these two versions, and the term "CS Linux computer" is used to indicate either a workstation or a zSeries mainframe running CS Linux, except where differences are described explicitly.

The MS API can be used by applications running on either a server or a Linux client. It cannot be used by applications running on Windows® clients.

This book contains the information required to develop C-language application programs that use the MS API to communicate with remote network management applications. It also provides a brief overview of MS concepts and provides detailed reference information for experienced MS programmers.

This book applies to Version 6.2 of CS Linux.

Who Should Use This Book

This book is intended for experienced C programmers who write Management Services applications for systems with CS Linux. Programmers may or may not have prior experience with SNA or the communication facilities of CS Linux.

Application programmers design and code transaction and application programs that use the CS Linux 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 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, "Introduction to Management Services," on page 1, provides an overview of CS Linux MS support. It describes the various levels of SNA network management support, the function sets and optional subsets supported by the CS Linux MS API, and the functions provided by the CS Linux MS verbs.
- Chapter 2, "Writing MS Applications," on page 5, contains information about writing, compiling, and linking MS applications.

- Chapter 3, "Management Services Verbs," on page 15, provides a detailed description of each of the MS verbs, including parameters and return codes.
- Chapter 4, "Management Services Indications," on page 39, provides a detailed description of each of the indications sent from CS Linux to the application, including parameters and return codes.
- Appendix A, "MS Function Sets," on page 45, lists the SNA MS option sets that the CS Linux MS API supports.

Typographic Conventions

Table 1, shows the typographic styles used in this document.

Table 1. Typographic Conventions

Special Element	Sample of Typography
Document title	*CS Linux Administration Guide*
File or path name	**ms_c.h**
Command or Linux utility	**cc**
Option or flag	**-L**
Parameter	*opcode*
Literal value or selection that the user can enter (including default values)	`0` (zero)
Constant	`AP_CONNECT_MS_NODE`
Return value	`AP_STATE_CHECK`
Variable representing a supplied value	*nnnn*
Environment variable	`LD_RUN_PATH`
Programming verb	CONNECT_MS_NODE
User input	**cc -L /opt/ibm/sna/lib -lms -lsna**
Function, call, or entry point	`ms_async`
Data structure	`MS_CALLBACK`
Hexadecimal value	`0x20`

Related Publications

For detailed information about SNA Management Services, refer to the following IBM document:

- *Systems Network Architecture: Management Services Reference*, SC30-3346

For general information about SNA, APPN, or LU 6.2 architecture, refer to the following IBM documents:

- *Systems Network Architecture:*
  - *APPN Architecture Reference*, SC30-3422
  - *Formats*, GA27-3136
  - *LU6.2 Reference: Peer Protocols*, SC31-6808
  - *Technical Overview*, GC30-3073

Chapter 1. Introduction to Management Services

This chapter introduces the CS Linux Management Services (MS) application programming interface (API). It includes information about the various types of MS support in SNA and about accessing them through Communications Server for Linux.

SNA Management Services Support Levels

SNA defines the following levels of MS support. Each level corresponds to a different generation of products that implement this support.

NMVT-level
An NMVT-level product transfers management information by sending network management vector transports (NMVTs) to, and receiving NMVTs from, a host focal point over a session between the physical unit (PU) in the node that supports the NMVT-level product and the system services control point (SSCP) at the host. This session is called a PU-SSCP session.

NetView® Version 2, Release 1 or earlier provides NMVT-level support.

Migration-level
A migration-level product transfers management information by sending and receiving CP_MSUs (Control Point Management Services Unit GDS variables) over an LU-LU session between independent type 6.2 logical units (LUs). A CP_MSU is a simple GDS variable containing an MS major vector. Migration-level focal points can receive alerts, but they do not support other MS categories.

OS/400® is an example of a migration-level product.

MDS-level
An MDS (Multiple Domain Support)-level product transfers management information by sending and receiving MDS_MUs (MDS Message Unit GDS variables) over LU type 6.2 sessions. An MDS_MU consists of a header, with detailed MS routing and correlation information, followed by a CP_MSU containing an MS major vector. MDS-level products can communicate with more than one focal point at the same time, although they use only a single focal point for a particular MS category (such as problem management).

OS/2® Communications Server/2 and NetView Version 2, Release 2 (as a subarea LU rather than a CP) provide MDS-level support.

CS Linux Management Services Support

The CS Linux MS API enables an application to communicate with other MS products or applications on the SNA network. CS Linux can support NMVT-level and MDS-level applications. The partner MS application can implement any of the levels described in "SNA Management Services Support Levels." CS Linux performs any data conversion that is required.

Management Services Application Programming Interface

The CS Linux MS API comprises the following elements:

MS verbs

Verbs are issued by an MS application to do the following:

- Inform CS Linux when it needs CS Linux resources to support receiving MS data and status indications.
- Send MS data (in either NMVT format or MDS_MU format) to an MS application elsewhere in the network.
- Register the application with CS Linux to receive incoming MS data from focal points (in either NMVT format or MDS_MU format).
- Register the application with CS Linux to receive information about which focal point is responsible for a particular MS category, so that CS Linux can route MDS_MU data to the appropriate application.

For more information about MS verbs, see Chapter 3, "Management Services Verbs," on page 15.

MS indications

Indications are either generated locally by CS Linux or used to forward data received from the network. For more information about MS indications, see Chapter 4, "Management Services Indications," on page 39.

Management Services Applications

The verbs and entry points you use when you write an MS application depend on whether the MS application:

- Only sends data
- Sends and receives data

MS Applications That Only Send Data

This most simple type of application only sends data and never receives any data from the CS Linux node. This type of application can use either the synchronous or asynchronous entry point and needs to use only one or both of the following verbs to send data:

- The SEND_MDS_MU verb sends data in MDS_MU format, which CS Linux sends to a remote MS application.
- The TRANSFER_MS_DATA verb sends data in NMVT format, which CS Linux sends to a remote MS application. The data can be either a complete NMVT or subvectors to which CS Linux adds the required NMVT header information.

For more information about the synchronous and asynchronous entry points, see "Description of the MS API Entry Points" on page 5.

MS Applications That Both Send and Receive Data

This type of application both sends data and receives data and status indications from the CS Linux node. When you write this type of application, you must include the following verbs (except where noted, you can use either the synchronous or asynchronous entry point):

1. Issue a CONNECT_MS_NODE verb to establish communication with the CS Linux node, so that the application can register to receive data, focal point indications, or both.
2. Register with the CS Linux node to indicate the type of data that the application wants to receive. You must use the asynchronous entry point to register with CS Linux using either or both of the following verbs:

- The REGISTER_MS_APPLICATION verb registers the application with CS Linux as an MDS-level application that can accept MDS_MUs. An option on the verb enables the application to request information about the focal point for a particular MS category. CS Linux uses the MDS_MU_RECEIVED indication, the FP_NOTIFICATION indication, or both, to pass the required data to the application.
- The REGISTER_NMVT_APPLICATION verb registers the application with CS Linux in one of the following ways:
  - As an NMVT-level application that accepts NMVTs with a particular MS major vector key. CS Linux then uses the NMVT_RECEIVED indication to pass NMVTs to the application.
  - As an MDS-level application that accepts NMVTs with a particular MS major vector key after they have been converted to MDS_MUs. CS Linux converts the received NMVTs to MDS_MUs and uses the MDS_MU_RECEIVED indication to pass the MDS_MUs to the application. This usage allows an MDS-level application to receive NMVT-level data and status indications without having to understand NMVT-level data formats.

  When the application registers with CS Linux, it supplies the address of a callback routine. CS Linux calls this callback routine when data of the requested type arrives at the node. For more information about the data structures that CS Linux supplies to the callback routine, see Chapter 4, "Management Services Indications," on page 39.

3. After registering itself, the application can do any of the following:
   - Send data to the CS Linux node using either or both of the following verbs:
     - The SEND_MDS_MU verb supplies data in MDS_MU format, which CS Linux sends to a remote MS application.
     - The TRANSFER_MS_DATA verb supplies data in NMVT format, which CS Linux sends to a remote MS application. The data can be either a complete NMVT or subvectors to which CS Linux adds the required NMVT header information.
   - Receive status information from the CS Linux node when CS Linux returns the following status indications:
     - The FP_NOTIFICATION indication provides information about the focal point for a particular MS category. CS Linux returns this indication to an MDS-level application that has registered to receive focal point information.
     - The MS_STATUS indication informs the application of changes in the status of the CS Linux system (when the application's communications path to its connected node has been lost, or when the CS Linux software has stopped). CS Linux returns this indication to both MDS-level and NMVT-level applications.
   - Receive data from the CS Linux node when CS Linux returns the following received data indications:
     - The MDS_MU_RECEIVED data indication returns an MDS_MU to an MDS-level application. The returned MDS_MU is one of the following:
       - The MDS_MU sent by a remote application if the MS application registered using REGISTER_MS_APPLICATION verb
       - An MDS_MU converted from an incoming NMVT if the MS application registered using the REGISTER_NMVT_APPLICATION verb
     - The NMVT_RECEIVED data indication returns an NMVT to an NMVT-level application that has registered to receive NMVTs.

4. When the application completes, it must end its registration with CS Linux by issuing one of the following verbs:
   - The UNREGISTER_MS_APPLICATION verb ends the application's registration with CS Linux. After the application issues this call, CS Linux no longer sends MDS_MUs to the application.
   - The UNREGISTER_NMVT_APPLICATION verb ends the application's registration with CS Linux so that it no longer accepts NMVTs with a particular MS major vector key.
5. After the application ends its registration with CS Linux, it must issue a DISCONNECT_MS_NODE verb to end communication with the CS Linux node and free the resources associated with the application.

For more information about the synchronous and asynchronous entry points, see "Description of the MS API Entry Points" on page 5.

NMVT Routing

When CS Linux receives an NMVT from a remote node, it uses the MS major vector key and the destination application name subfields of the NMVT to determine to which MS application to send the NMVT in the following order of preference:

1. CS Linux attempts to find an NMVT-level application that has registered with an application name matching the NMVT's destination name, in the following order of preference:
   a. An application that has registered to accept the specific major vector key carried on the incoming NMVT
   b. An application that has registered to accept SNA Service Point Command Facility (SPCF) keys, if the major vector key is in the range `0x8061–0x8064`
   c. An application that has registered to accept all keys
2. If CS Linux cannot find a suitable NMVT-level application, CS Linux attempts to find an MDS-level application that has registered with an application name matching the NMVT's destination name and has registered to accept NMVTs after conversion to MDS_MUs. The order of preference for selecting an application that can accept the appropriate major vector key is the same as for NMVT-level applications.

Chapter 2. Writing MS Applications

This chapter describes how an MS application:

- Uses the MS API entry points
- Schedules asynchronous events
- Is compiled and linked to use the MS API

Description of the MS API Entry Points

An application accesses the MS API using the following entry point function calls:

ms An application uses this entry point to issue an MS verb synchronously. CS Linux does not return control to the application until verb processing has finished. All MS verbs except REGISTER_MS_APPLICATION and REGISTER_NMVT_APPLICATION can be issued through this entry point.

An application can use only this entry point if both of the following conditions are true:

- The application only needs to send MS data using the TRANSFER_MS_DATA verb or the SEND_MDS_MU verb or both. (The application does not need to receive MS data or status indications.)
- The application can suspend while waiting for CS Linux to completely process a verb.

The `ms` entry point is defined in the MS header file **/opt/ibm/sna/include/ms_c.h**.

ms_async
An application uses this entry point to issue an MS verb asynchronously. CS Linux returns control to the application immediately, with a returned value indicating whether verb processing is still in progress or has completed. If the returned value indicates that verb processing is still in progress, CS Linux uses an application-supplied callback routine to return the results of the verb processing. If the returned value indicates that verb processing is complete, the callback routine will not be invoked.

All MS verbs can be issued through this entry point. The REGISTER_MS_APPLICATION and REGISTER_NMVT_APPLICATION verbs must be issued through this entry point.

An application must use this entry point if either of the following conditions is true:

- The application needs to receive MS data and status indications.
- The application cannot suspend while waiting for CS Linux to completely process a verb.

The `ms_async` entry point is defined in the MS header file **/opt/ibm/sna/include/ms_c.h**.

Callback routine for `ms_async`
An application must supply a pointer to a callback routine when it uses the asynchronous MS API entry point. CS Linux uses this callback routine both for completion of a verb and also for returning MS data and status indications.

Synchronous Entry Point: ms

An application uses ms to issue an MS verb synchronously. CS Linux does not return control to the application until verb processing has finished.

Function Call

```
void ms (
        AP_UINT32  target_handle,
        void *     msvcb
        );
```

Supplied Parameters

An application supplies the following parameters when it uses the ms entry point:

target_handle
For the UNREGISTER_MS_APPLICATION, UNREGISTER_NMVT_APPLICATION, and DISCONNECT_MS_NODE verbs, the application supplies the value that was returned on the CONNECT_MS_NODE verb. This parameter is used to identify the target CS Linux node.

For all other verbs, this parameter is not used; set it to 0 (zero).

msvcb Pointer to a Verb Control Block (VCB) that contains the parameters for the verb being issued. The VCB structure for each verb is described in Chapter 3, "Management Services Verbs," on page 15. These structures are defined in the MS API **/opt/ibm/sna/include/ms_c.h**.

Note: The MS VCBs contain many parameters marked as "reserved"; some of these are used internally by the CS Linux 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 CS Linux will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future CS Linux 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 ms entry point does not have a return value. When the call returns, the application should examine the return code in the VCB to determine whether the verb completed successfully and to determine parameters it needs for further verbs. In particular, when the CONNECT_MS_NODE verb completes successfully, the VCB contains the *target_handle* that the application should use when the application issues subsequent verbs.

Using the Synchronous Entry Point

Only one synchronous verb can be outstanding at any time for each target handle. A synchronous verb fails with the primary return code AP_STATE_CHECK and secondary return code AP_SYNC_PENDING if another synchronous verb for the same target handle is in progress.

Asynchronous Entry Point: ms_async

An application uses ms_async to issue an MS verb asynchronously. The application also supplies a pointer to a callback routine. CS Linux returns control to the application immediately with a returned value that indicates whether verb processing is still in progress or has completed. In most cases, verb processing is still in progress when control returns to the application. In these cases, CS Linux uses the application-supplied callback routine to return the results of the verb processing at a later time. In some cases, verb processing is complete when CS Linux returns control to the application, so CS Linux does not use the application's callback routine.

Function Call

```
unsigned short ms_async(
                        AP_UINT32     target_handle,
                        void *        msvcb,
                        VMV_CALLBACK  comp_proc,
                        AP_CORR       corr
                       );

typedef void (*VMV_CALLBACK) (
                              AP_UINT32   target_handle,
                              void *      msvcb,
                              AP_CORR     corr
                             );

 typedef union ap_corr  {
                         void *          corr_p;
                         AP_UINT32       corr_l;
                         AP_INT32        corr_i;
                        } AP_CORR;
```

For more information about the parameters in the VMV_CALLBACK structure, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

Supplied Parameters

An application supplies the following parameters when it uses the ms_async entry point:

target_handle
: Identifier for the target CS Linux node. For the REGISTER_*, UNREGISTER_*, and DISCONNECT_MS_NODE verbs, the application supplies the value that was returned on the CONNECT_MS_NODE verb.

  For all other verbs, this parameter is not used; set it to 0 (zero).

msvcb
: Pointer to a Verb Control Block (VCB) that contains the parameters for the verb being issued. The VCB structure for each verb is described in Chapter 3, "Management Services Verbs," on page 15. These structures are defined in the MS API header file **/opt/ibm/sna/include/ms_c.h**.

  Note: The MS VCBs contain many parameters marked as "reserved"; some of these are used internally by the CS Linux 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 CS Linux will not misinterpret any of its internally-used parameters,

and also that your application will continue to work with future CS Linux 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));

comp_proc
The callback routine that CS Linux will call when the verb completes. For more information about the requirements for a callback routine, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

corr An optional correlator for use by the application. This parameter is defined as a C union so that the application can specify any of three different parameter types: pointer, unsigned long, or integer.

CS Linux does not use this value, but passes it as a parameter to the callback routine when the verb completes. This value enables the application to correlate the returned information with its other processing.

Returned Values

The asynchronous entry point returns one of the following values:

AP_COMPLETED
The verb has already completed. The application can examine the parameters in the VCB to determine whether the verb completed successfully. CS Linux does not call the supplied callback routine for this verb.

AP_IN_PROGRESS
The verb has not yet completed. The application can continue with other processing, including issuing other MS verbs, provided that they do not depend on the completion of the current verb. However, the application should not attempt to examine or modify the parameters in the VCB supplied to this verb.

CS Linux calls the supplied callback routine to indicate when the verb processing completes. The application can then examine the VCB parameters.

Using the Asynchronous Entry Point

When using the asynchronous entry point, note the following:

- If an application specifies a null pointer in the *comp_proc* parameter, the verb will complete synchronously (as though the application issued the verb using the synchronous entry point).
- If the call to `ms_async` is made from within an application callback, specifying a null pointer in the *comp_proc* parameter is not permitted. In such cases, CS Linux rejects the verb with primary return code value `AP_PARAMETER_CHECK` and secondary return code value `AP_SYNC_NOT_ALLOWED`.
- The application must not attempt to use or modify any parameters in the VCB until the callback routine has been called.
- Multiple verbs do not necessarily complete in the order in which they were issued. In particular, if an application issues an asynchronous verb followed by a synchronous verb, the completion of the synchronous verb does not guarantee that the asynchronous verb has already completed.

The Callback Routine Specified on the ms_async Entry Point

When using the asynchronous MS API entry point, the application must supply a pointer to a callback routine. CS Linux uses this callback routine both for completion of a verb and also for returning MS data and status indications. The application must examine the *opcode* parameter in the VCB to determine which event is contained in the callback routine.

This section describes how CS Linux uses the callback routine and the functions that the callback routine must perform.

Callback Function

```
typedef void (*VMV_CALLBACK) (
                              AP_UINT32   target_handle,
                              void *      msvcb,
                              AP_CORR     corr
                             );

 typedef union ap_corr  {
                         void *          corr_p;
                         AP_UINT32       corr_l;
                         AP_INT32        corr_i;
                        } AP_CORR;
```

Supplied Parameters

CS Linux calls the callback routine with the following parameters:

target_handle
: For MS data and status indications, CS Linux passes the target handle that was supplied with the REGISTER_MS_APPLICATION or REGISTER_NMVT_APPLICATION verb. For completion of verbs, this parameter is undefined.

msvcb One of the following:

- For MS data and status indications, a pointer to a VCB supplied by CS Linux.
- For completion of verbs, a pointer to the VCB supplied by the application. The VCB now includes the returned parameters set by CS Linux.

corr The correlator value supplied by the application. This value enables the application to correlate the returned information with its other processing.

The callback routine need not use all of these parameters (except as described in "Using the Callback Routine for Indications"). The callback routine can perform all the necessary processing on the returned parameters, or it can simply set a variable to inform the MS application that the verb has completed.

Returned Values

The callback function does not return any values.

Using the Callback Routine for Indications

The callback routine supplied with the REGISTER_MS_APPLICATION VCB can receive the following indications:

- FP_NOTIFICATION (if the application requested this information when registering)
- MDS_MU_RECEIVED
- MS_STATUS

The callback routine supplied with the REGISTER_NMVT_APPLICATION VCB can receive the following indications:

- NMVT_RECEIVED (if the application did not request conversion from NMVT-level data to MDS-level data)
- MDS_MU_RECEIVED (if the application requested conversion from NMVT-level data to MDS-level data)
- MS_STATUS

Although the application allocates the VCBs for MS verbs, CS Linux allocates the VCBs for indications. Therefore, the application has access to the VCB information only from within the callback routine; the VCB pointer that CS Linux supplies to the callback routine is not valid outside the callback routine. The application must either complete all the required processing from within the callback routine, or it must make a copy of any VCB data that it needs to use outside this routine.

Processing of indications in the callback routine must fulfill the following additional requirements:

- If an NMVT-level application uses REGISTER_NMVT_APPLICATION to receive incoming NMVTs, it must be capable of receiving a data length of 512 bytes (the maximum NMVT size).
- If an MDS-level application uses REGISTER_NMVT_APPLICATION to receive incoming NMVTs after conversion to MDS_MUs, it must be capable of receiving a data length of 700 bytes, which allows for the maximum NMVT size together with the MDS_MU header information. (This requirement does not apply to an application using REGISTER_MS_APPLICATION to receive MDS_MUs, because the application can specify the maximum data length it can accept, and CS Linux segments the data if necessary.)
- If an MDS-level application uses REGISTER_MS_APPLICATION to receive incoming MDS_MUs, it must be capable of receiving data of length up to the value specified for the *max_rcv_size* parameter on the REGISTER_MS_APPLICATION verb.

Scope of Target Handle

Each application that needs to use MS must issue the CONNECT_MS_NODE verb to obtain its own handle. No two MS applications can use the same MS target handle.

In particular, if the application that issued CONNECT_MS_NODE later forks to create a child process, the child process cannot issue any MS verbs that use the target handle obtained by the parent process. However, the child process can issue another CONNECT_MS_NODE to obtain its own target handle.

Scheduling Asynchronous Events

The method that an application uses to schedule asynchronous events depends on which of the following types of application it is:

Single-threaded applications
: Applications that are based around a single main thread of execution to receive and process requests

Multithreaded applications
: Applications that can have several threads of execution receiving and processing requests

Motif applications
Applications that use the Motif interface and for which the application code consists mainly of callbacks from the Motif libraries

Single-Threaded Applications

To schedule asynchronous events, single-threaded applications use the application scheduling mode.

Application Scheduling Mode

Application scheduling mode provides a way for callback functions to be called from the application's main thread of execution. With this mode of operation, the MS API library has no need for multiple threads or signals.

Application scheduling mode works on the assumption that at the heart of an application there is a scheduling loop which includes a call to `select` or `poll` to determine whether there is work from one or more work sources. Application scheduling provides access to the file descriptor the MS API library uses to communicate with the CS Linux node. The application adds this file descriptor to its main scheduling `select` call, and when `select` indicates events on that file descriptor, the application calls a function to process the events and, if necessary, make the MS API callbacks. For more information about callbacks, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

Note: SNA callbacks are not restricted to the event processor. Callbacks can be made from within any SNA library functions.

To use application scheduling mode, incorporate the following steps into your application code:

1. In the initialization section of the application, indicate that you want to use application scheduling mode by adding the following function call before the first call into any SNA library:

   ```
   SNA_USE_FD_SCHED();
   ```

   The `SNA_USE_FD_SCHED` call has no return value.

2. Write the main scheduling loop to set up the file descriptors and call `select` according to the following logic. Note that `SNA_GET_FD` returns an integer which is either a valid SNA file descriptor or `-1` if there is no open SNA file descriptor. (The return of `-1` indicates either that there has been no call into any SNA library, or that there has been a serious error resulting in the file descriptor being closed.)

   ```
   fd= SNA_GET_FD();
   if (fd != -1)
   {
      FD_SET(fd, &fdset);
   }
   select(nfds, *fdset, *fdset, *fdset, timeout);
   ```

3. If the `select` call returns with an indication of events on the SNA file descriptor, the application simply calls `SNA_EVENT_FD`. This function (which has no return value) processes all the events on the SNA file descriptor. If these events cause any MS verbs to complete, the MS API callback functions will be issued from within `SNA_EVENT_FD`.

   Note that there is not a one-to-one mapping between calls to `SNA_EVENT_FD` and callbacks. There may be no callbacks (if the event does not complete a verb), or there may be many callbacks (because`SNA_EVENT_FD` processes all events before it returns).

   ```
   SNA_EVENT_FD();
   ```

Note:

1. In general, `SNA_GET_FD` returns the same file descriptor each time you call it. However, you should call it before each call to `select` or `poll`, because there are some situations (such as when your application has issued the `fork` call, or in error cases) when the file descriptor may change.
2. The SNA event handler processes internal control messages between CS Linux components, which do not result in a call to your application's callback routine, in addition to processing MS indications and the completion of MS verbs. In addition, if there are multiple MS verbs outstanding, the SNA event handler may call the callback routine for any of these verbs (because the same file descriptor is used for all verbs).

   Because of this, you cannot assume that an event on the SNA file descriptor corresponds to the completion of a particular verb. The correlator information passed to the callback routine is the correct method of checking which verb has completed, or whether this is an indication rather than a verb completion. The callback routine may either handle the verb completion or indication itself, or pass information to the main processing loop to indicate how the verb completion or indication should be processed. For more details of how the callback routine should operate, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

Multithreaded Applications

CS Linux API libraries are available for linking with multithreaded applications. When you develop applications to operate in a multithreaded environment, the following restrictions apply:

- When an application uses the asynchronous entry point, the application is required to maintain the consistency of its data structures when callbacks are invoked. Consistency of data structures can be maintained using the multithreading `lock` or `mutex` facilities. The callbacks are made in the context of a separate thread created and managed from within the CS Linux API library. Since asynchronous callbacks run using a separate thread, the application is not required to provide a source of scheduling to enable the callbacks. Do not use application scheduling mode in a multithreaded application.
- The application must perform any required cleanup processing (for example, issuing UNREGISTER_MS_APPLICATION or UNREGISTER_NMVT_APPLICATION, as appropriate, and issuing DISCONNECT_MS_NODE) before a thread terminates. The MS library does not maintain any correlation between threads and MS verb usage and does not perform this processing automatically when a thread terminates.

Motif Applications

Applications that use the Motif interface, and whose code therefore consists mainly of callbacks from the Motif libraries, are required to add SNA events to the main Xt library scheduling loop. The SNA events enable the CS Linux library to run callbacks in order to process asynchronous verb completions.

Add the following lines to your code before the first call into any SNA library:

```
#include <Xt.h>
int app_context;
...
XtAppInitialize(app_context...)
...
SNA_USE_XT_SCHED(app_context);
```

The SNA_USE_XT_SCHED call has no return values. It calls the XtAppAddInput function to register the SNA work sources. When work subsequently arrives on the SNA file descriptor, the Xt library scheduling loop calls the SNA event handler, which then makes any required API callbacks to the application.

Note: SNA callbacks can also be made from within other calls into the SNA library.

MS API Header File

The header file to be used with MS applications is **/opt/ibm/sna/include/ms_c.h**. This file contains the definitions of the MS API entry points and the MS VCBs. It also includes the common interface header file **/opt/ibm/sna/include/values_c.h**; these two files contain all the constants defined for supplied and returned parameter values at the MS API. The file **values_c.h** also includes definitions of parameter types such as AP_UINT16 that are used in the MS VCBs.

Compiling and Linking the MS Application

Before compiling and linking an MS 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 the application on a 64–bit version of Linux for zSeries.

Applications are compiled with different options in order to select one of the scheduling modes described in “Scheduling Asynchronous Events” on page 10.

Note: Applications that use asynchronous API callbacks must either be built as multithreaded applications or include support for the application scheduled mode. Motif applications must include the code fragment described in “Motif Applications” on page 12.

When compiling your MS application, specify the following option to indicate to the compiler which directory contains the CS Linux header files your application requires:

-I /opt/ibm/sna/include

When linking your MS application, use the following options:

-L Indicates the directory containing one or more libraries to be used when linking the application.

This directory is normally **/opt/ibm/sna/lib**, except for 64–bit applications on Linux for zSeries where it is **/opt/ibm/sna/lib64**.

-l Indicates the name of a library to be used when linking the application.

The command you use to link the application depends on which type of application it is, as described below.

Linking Motif Applications and Applications That Use Application Scheduled Mode

Link with the following options:

-L /opt/ibm/sna/lib -lms -lsna -lpLiS

For a 64–bit application on Linux for zSeries, replace **/opt/ibm/sna/lib** with **/opt/ibm/sna/lib64**.

Note: Check your Motif documentation for information on which libraries and other options are required to link your Motif application.

Linking Multithreaded Applications

For Linux, link with the following options:

-L /opt/ibm/sna/lib -lms -lsna_r -lpthread -lpLiS

For a 64–bit application on Linux for zSeries, replace **/opt/ibm/sna/lib** with **/opt/ibm/sna/lib64**.

Chapter 3. Management Services Verbs

For each MS verb, this chapter provides the following information:

- Purpose and usage of the verb.
- Verb Control Block (VCB) structure used by the verb. All the VCB structures are defined in the header file **/opt/ibm/sna/include/ms_c.h**.
- Supplied parameters (VCB fields supplied to the verb). For each parameter, the following information is listed:
  - Description
  - Valid values and their meanings
  - Additional information where necessary
- Returned parameters. When a verb completes, it contains the following returned parameters:

  primary_rc
  : This parameter indicates whether the verb completed successfully. If the verb did not complete successfully, this parameter indicates a category of reasons for unsuccessful execution.

  secondary_rc
  : This parameter indicates a specific reason for unsuccessful execution.

  In addition, some verbs have additional returned parameters.

Many of the supplied and returned parameter values are numeric. To simplify coding, make the applications more portable, and make the program source easier to read, these values are represented by symbolic constants defined in the header file **/opt/ibm/sna/include/ms_c.h**. For example, the *opcode* (operation code) parameter for the SEND_MDS_MU verb is the value represented by the symbolic constant `AP_SEND_MDS_MU`.

Because different systems store these values differently in memory, it is important that you use the symbolic constant, and not the numeric value, when setting values for supplied parameters or when testing values of returned parameters. The value shown in the header file may not be in the format recognized by your system.

Note: The MS VCBs contain many parameters marked as "reserved"; some of these are used internally by the CS Linux 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 CS Linux will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future CS Linux 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));
```

CONNECT_MS_NODE

This verb connects an application to a CS Linux node. It returns a handle that should be used on all subsequent calls to the MS entry points.

An application that only sends data using either the TRANSFER_MS_DATA verb or the SEND_MDS_MU verb and does not need to receive MS data or status indications does not need to issue this verb or supply a handle to any subsequent calls to the MS entry points.

VCB Structure

```
typedef struct connect_ms_node
{
AP_UINT16             opcode;           /* Verb operation code          */
unsigned char         reserv2;          /* reserved                     */
unsigned char         format;           /* reserved                     */
AP_UINT16             primary_rc;       /* Primary return code          */
AP_UINT32             secondary_rc;     /* Secondary return code        */
unsigned char         node_name[64];    /* Name of Node to connect to   */
AP_UINT32             target_handle;    /* Handle to identify Node on   */
                                        /*   subsequent verbs           */
} CONNECT_MS_NODE;
```

Supplied Parameters

An application supplies the following parameters when it issues the CONNECT_MS verb:

opcode AP_CONNECT_MS_NODE

node_name

Name of the CS Linux node to connect to. This is an ASCII character string.

If the application will be registering to receive NMVTs to act as a service point for an NMVT-level version of the NetView program, specify the name of a node that owns a direct connection to the NetView host (the node whose PU-SSCP session is used to transmit NMVTs to the NetView program). For more information about NMVT-level programs, see Chapter 1, “Introduction to Management Services,” on page 1.

If any of the following conditions is true, you can set this parameter to all binary zeros (you do not need to specify the node name):

- CS Linux is running with all components on a single Linux computer (not on a LAN).
- The CS Linux LAN contains only one server.
- The application is MDS-level and will be sending and receiving data in MDS_MU format and not in NMVT format.

When the CS Linux LAN has multiple servers and this parameter is set to all binary zeros, the application will be connected to the node on the same server as the application, if available, or to any other available node.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful. If the verb execution was successful, CS Linux also

returns a target handle that the application uses on subsequent MS entry points. If the verb execution was not successful, CS Linux returns parameters to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameters:

primary_rc
AP_OK

secondary_rc
Not used.

target_handle
Returned value for use on future verbs directed to this node.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_NODE_NAME
The *node_name* parameter did not match the name of any CS Linux node.

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

primary_rc
AP_STATE_CHECK

secondary_rc
One of the following:

AP_CONNECT_FAILED
An error occurred in connecting to the node either because the specified node is not active or, if a null node name was specified, because no nodes are active.

AP_INVALID_TARGET_STATE
The target handle used on the call to MS was not set to 0 (zero). For CONNECT_MS_NODE, the target handle must be set to 0 (zero).

AP_SYNC_PENDING
The application used the synchronous entry point to issue this verb, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc
One of the following:

AP_COMM_SUBSYSTEM_NOT_LOADED
The CS Linux software has not been started or has been stopped.

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

DISCONNECT_MS_NODE

This verb disconnects an application from a node, freeing all resources associated with that connection. The node from which the application wants to disconnect is identified by the *target_handle* parameter on the call.

VCB Structure

```
typedef struct disconnect_ms_node
{
AP_UINT16                opcode;       /* Verb operation code          */
unsigned char            reserv2;      /* reserved                     */
unsigned char            format;       /* reserved                     */
AP_UINT16                primary_rc;   /* Primary return code          */
AP_UINT32                secondary_rc; /* Secondary return code        */
} DISCONNECT_MS_NODE;
```

Supplied Parameters

An application supplies the following parameter when it issues DISCONNECT_MS_NODE:

opcode AP_DISCONNECT_MS_NODE

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
AP_OK

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_TARGET_HANDLE
The supplied target handle was not a valid value returned on a previous CONNECT_MS_NODE verb.

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

primary_rc
AP_STATE_CHECK

secondary_rc
One of the following:

AP_INVALID_TARGET_STATE
The application issued DISCONNECT_MS_NODE while CONNECT_MS_NODE or a previous DISCONNECT_MS_NODE was still outstanding.

AP_SYNC_PENDING
The application used the synchronous entry point to issue this verb, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

AP_VERB_IN_PROGRESS
The application issued DISCONNECT_MS_NODE while a previous asynchronous MS verb was still outstanding.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute successfully because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

REGISTER_MS_APPLICATION

The REGISTER_MS_APPLICATION verb registers the MS application with CS Linux as an MDS-level application that can receive MDS_MUs. Before issuing this verb, the application must issue CONNECT_MS_NODE to obtain a target handle for the CS Linux node. This handle is a required parameter to the MS entry point for REGISTER_MS_APPLICATION.

An application must always issue this verb using the asynchronous MS entry point and supply a callback routine. CS Linux uses this callback routine to return received MDS_MUs to the application. (For more information about the MS entry points, see Chapter 2, "Writing MS Applications," on page 5.)

VCB Structure

```
typedef struct register_ms_application
{
   AP_UINT16          opcode;             /* Verb operation code               */
   unsigned char      reserv2;            /* reserved                          */
   unsigned char      format;             /* reserved                          */
   AP_UINT16          primary_rc;         /* Primary return code               */
   AP_UINT32          secondary_rc;       /* Secondary return code             */
   unsigned char      ms_appl_name[8];    /* MS application name               */
   unsigned char      ms_category[8];     /* MS category                       */
   AP_UINT16          max_rcv_size;       /* Maximum size that can be received */
} REGISTER_MS_APPLICATION;
```

Supplied Parameters

The application supplies the following parameters when it issues the REGISTER_MS_APPLICATION verb:

opcode AP_REGISTER_MS_APPLICATION

ms_appl_name
A name identifying this application. An application can register more than once using different application names. The name has the following requirements:

- It cannot match the name used by any other application that is currently registered as an MS application.
- It cannot be either NODE or UNIX, which are reserved for use by CS Linux components.
- It must be eight characters long; pad on the right with EBCDIC space characters (0x40) if necessary.
- It can be one of the following:
  - An EBCDIC string, using type-1134 characters (uppercase A–Z and numerals 0–9)
  - One of the MS Discipline-Specific Application Programs specified in an appendix of *IBM Systems Network Architecture: Management Services Reference*

ms_category
If the application needs to obtain the name of its focal point for a particular MS category, specify the category name here. If the application

does not need to obtain focal point information, set this parameter to eight binary zeros. The application can register more than once for different MS category names.

The MS category name can be one of the following:

- A user-defined category name, an 8-byte EBCDIC string using type-1134 characters (uppercase A–Z and numerals 0–9)
- One of the category names specified in the MS Discipline-Specific Application Programs table of an appendix of *IBM Systems Network Architecture: Management Services Reference*

Names of either type should be padded to eight bytes with trailing space (`0x40`) characters if necessary.

CS Linux returns details of the focal point using an FP_NOTIFICATION indication on the callback routine that was supplied with REGISTER_MS_APPLICATION. If the focal point subsequently changes, CS Linux sends another FP_NOTIFICATION with the new information.

max_rcv_size
The maximum number of bytes that the application can accept in one message. If an incoming MDS_MU is longer than this size, CS Linux segments it and delivers each segment in a separate MDS_MU_RECEIVED signal.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
`AP_OK`

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
`AP_PARAMETER_CHECK`

secondary_rc
One of the following:

`AP_MS_APPL_NAME_ALREADY_REGD`
Another application is currently registered with the specified name, or the application specified one of the two reserved names, `NODE` and `UNIX`.

`AP_INVALID_APPLICATION_NAME`
The supplied application name contains a character not in the

EBCDIC type-1134 character set, and the name is not one of the MS Discipline-Specific Application Program names.

AP_INVALID_CATEGORY_NAME
The supplied category name contains a character not in the EBCDIC type-1134 character set, and the name is not one of the MS Discipline-Specific Application Program category names.

AP_INVALID_TARGET_HANDLE
The target handle supplied by the entry point used by the verb is not a valid value returned on a previous CONNECT_MS_NODE verb.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

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

primary_rc
AP_STATE_CHECK

secondary_rc

AP_INVALID_TARGET_STATE
The application issued this verb while CONNECT_MS_NODE or DISCONNECT_MS_NODE was outstanding.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc
One of the following:

AP_COMM_SUBSYSTEM_NOT_LOADED
The CS Linux software is not loaded.

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

MDS Support Not Configured: If the verb does not execute because the CS Linux configuration does not allow it, CS Linux returns the following parameter:

primary_rc

AP_FUNCTION_NOT_SUPPORTED
The CS Linux local node is not configured to support MDS-level network management applications. Only NMVT-level applications can be used.

CS Linux does not return a *secondary_rc* when it is not configured for MDS-level support.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

REGISTER_NMVT_APPLICATION

The REGISTER_NMVT_APPLICATION verb registers the MS application with CS Linux as an NMVT-level application that can receive NMVTs. This verb is normally used by an NMVT-level application, but it can also be used by an MDS-level application that can receive NMVTs after they have been converted to MDS_MUs. Before issuing this verb, the application must issue CONNECT_MS_NODE to obtain a target handle for the CS Linux node. This handle is a required parameter to the MS entry point for REGISTER_NMVT_APPLICATION.

An application must always issue this verb using the asynchronous MS entry point and supply a callback routine. CS Linux uses this callback routine to return received NMVTs to the application. For more information about the MS entry points, see Chapter 2, "Writing MS Applications," on page 5.

CS Linux routes an NMVT to this application only if both the destination name and the MS major vector key in the NMVT match the values supplied on this call. For more information, see "NMVT Routing" on page 4.

VCB Structure

```
typedef struct register_nmvt_application
{
   AP_UINT16      opcode;                /* Verb operation code               */
   unsigned char  reserv2;               /* reserved                          */
   unsigned char  format;                /* reserved                          */
   AP_UINT16      primary_rc;            /* Primary return code               */
   AP_UINT32      secondary_rc;          /* Secondary return code             */
   unsigned char  ms_appl_name[8];       /* MS application name               */
   AP_UINT16      ms_vector_key_type;    /* MS vector key accepted by appl    */
   unsigned char  conversion_required;   /* MDS level application requesting  */
                                         /*  MDS_MUs                          */
} REGISTER_NMVT_APPLICATION;
```

Supplied Parameters

An application supplies the following parameters when it issues the REGISTER_NMVT_APPLICATION verb:

opcode AP_REGISTER_NMVT_APPLICATION

ms_appl_name
A name identifying this application. An application can register more than once using different application names. This name has the following requirements:

- It cannot match the name used by any other application that is currently registered to accept the same range of keys as specified by the *ms_vector_key_type* parameter.
- It cannot be either NODE and UNIX, which are reserved for use by CS Linux components.
- It must be eight characters long; pad on the right with EBCDIC space characters (0x40) if necessary.

- It can be one of the following:
  - An EBCDIC string, using type-1134 characters (uppercase A–Z and numerals 0–9)
  - One of the MS Discipline-Specific Application Programs specified in an appendix of *IBM Systems Network Architecture: Management Services Reference*

Incoming NMVTs will be routed to this application only if the value specified in this parameter matches the Destination Application Name (`0x50`) subfield of the MS major vector within the NMVT.

ms_vector_key_type
The MS major vector key or keys accepted by the application. CS Linux routes incoming NMVTs to the application that issued this verb only if the MS major vector key in the NMVT matches the value or values specified here.

Specify one of the following:

0x*nnnn* The 2-byte hexadecimal value of a particular major vector key.

AP_SPCF_KEYS
Accept all major vector keys in the range `0x8061`–`0x8064`. This value is intended for use by an application that is implementing the SNA Service Point Command Facility (SPCF) function; do not use it if your application is not implementing this function. The *ms_appl_name* parameter must not match the application name of any other application that is registered to accept SPCF keys.

AP_ALL_KEYS
Accept all major vector keys. The *ms_appl_name* parameter must not match the application name of any other application that is registered to accept all keys.

An application can issue multiple REGISTER_NMVT_APPLICATION verbs (with the same application name or different application names) to accept NMVTs for more than one key or range of keys.

CS Linux uses both the name and the key to determine which application receives the NMVT. Therefore, two or more applications can register to accept NMVTs for the same range of keys (`AP_SPCF_KEYS` or `AP_ALL_KEYS`), provided they use different application names. However, only one application can accept NMVTs for a specific key. If you specify a particular major vector key, the verb returns an error if another application has already registered to accept NMVTs for the specified key.

conversion_required
Specifies whether the registering application is MDS-level and requires conversion of NMVTs to MDS_MUs. Specify one of the following:

AP_YES The application is MDS-level; NMVTs should be converted to MDS_MUs.

AP_NO The application is NMVT-level; NMVTs should not be converted.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
AP_OK

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
One of the following values:

AP_ALL_APPL_ALREADY_REGISTERED
Indicates one of the following error conditions:

- This application has already registered to accept all keys.
- Another application has registered to accept all keys using the same application name.
- The application registered to accept all keys using one of the two reserved names, NODE and UNIX.

AP_INVALID_APPLICATION_NAME
The supplied application name contains a character not in the EBCDIC type-1134 character set, and the name is not one of the MS Discipline-Specific Application Program names.

AP_INVALID_TARGET_HANDLE
The target handle supplied by the entry point used with the verb is not a valid value returned on a previous CONNECT_MS_NODE verb.

AP_KEY_APPL_ALREADY_REGISTERED
Indicates one of the following error conditions:

- Another application has already registered to accept NMVTs for this specific key. Only one application can register for each key.
- The application registered to accept a specific key using one of the two reserved names NODE and UNIX.

AP_SPCF_APPL_ALREADY_REGD
Indicates one of the following error conditions:

- This application has already registered to accept SPCF keys.
- Another application has registered to accept SPCF keys using the same application name.
- The application registered to accept SPCF keys using one of the two reserved names NODE and UNIX.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this

verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

State Check: If the verb fails to execute because of a state error, CS Linux returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc

AP_INVALID_TARGET_STATE
The application issued this verb while CONNECT_MS_NODE or DISCONNECT_MS_NODE was outstanding.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc
One of the following:

AP_COMM_SUBSYSTEM_NOT_LOADED
The CS Linux software is not loaded.

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

SEND_MDS_MU

An MDS-level application uses this verb to send network management data in MDS_MU format. An MDS-level application can also send data in NMVT format using TRANSFER_MS_DATA. To send alert information, always use TRANSFER_MS_DATA. Do not use SEND_MDS_MU to send alert information.

The application can supply a complete MDS_MU to be sent, or it can supply some of the required subvectors and request CS Linux to add additional subvectors. For more information about the format of MDS_MUs, including the format of the subvectors that CS Linux adds, refer to *IBM Systems Network Architecture: Formats*.

If the destination application is NMVT-level, CS Linux automatically converts the supplied MDS_MU to an NMVT.

An error that occurs while sending the MDS_MU to the destination application is reported to the application in different ways, depending on where it is detected:

- If the CS Linux local node detects an error, it returns an error return code to the SEND_MDS_MU verb.

- If a remote node detects an error, it sends an error MDS_MU. CS Linux returns the error MDS_MU to the application in an MDS_MU_RECEIVED indication, provided the application has registered to receive MDS_MUs.

VCB Structure

```
typedef struct send_mds_mu
{
   AP_UINT16       opcode;               /* Verb operation code          */
   unsigned char   reserv2;              /* reserved                     */
   unsigned char   format;               /* reserved                     */
   AP_UINT16       primary_rc;           /* Primary return code          */
   AP_UINT32       secondary_rc;         /* Secondary return code        */
   unsigned char   options;              /* Verb options                 */
   unsigned char   reserv3;              /* reserved                     */
   unsigned char   originator_id[8];     /* Originator ID                */
   unsigned char   pu_name[8];           /* Physical unit name           */
   unsigned char   reserv4[4];           /* reserved                     */
   AP_UINT16       dlen;                 /* Length of data               */
   unsigned char   *dptr;                /* Data                         */
} SEND_MDS_MU;
```

Supplied Parameters

An application supplies the following parameters when it issues SEND_MDS_MU:

opcode AP_SEND_MDS_MU

options This parameter is a one-byte value, with individual bits used as follows to indicate 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.

Bit 0 Add Date/Time subvector to the data. Set this bit to one of the following values:

0 Requests that CS Linux add the subvector

1 Requests that CS Linux not add the subvector

Bit 1 Add Product Set ID subvector to the data. Set this bit to one of the following:

0 Requests that CS Linux add the subvector. If the application supplies data that already contains a Product Set ID subvector, CS Linux adds its own Product Set ID subvector immediately preceding the existing one.

1 Requests that CS Linux not add the subvector.

Bit 2 Reserved. Must be set to 0.

Bit 3 Log the data in the CS Linux error log file. Set this bit to one of the following:

0 Requests that CS Linux log the data.

1 Requests that CS Linux not log the data.

Bit 4 Specifies whether MS is to use default or direct routing to send the MS data to the destination application. Set this bit to one of the following:

0 Requests that CS Linux use default routing. Specify default routing unless the application has received an

FP_NOTIFICATION indication that describes the destination application and has the *fp_routing* parameter set to AP_DIRECT. For more information, see "FP_NOTIFICATION" on page 39.

1 Requests that CS Linux use direct routing.

Bits 5-7
Reserved. Must be set to 0.

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

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

pu_name
Destination physical unit for this MDS-MU. Set this to one of the following:

A name of a PU
Specify an 8-byte type-A EBCDIC string, padded to the right with EBCDIC spaces (0x40). Applications that use SEND_MDS_MU to respond to MDS_MU_RECEIVED indications that were converted from incoming NMVTs should specify the *pu_name* received in the MDS_MU_RECEIVED indication.

In this case, the *pu_name* must match a *pu_name* specified on the definition of a link station (LS); the MDS_MU is sent over this link station. For more information about defining an LS, refer to *CS Linux Administration Guide*.

All binary zeros
Use this value for MDS_MUs that are to be transported using the normal MDS routing protocols.

dlen Length of the data string supplied by the application.

dptr A pointer to the data string supplied by the application. This data string must contain a complete MDS_MU, except as follows:

- If the application used the *options* parameter to add one or more subvectors, these subvectors can be omitted from the supplied data.
- The *Origin Net ID* and *Origin NAU Name* fields can be set to all EBCDIC spaces (0x40); in this case, CS Linux fills in the appropriate information before sending the data.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
AP_OK

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
One of the following:

AP_INVALID_DATA_SIZE
The length field in the supplied MDS_MU does not correspond to the value in the *dlen* parameter.

AP_INVALID_MDS_MU_FORMAT
The supplied data string does not contain a valid MDS_MU.

AP_INVALID_PU_NAME
CS Linux cannot find an active PU with the name specified by the supplied *pu_name* parameter.

State Check: If the verb fails to execute because of a state error, CS Linux returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
One of the following:

AP_SSCP_PU_SESSION_NOT_ACTIVE
The application specified a PU name, but no session exists between this PU and an SSCP.

AP_SYNC_PENDING
This verb was issued using the synchronous entry point, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

MDS Support Not Configured: If the verb does not execute because the CS Linux configuration does not allow it, CS Linux returns the following parameter:

primary_rc

AP_FUNCTION_NOT_SUPPORTED
The CS Linux local node is not configured to support MDS-level network management applications. Only NMVT-level applications can be used.

CS Linux does not return a *secondary_rc* when it is not configured for MDS-LEVEL support.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

TRANSFER_MS_DATA

This verb is used by:

- NMVT-level applications to respond to previously received NMVT requests and to send unsolicited NMVTs
- NMVT-level and MDS-level applications to send unsolicited NMVTs (such as alert information)

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

VCB Structure

```
typedef struct transfer_ms_data
{
   AP_UINT16       opcode;                /* Verb operation code              */
   unsigned char   data_type;             /* Type of data supplied by appl    */
   unsigned char   format;                /* reserved                         */
   AP_UINT16       primary_rc;            /* Primary return code              */
   AP_UINT32       secondary_rc;          /* Secondary return code            */
   unsigned char   options;               /* Verb options                     */
   unsigned char   reserv3;               /* reserved                         */
   unsigned char   originator_id[8];      /* Originator ID                    */
   unsigned char   pu_name[8];            /* Physical unit name               */
   unsigned char   reserv4[4];            /* reserved                         */
   AP_UINT16       dlen;                  /* Length of data                   */
   unsigned char   *dptr;                 /* Data                             */
} TRANSFER_MS_DATA;
```

Supplied Parameters

The application supplies the following parameters when it issues the TRANSFER_MS_DATA verb:

opcode SV_TRANSFER_MS_DATA

data_type
Specify one of the following values:

SV_NMVT
The data contains a complete NMVT. CS Linux converts the data to MDS_MU or CP_MSU format if the data contains an alert and the alert is to be sent to an MDS-level or migration-level focal point.

An application that is responding to an NMVT_RECEIVED indication must supply a complete NMVT and must use the value SV_NMVT to indicate this.

SV_ALERT_SUBVECTORS
The data contains MS subvectors in the SNA-defined format for an alert major vector. CS Linux adds an NMVT header and an alert major vector header. CS Linux converts the data to MDS_MU or CP_MSU format if the alert is to be sent to an MDS-level or migration-level focal point.

SV_USER_DEFINED
The data contains a complete NMVT request unit. CS Linux always logs the data but does not send it to any focal point.

SV_PDSTATS_SUBVECTORS
The data contains problem determination statistics. CS Linux always logs the data but does not send it to any focal point.

options 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 the *data_type* parameter is set to SV_USER_DEFINED.)

Bit 0 Add Date/Time subvector to the data. Set this bit to one of the following values:

0 Requests that CS Linux add the subvector

1 Requests that CS Linux not add the subvector

Bit 1 Add Product Set ID subvector to the data. Set this bit to one of the following:

0 Requests that CS Linux add the subvector. If the application supplies data that already contains a Product Set ID subvector, CS Linux adds its own Product Set ID subvector immediately preceding the existing one.

1 Requests that CS Linux not add the subvector.

Bit 2 Send the data to the focal point or the PU specified by the *pu_name* parameter if this verb is being used to send a reply to a previously received NMVT. Set this bit to one of the following:

0 Requests that CS Linux send the data

1 Requests that CS Linux not send the data

Bit 3 Log the data in the CS Linux error log file. Set this bit to one of the following:

0 Requests that CS Linux log the data.

1 Requests that CS Linux not log the data.

Bits 4–7
Reserved. Must be set to 0.

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

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

pu_name
Destination physical unit for this NMVT. Set this to one of the following:

A name of a PU
Specify an 8-byte type-A EBCDIC string, padded to the right with EBCDIC spaces (`0x40`).

Applications that use TRANSFER_MS_DATA to respond to NMVT_RECEIVED indications should specify the *pu_name* received in the NMVT_RECEIVED indication.

Applications that send unsolicited alerts normally should not specify a *pu_name* (they should set this parameter to all binary zeros) unless the application expressly wishes the alert data to be sent to a specific physical unit. In this case, the *pu_name* must match a *pu_name* specified on the definition of a link station (LS); the NMVT is sent over this link station. For more information about defining an LS, refer to *CS Linux Administration Guide*.

All binary zeros
To specify no *pu_name*. The data contained in TRANSFER_MS_DATA verbs that have the *data_type* parameter set to `SV_NMVT` and all binary zeros specified for the *pu_name* parameter are sent over the default PU session if it is available.

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 CS Linux to add one or more subvectors to the supplied data, the total length after addition of any required headers and 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, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameters:

primary_rc
`SV_OK`

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
SV_PARAMETER_CHECK

secondary_rc
One of the following:

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

AP_INVALID_DATA_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 CS Linux should add one or more subvectors to it, but the added headers and subvectors increased the data size beyond 512 bytes.

AP_INVALID_PU_NAME
CS Linux could not find an active PU with the name specified by the supplied *pu_name* parameter.

State Check: If the verb fails to execute because of a state error, CS Linux returns the following parameters:

primary_rc
SV_STATE_CHECK

secondary_rc
One of the following:

AP_SYNC_PENDING
This verb was issued using the synchronous entry point, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

SV_SSCP_PU_SESSION_NOT_ACTIVE
The application requested CS Linux to send data by setting bit 2 of the *options* parameter to 0, but the session to the appropriate PU was not active.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute successfully because the CS Linux software is not active, CS Linux returns one of the following parameters:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

SV_UNEXPECTED_DOS_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

UNREGISTER_MS_APPLICATION

The UNREGISTER_MS_APPLICATION verb indicates to CS Linux that this application, which previously registered to receive MDS_MUs, no longer wants to receive them. After this verb completes successfully, CS Linux no longer sends any received MDS_MUs to the application.

Before terminating, an application should always issue UNREGISTER_MS_APPLICATION for all its registered application names, followed by DISCONNECT_MS_NODE.

VCB Structure

```
typedef struct unregister_ms_application
{
   AP_UINT16       opcode;           /* Verb operation code                */
   unsigned char   reserv2;          /* reserved                           */
   unsigned char   format;           /* reserved                           */
   AP_UINT16       primary_rc;       /* Primary return code                */
   AP_UINT32       secondary_rc;     /* Secondary return code              */
   unsigned char   ms_appl_name[8];  /* MS application name                */
} UNREGISTER_MS_APPLICATION;
```

Supplied Parameters

The application supplies the following parameters when it issues UNREGISTER_MS_APPLICATION:

opcode AP_UNREGISTER_MS_APPLICATION

ms_appl_name
The name identifying the application that is unregistering. This must be a name that the application has previously specified using REGISTER_MS_APPLICATION. The string must be eight characters long; pad on the right with EBCDIC space characters (0x40) if necessary.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
AP_OK

CS Linux does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

When a verb does not execute successfully, CS Linux 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, CS Linux returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
One of the following:

AP_INVALID_TARGET_HANDLE
The supplied target handle was not a valid value returned on a previous CONNECT_MS_NODE verb.

AP_MS_APPL_NAME_NOT_REGD
The application has not previously issued REGISTER_MS_APPLICATION with the application name specified on this verb.

State Check: If the verb fails to execute because of a state error, CS Linux returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
One of the following:

AP_INVALID_TARGET_STATE
The application issued this verb while CONNECT_MS_NODE or DISCONNECT_MS_NODE was outstanding.

AP_SYNC_PENDING
This verb was issued using the synchronous entry point, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

MDS Support Not Configured: If the verb does not execute because the CS Linux configuration does not allow it, CS Linux returns the following parameter:

primary_rc

AP_FUNCTION_NOT_SUPPORTED
The CS Linux local node is not configured to support MDS-level network management applications. Only NMVT-level applications can be used.

CS Linux does not return a *secondary_rc* when it is not configured for MDS-LEVEL support.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

UNREGISTER_NMVT_APPLICATION

The UNREGISTER_NMVT_APPLICATION verb indicates to CS Linux that this application, which previously registered to receive NMVTs for a given application name, no longer wants to receive them for that name.

If the application used the same application name in multiple REGISTER_NMVT_APPLICATION verbs to accept different types of NMVTs, unregistering this name means that the application no longer receives any of these NMVTs. However, if the application registered using more than one name, it continues to receive NMVTs of the types specified for any remaining application names.

Before terminating, an application should always issue UNREGISTER_NMVT_APPLICATION for all its registered application names, followed by DISCONNECT_MS_NODE.

VCB Structure

```
typedef struct unregister_nmvt_application
{
   AP_UINT16       opcode;           /* Verb operation code                */
   unsigned char   reserv2;          /* reserved                           */
   unsigned char   format;           /* reserved                           */
   AP_UINT16       primary_rc;       /* Primary return code                */
   AP_UINT32       secondary_rc;     /* Secondary return code              */
   unsigned char   ms_appl_name[8];  /* MS application name                */
} UNREGISTER_NMVT_APPLICATION;
```

Supplied Parameters

An application supplies the following parameters when it issues UNREGISTER_NMVT_APPLICATION:

opcode AP_UNREGISTER_NMVT_APPLICATION

ms_appl_name
The name identifying the application that is unregistering. This must be a name that the application has previously specified using REGISTER_NMVT_APPLICATION. The string must be eight characters long; pad on the right with EBCDIC space characters (0x40) if necessary.

Returned Parameters

After the verb executes, CS Linux returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, CS Linux returns the following parameter:

primary_rc
AP_OK

CS Linux does not return a *secondary_rc* when the verb executes successfully.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
One of the following:

AP_APPL_NOT_REGISTERED
The application has not previously issued REGISTER_NMVT_APPLICATION with the application name specified on this verb.

AP_INVALID_TARGET_HANDLE
The supplied target handle was not a valid value returned on a previous CONNECT_MS_NODE verb.

State Check: If the verb fails to execute because of a state error, CS Linux returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
One of the following:

AP_INVALID_TARGET_STATE
The application issued this verb while CONNECT_MS_NODE or DISCONNECT_MS_NODE was outstanding.

AP_SYNC_PENDING
This verb was issued using the synchronous entry point, but another synchronous verb was in progress for this target handle. Only one synchronous verb can be in progress on a particular target handle at any time.

AP_SYNC_NOT_ALLOWED
The application used the synchronous MS entry point to issue this verb within a callback routine. The application must use the asynchronous entry point to issue any verb from a callback routine.

CS Linux Software Not Active: If the verb does not execute because the CS Linux software is not active, CS Linux returns the following parameter:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The CS Linux software has failed.

CS Linux does not return a *secondary_rc* when the CS Linux software is not active.

System Error: If the verb does not execute because of a system error, CS Linux returns the following parameters:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
An operating system call failed during processing of the verb.

secondary_rc
The return code from the operating system call. For the meaning of this return code, check the returned value in the file **/usr/include/sys/errno.h**.

Chapter 4. Management Services Indications

For each indication, this chapter provides the following information:

- Purpose of the indication, and the conditions in which CS Linux returns it to an application.
- Description of the indication. For consistency, the term verb control block (VCB) is used to describe the indications, although this structure is not associated with a verb issued by the application. All the VCB structures are defined in the header file **/opt/ibm/sna/include/ms_c.h**.
- For each parameter in the VCB structure, the following information is listed:
  - Description
  - Values that can be returned and their meanings
  - Additional information where necessary

Many of the supplied and returned parameter values are numeric. To simplify coding, make the applications more portable, and make the program source easier to read, these values are represented by symbolic constants defined in the header file **/opt/ibm/sna/include/ms_c.h**. For example, the *opcode* (operation code) parameter for the FP_NOTIFICATION indication is the value represented by the symbolic constant `AP_FP_NOTIFICATION`.

Because different systems store these values differently in memory, it is important that you use the symbolic constant, and not the numeric value, when setting values for supplied parameters or when testing values of returned parameters. The value shown in the header file may not be in the format recognized by your system.

Note: Although the application allocates the VCBs for MS verbs, CS Linux allocates the VCBs for indications. Therefore, the application has access to the VCB information only from within the callback routine; the VCB pointer that CS Linux supplies to the callback routine is not valid outside the callback routine. The application must either complete all the required processing from within the callback routine, or it must make a copy of any VCB data that it needs to use outside this routine.

FP_NOTIFICATION

CS Linux sends this status indication to an MDS-level application that has requested information about the focal point for a particular MS category. The application requests this information by issuing REGISTER_MS_APPLICATION with the name of a particular MS category specified as part of the focal point data string. CS Linux sends FP_NOTIFICATION to inform the application of its current focal point for that category. Each time the focal point changes, CS Linux sends another FP_NOTIFICATION.

This indication is returned using the callback routine that the application supplied on the REGISTER_MS_APPLICATION verb. For more information about the requirements for this callback routine, see “The Callback Routine Specified on the ms_async Entry Point” on page 9.

VCB Structure

```
typedef struct fp_notification
{
   AP_UINT16       opcode;
   unsigned char   reserv2;            /* reserved                            */
   unsigned char   format;             /* reserved                            */
   AP_UINT16       primary_rc;         /* Primary return code                 */
   AP_UINT32       secondary_rc;       /* Secondary return code               */
   unsigned char   fp_routing;         /* routing to use with this focal point */
   unsigned char   reserv1;            /* reserved                            */
   AP_UINT16       fp_data_length;     /* Length of incoming focal point data  */
   unsigned char   *fp_data;           /* Focal point data                    */
} FP_NOTIFICATION;
```

Parameters

CS Linux includes the following parameters when it sends FP_NOTIFICATION to an MDS-level application:

opcode AP_FP_NOTIFICATION

fp_routing
Specifies whether applications should use default or direct routing when sending MDS_MUs to this focal point. Possible values are:

AP_DEFAULT
MDS_MUs should be delivered to the focal point using default routing.

AP_DIRECT
MDS_MUs should be routed on a session directly to the focal point.

fp_data_length
Length of focal point data. This can be up to 78 bytes.

fp_data Focal point data, which consists of the following:

- Focal Point Notification (0xE1) subvector
- Focal Point Identification (0x21) subvector, which contains an *MS Category* subfield. The *MS Category* subfield identifies the category for which the application requested focal point information and contains the following subfields:
  - Focal point network identifier (NETID)
  - Focal point network accessible unit (NAU) name
  - Application ID

When sending an MDS_MU in the MS category associated with this focal point, the application should include the information from these subfields in the MDS_MU to ensure that it is routed to the appropriate focal point. For full details of the information contained in these subvectors, refer to the IBM manual *System Network Architecture: Formats*.

MDS_MU_RECEIVED

CS Linux uses this data indication to route an MDS_MU to an MDS-level application in the following cases:

- A remote MDS-level application has sent an MDS_MU, and this application has used REGISTER_MS_APPLICATION to accept MDS_MUs.

- A remote application has sent an NMVT, and this application has used REGISTER_NMVT_APPLICATION to accept NMVTs after conversion to MDS_MUs. For information about how CS Linux determines which MS application receives an incoming NMVT, see "NMVT Routing" on page 4.

To return the MDS_MU_RECEIVED indication, CS Linux uses the callback routine that the application supplied on the REGISTER_MS_APPLICATION or REGISTER_NMVT_APPLICATION verb. For more information about the requirements for this callback routine, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

VCB Structure

```
typedef struct mds_mu_received
{
   AP_UINT16       opcode;         /* Verb operation code                 */
   unsigned char   reserv2;        /* reserved                            */
   unsigned char   format;         /* reserved                            */
   AP_UINT16       primary_rc;     /* Primary return code                 */
   AP_UINT32       secondary_rc;   /* Secondary return code               */
   unsigned char   first_message;  /* First message for current MDS_MU    */
   unsigned char   last_message;   /* Last message for current MDS_MU     */
   unsigned char   pu_name[8];     /* Physical unit name                  */
   unsigned char   reserv3[8];     /* reserved                            */
   AP_UINT16       mds_mu_length;  /* Length of incoming MDS_MU           */
   unsigned char   *mds_mu;        /* MDS_MU data                         */
} MDS_MU_RECEIVED;
```

Parameters

CS Linux includes the following parameters when it sends the MDS_MU_RECEIVED indication to the MS application:

opcode AP_MDS_MU_RECEIVED

first_message
: Indicates whether this message is the first, or only, message for this MDS_MU. The MDS_MU is normally sent to the application as a single message (both *first_message* and *last_message* are AP_YES). However, if the MDS_MU is larger than the *max_rcv_size* specified when the application issued REGISTER_MS_APPLICATION, CS Linux segments the MDS_MU and sends it to the application as multiple messages. Possible values are:

  AP_YES First or only message for this MDS_MU.

  AP_NO Second or subsequent message for this MDS_MU.

last_message
: Indicates whether this message is the last, or only, message for this MDS_MU. The MDS_MU is normally sent to the application as a single message (both *first_message* and *last_message* are AP_YES). However, if the MDS_MU is larger than the *max_rcv_size* specified when the application issued REGISTER_MS_APPLICATION, CS Linux segments the MDS_MU and sends it to the application as multiple messages. Possible values are:

  AP_YES Last or only message for this MDS_MU.

  AP_NO First or subsequent message for a segmented MDS_MU. At least one more message follows.

pu_name
: If the MDS_MU was converted from an incoming NMVT, this parameter is the name of the physical unit from which the NMVT was received. If the

NMVT requires a response, the application must send the response using the SEND_MDS_MU verb, and must set the *pu_name* parameter of the SEND_MDS_MU to this name.

The MDS_MU was converted from an incoming NMVT only if the application used the REGISTER_NMVT_APPLICATION verb to register itself as an MDS-level application that accepts NMVTs after conversion to MDS_MUs. If the MDS_MU was received from the MDS-level transport mechanism, this parameter is set to binary zeros.

mds_mu_length
Length of MDS_MU data included on this message. This can be a complete MDS_MU or a segment of complete MDS_MU, depending on the *first_message* and *last_message* parameters.

mds_mu
A pointer to the MDS_MU data string.

MS_STATUS

CS Linux sends this status indication to a registered application (either MDS-level or NMVT-level) to inform the application of one of the following changes in the status of the CS Linux system:

- The application's communications path to the CS Linux local node has been lost because the connected node or an associated component is no longer active.
- The CS Linux software has been stopped.

CS Linux returns the MS_STATUS indication on the callback routine that the application supplied to the REGISTER_MS_APPLICATION or REGISTER_NMVT_APPLICATION verb. For more information about the requirements for this callback routine, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

After the application receives the MS_STATUS indication, CS Linux rejects all subsequent verbs using the relevant target handle, except for DISCONNECT_MS_NODE.

VCB Structure

```
typedef struct ms_status
{
   AP_UINT16           opcode;               /* Verb operation code        */
   unsigned char       reserv2;              /* reserved                   */
   unsigned char       format;               /* reserved                   */
   AP_UINT16           primary_rc;           /* Primary return code        */
   AP_UINT32           secondary_rc;         /* Secondary return code      */
   AP_UINT32           status;               /* status being reported      */
   AP_UINT32           dead_target_handle;   /* Handle of dead connection  */
   unsigned char       reserv1[32];          /* reserved                   */

} MS_STATUS;
```

Parameters

CS Linux includes the following parameters when it sends the MS_STATUS indication to the MS application:

opcode AP_MS_STATUS

status

AP_TARGET_HAS_DIED
This value indicates that the connected node or the CS Linux software is no longer running.

dead_target_handle
A null value for this parameter indicates that the CS Linux software on the local computer (where the application is running) has been stopped. All target handles that the application was using are disconnected and are no longer valid.

A non-null value for this parameter indicates the target handle of the failed node. The application should issue DISCONNECT_MS_NODE for this target handle to free the resources associated with it.

The application can attempt to reconnect to a target node by periodically issuing CONNECT_MS_NODE; this call will fail until the target node or the local CS Linux software is restarted.

NMVT_RECEIVED

CS Linux uses this data indication to route an NMVT received from a remote node to an NMVT-level application that has registered to receive NMVTs. For information about how CS Linux determines which MS application receives an incoming NMVT, see "NMVT Routing" on page 4.

This indication is returned using the callback routine that the application supplied on the REGISTER_NMVT_APPLICATION verb. For more information about the requirements for this callback routine, see "The Callback Routine Specified on the ms_async Entry Point" on page 9.

VCB Structure

```
typedef struct nmvt_received
{
   AP_UINT16       opcode;              /* Verb operation code          */
   unsigned char   reserv2;             /* reserved                     */
   unsigned char   format;              /* reserved                     */
   AP_UINT16       primary_rc;          /* Primary return code          */
   AP_UINT32       secondary_rc;        /* Secondary return code        */
   unsigned char   pu_name[8];          /* Physical unit name           */
   unsigned char   reserv3[6];          /* reserved                     */
   AP_UINT16       nmvt_length;         /* Length of incoming NMVT      */
   unsigned char   *nmvt;               /* NMVT data                    */
} NMVT_RECEIVED;
```

Parameters

CS Linux includes the following parameters when it sends the NMVT_RECEIVED indication to the MS application:

opcode AP_NMVT_RECEIVED

pu_name
Name of the physical unit from which the NMVT originated. This is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

If the incoming NMVT requires a response, the application must send the response using TRANSFER_MS_DATA and must set the *pu_name* parameter of TRANSFER_MS_DATA to the *pu_name* returned here.

nmvt_length
Length of NMVT data, which can be up to 512 bytes.

nmvt Full NMVT, containing MS major vector of the type or types specified on the REGISTER_NMVT_APPLICATION.

Appendix A. MS Function Sets

This appendix provides information about the SNA MS function sets that the CS Linux MS API supports. For more information about these function sets, refer to the IBM manual *Systems Network Architecture: APPN Architecture Reference*.

Base Function Sets

The CS Linux MS API supports the following base function sets:

- Management Services—Multiple-Domain Support (MDS)
  - 150 SNA/MS MDS Common Base
  - 151 SNA/MS MDS End Node Support
  - 152 SNA/MS MDS Network Node Support
- Management Services—MS Capabilities Function Set
  - 160 SNA/MS MS_CAPS Base End Node Support
  - 161 SNA/MS MS_CAPS Have a Backup or Implicit Focal Point
  - 163 SNA/MS MS_CAPS Base Network Node Support
- Management Services—Entry Point Alert Function Set
  - 170 SNA/MS MS EP Alert Base Subset

Optional Function Sets

The CS Linux MS API supports the following optional function sets:

- Management Services—MS Capabilities Function Set
  - 162 SNA/MS MS_CAPS Be a Sphere of Control (SOC) End Node
  - 164 SNA/MS MS_CAPS Have a Subarea FP
- Management Services—Entry Point Alert Function Set
  - 171 SNA/MS Problem Diagnosis Data in Alert
  - 174 SNA/MS Operator Initiated Alert
  - 175 SNA/MS Qualified Message Data in Alert
  - 176 SNA/MS Self-Defining Message Text Subvector in Alert
  - 177 SNA/MS LAN Alert
  - 178 SNA/MS SDLC/LAN LLC Alert
  - 179 SNA/MS X.21 Alert
  - 181 SNA/MS X.25 Alert
  - 182 SNA/MS Held Alert for CPMS

Function Sets Not Supported

CS Linux MS does not provide support for the following function sets:

- Management Services—File Services (option sets 1500, 1501).
- Management Services—Change Management (option sets 1510–1518).
- Management Services—Operations Management (option sets 1520, 1521). Option set 1520, SNA/MS Common Operations Services, is implemented by the CS Linux Service Point Command Facility.

Function Sets Not Supported

Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in Communications Server for Linux enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard, when these devices are supported by the underlying operating system
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in this product. Consult the assistive technology documentation for specific information when using such products to access Communications Server for Linux interfaces.

Appendix C. Notices

IBM may not offer all of the products, services, or features discussed in this document. 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:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, North Carolina 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 measurement 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. _enter the year or years_. All rights reserved.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
Advanced Peer-to-Peer Networking
AIX
Application System/400
AS/400
CICS
DATABASE 2
DB2
Enterprise System/3090
Enterprise System/4381
Enterprise System/9000
ES/3090
ES/9000
eServer
IBM
IBMLink
IMS
Language Environment
MQSeries
MVS
MVS/ESA
MVS/XA
NetView
Operating System/2
Operating System/400
OS/2
OS/400
PowerPC
PowerPC Architecture
S/390
SAA
SP
System/370
System/390
Systems Application Architecture
VSE/ESA
VTAM
WebSphere
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies:

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

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

Intel is a trademark of Intel Corporation.

Linux is a trademark of Linus Torvalds.

RedHat and RPM are trademarks of Red Hat, Inc.

SuSE Linux is a trademark of SuSE Linux AG.

UnitedLinux is a trademark of UnitedLinux LLC.

Microsoft, Windows, Windows NT, Windows 2003, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and 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:

- CS Linux, Version 6.2
- Host Publisher
- Systems Network Architecture (SNA)
- Host configuration
- Virtual Telecommunications Access Method (VTAM®)
- Advanced Program-to-Program Communication (APPC)
- Programming
- Other IBM networking topics

For books in the CS Linux library, brief descriptions are provided. For other books, only the titles, order numbers, and, in some cases, the abbreviated title used in the text of this book are shown here.

CS Linux Version 6.2 Publications

The CS Linux library comprises the following books. In addition, softcopy versions of these documents are provided on the CD-ROM. See *IBM CS 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 CS Linux Quick Beginnings* (GC31-6768-00 and GC31-6769-00)

  This book is a general introduction to CS Linux, including information about supported network characteristics, installation, configuration, and operation.
- *IBM CS Linux Administration Guide* (SC31-6771-00)

  This book provides an SNA and CS Linux overview and information about CS Linux configuration and operation.
- *IBM CS Linux Administration Command Reference* (SC31-6770-00)

  This book provides information about SNA and CS Linux commands.
- *IBM CS Linux CPI-C Programmer's Guide* (SC31-6774-00)

  This book provides information for experienced "C" or Java™ programmers about writing SNA transaction programs using the CS Linux CPI Communications API.
- *IBM CS Linux APPC Programmer's Guide* (SC31-6773-00)

  This book contains the information you need to write application programs using Advanced Program-to-Program Communication (APPC).
- *IBM CS Linux LUA Programmer's Guide* (SC31-6776-00)

  This book contains the information you need to write applications using the Conventional LU Application Programming Interface (LUA).
- *IBM CS Linux CSV Programmer's Guide* (SC31-6775-00)

  This book contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).
- *IBM CS Linux MS Programmer's Guide* (SC31-6777-00)

This book contains the information you need to write applications using the Management Services (MS) API.

- *IBM CS Linux NOF Programmer's Guide* (SC31-6778-00)

  This book contains the information you need to write applications using the Node Operator Facility (NOF) API.
- *IBM CS Linux Diagnostics Guide* (GC31-6779-00)

  This book provides information about SNA network problem resolution.
- *IBM CS Linux APPC Application Suite User's Guide* (SC31-6772-00)

  This book provides information about APPC applications used with CS Linux.
- *IBM Communications Server for Linux Glossary* (GC31-6780-00)

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

Publications for Host Publisher

The following books contain information about the Host Publisher feature that is included with CS Linux:

- *User's Guide for IBM Host Publisher, Version 2* (GC31-8728)
- *Planning and Installation Guide for Host Publisher, Version 2 for Windows NT®, AIX® and Solaris* (SC31-8730)

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)
- *APPN Architecture and Product Implementations Tutorial* (GG24-3669)

Host Configuration Publications

The following books contain information about host configuration:

- *ES/9000, ES/3090 IOCP User's Guide Volume A04* (GC38-0097)
- *3174 Establishment Controller Installation Guide* (GG24-3061)
- *3270 Information Display System 3174 Establishment Controller: Planning Guide* (GA27-3918)
- *OS/390 Hardware Configuration Definition (HCD) User's Guide* (SC28-1848)

VTAM Publications

The following books contain information about VTAM:

- *VTAM V4R4 Network Implementation Guide* (SC31-8370)
- *VTAM V4R4 Diagnosis* (LY43-0078)
- *VTAM V4R4 Resource Definition Reference* (SC31-8377)

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)
- *OS/390 Communications Server: APPC Application Suite User's Guide* (SC31-8085)

Programming Publications

The following books contain information about programming:

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

Other IBM Networking Publications

The following books contain information about other topics related to CS Linux:

- *Advanced Data Communications for Stores: Programming Reference and Operations Manual* (SH20-2406)
- *Local Area Network Concepts and Procedures* (SK2T-1306)
  - Volume 1 (SG24-4753)
  - Volume 2 (SG24-4754)
  - Volume 3 (SG24-4755)
  - Volume 4 (SG24-4756)
- *IBM Network Control Program Resource Definition Guide* (SC30-3349)
- *IBM Netview Operations* (SC30-3364)

Index

R

S

T

U

V

Communicating Your Comments to IBM

If you especially like or dislike anything about this document, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

Please send your comments to us in either of the following ways:

- If you prefer to send comments by FAX, use this number: 1+919-254-9823
- If you prefer to send comments electronically, use this address:
  - comsvrcf@us.ibm.com
- If you prefer to send comments by post, use this address:

  International Business Machines Corporation
  Attn: Globalization Services
  P.O. Box 12195, 3039 Cornwallis Road
  Department G7IA, Building 500/C10A
  Research Triangle Park, North Carolina 27709-2195

Make sure to include the following in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies

Program Number: 5724-i33 and 5724-i34

Printed in USA

SC31-6777-00