What's new in IBM Communications Server for Linux V6.2.1

This document describes the maintenance release changes for version 6.2.1.0 of the Communications Server for Linux (Product No. 5724-i33, CS Linux) and the Communications Server for Linux on zSeries (Product No. 5724-i34, CS Linux on zSeries). Descriptions apply to both Communications Server for Linux and Communications Server for Linux on zSeries products.

The changes provided in this maintenance release for Communications Server for Linux V6.2.1.0 incorporate:

1. 2.6 Kernel Support - Updated Linux operating system support
2. Linux on Power - Additional platform support and packaging
3. AIX Remote API Client - Additional platform and packaging
4. Primary LU support - LUA programming interface to support Primary LU 0 flows
5. Remote API Client updates - Updates provided for the clients
6. LUA Programmer's Guide updates - Additional Primary LU interface
7. Administration Guide updates - Additional commands to define and query HPR timer parameters
8. NOF Programmer's Guide updates - Additional verbs to define and query HPR timer parameters

This new function is described in more detail below.

1. 2.6 Kernel Support

CS Linux and CS Linux on zSeries now supports server installations on the 2.6 Linux kernel distributions from Red Hat and SuSE. The distributions supported for the server installations are RHEL 4 and SLES 9-SP1. When installing the CS Linux product on Intel system, the kernel must be a 32-bit Linux distribution. When installing CS Linux on a Linux on Power (OpenPower or Power5) system, the kernel must be a 64-bit Linux distribution. The CS Linux on zSeries product may be installed on a 31-bit or 64-bit Linux kernel. However, in the future, Linux distributions for the zSeries platform will be for 64-bit kernel only. Applications can use 32-bit or 64-bit libraries.

For the Remote API Client, which has had 2.6 Linux kernel distribution support, client SNA applications can call only 32-bit libraries on Intel platforms. Thus, if the client application is running on a x86_64 Linux distribution, only the 32-bit libraries can be used. For Linux on zSeries and Linux on Power platforms, client SNA applications can use either 32-bit or 64-bit libraries.

Linux kernel distributions that are supported on the various platforms are mapped in the table below:

Table 1. Supported Linux kernel distributions

Platform	**RHAS 2.1**	**SLES 8**	**RHEL 3**	**SLES 9**	**RHEL 4**	**AIX V5**
Intel						
Client (i686 - 32 bit)	x	x	x	x	x	
Client (x86_64), 32-bit API		x	x	x	x	
Server (i686 - 32 bit)	x	x	x	x	x	
OpenPower or Power 5 (64-bit kernel, ppc64)						

Table 1. Supported Linux kernel distributions (continued)

Platform	RHAS 2.1	SLES 8	RHEL 3	SLES 9	RHEL 4	AIX V5
Client				x	x	
Server				x	x	
Linux on zSeries (s390, 31-bit)						
Client		x	x	x	x	
Server		x	x	x*	x*	
Linux on zSeries (s390x, zSeries, 64-bit)						
Client		x	x	x	x	
Server		x	x	x	x	
AIX						
Client - 32 bit						x
Client - 64 bit						x

* indicates support that may be deprecated in the future Linux releases

2. Linux on Power

The CS Linux product (5724-i33) now has multi-platform support that includes the CS Linux server for Linux on Power platforms, which are the OpenPower servers and the Power5 pSeries servers. When installing the server, the support for these platforms requires either RHEL 4 or SLES 9-SP1. To install a CS Linux server on a Linux on Power, refer to the README.ppc64.xx_yy (where xx_yy is the locale for the system).

The CS Linux Remote API client for Linux on Power is shipped with the CS Linux and the CS Linux for zSeries products. The new client support is found in the /ibm-commserver-clients/linux-ppc64 directory in the install disk. The Remote API client runs in ″User Space″ on Linux and is not kernel dependent. To install the client, refer to the /ibm-commserver-clients/linux-ppc64/README.

3. AIX Remote API Client

Communications Servers running in a domain of servers supporting Remote API clients can now provide support for SNA applications running on AIX V5 platforms. When writing applications for the AIX platform, you should use the same description for compiling and linking found in the Communications Server for AIX (http://www.ibm.com/software/network/commserver/library/aix).

The connection between the AIX client and the Linux servers will be over TCP/IP. Any SNA application on AIX that uses the CPI-C, APPC, LUA and some NOF interfaces for the Communications Server for AIX V5 and above, can use this AIX Remote API client. The older CS/AIX V4.2 interfaces are not supported by the Remote API Client.

4. Primary LU Support

LUA applications usually connect to host mainframes as secondary LUs. This means that the definition for the sessions are controlled by the host application which sends the BIND to start a session. Communications Server now has the ability to act as a Primary LU to downstream SNA dependent devices over LAN interfaces using the RUI_INIT_PRIMARY interface. Using this interface, an application can connect downstream dependent LU sessions without the need for a host mainframe.

To use Primary LU applications, the node must be configured with downstream LU (or a Downstream PU template) that have a host LU name of #PRIRUI#. This will indicate to the server that the applications using RUI_INIT_PRIMARY will control these PUs and the LU resources assigned to them. The PUs can only be used on LAN ports. Refer to the LUA Programming Guide updates (below) for the interface information for programming applications to use Primary LU support.

5. Remote API Clients Updates

5.1 Name Change

The Communications Server Remote API Client will now show the name ″IBM Remote API Client″ in place of the previous name.

5.2 Linux and AIX Client Installation

If you are installing the Communications Server server or client on a system that already has a previous version of the code, you must un-install the server or client first. Configuration files will not be deleted or modified by the un-install and install process.

5.3 Windows clients

If you are installing the Communications Server client on a system that already has a previous version of the code, you must take the following steps to retain your current configuration:

- Stop the Communications Server client service by issuing **net stop sxclient**. This command must be typed as seen (not translated).
- Close the Windows client monitor icon (usually found in the lower right portion of the desktop).
- Install the product without removing the previous version.

If you remove the previous Windows client that is installed, the configuration information is deleted from the Windows registry database and the configuration information must be entered again after the later version is installed

The IBM Remote API Client for Windows now includes a the diagnostic tool for service, **snagetpd.exe**. This tool creates a compact file, **snapd.exe**, that is self-extracting. The problem determination file contains:

- Relevant registry contents
- Directory contents of everything in and below the install directory
- File version information of all installed binaries
- Locations of all log and trace files
- Output from commands ’ver’, ’ipconfig /all’, ’route print’, ’netstat -an’, ’netstat -a’

The tool copies all log and trace files into snapd. When directed by Communications Server Linux service to provide problem determination information, you should send a snapd.exe file It will aid in providing service for reported problems.

APAR fixes that are included for the Windows Remote API Client are:

- LI70604, LI70605 - Installation aborts on some Windows XP systems that do not have one of the default locales set.
- LI70677, LI70678 - WinCPIC application's main thread cannot issue more than one ALLOCATE.

6. LUA Programmer's Guide Updates

The following information are additions to the LUA Programmer's Guide for describing the program interface for RUI_INIT_PRIMARY. You should refer to the LUA Programmer's Guide section for RUI_INIT for any considerations or features that may not be fully described in this section.

6.1 Designing and Writing LUA Applications: SNA Information for RUI Primary

This section contains SNA considerations for writing CS Linux RUI Primary applications for communications with a downstream LU.

This guide does not attempt to explain SNA concepts in detail. If you need specific information about SNA message flows, refer to the documentation for the host application for which you are designing your CS Linux LUA application.

6.2 Responsibilities of the Primary RUI Application

A Primary RUI application has control of both LU-SSCP and PLU-SLU sessions at the Request/Response Unit (RU) level, and can send and receive SNA RUs on these sessions. The PU-SSCP session is internal to CS Linux and the Primary RUI application cannot access it.

Because a Primary RUI application works at the RU level, it has a large degree of control over the data flow to and from the secondary LU. However, it takes greater responsibility than a regular LUA application for ensuring that the SNA messages it sends are valid and that the RU level protocols (for example bracketing and chaining) are used correctly. In particular, note that CS Linux does not attempt to verify the validity of RUs sent by a Primary RUI application.

The Primary RUI application is responsible for:

- Initializing downstream LUs using RUI_INIT_PRIMARY, and terminating them using RUI_TERM
- Processing NOTIFY messages from the secondary LU as secondary applications start and stop
- Processing INIT-SELF and TERM-SELF to activate and deactivate the PLU-SLU session
- Building, sending, receiving and parsing 3270 data stream messages in data RUs
- Implementing RU level protocols (request control, bracketing, chaining, direction)
- Cryptography (if required)
- Compression (if required).

6.3 Restrictions

CS Linux does not support the following for Primary RUI applications:

- Downstream PUs over DLUR
- Dynamically Defined Dependent LUs (DDDLU)
- Sending STSN (to reset sequence numbers, the application should UNBIND and re-BIND the session).

6.4 Configuration Information: RUI_INIT_PRIMARY Link Configuration

For a Primary RUI application communicating with a downstream LU, the only configuration required is the downstream LU (or a Downstream PU template) that as a host LU name of #PRIRUI#.

6.5 RUI_VERBS: RUI_INIT_PRIMARY Verb Control Description

The RUI_INIT_PRIMARY verb establishes the SSCP-LU session for an SNA Primary application that is communicating with a downstream LU. If the RUI application acts as an SNA secondary and communicates with a host LU, it must use RUI_INIT instead of RUI_INIT_PRIMARY.

6.5.1 Supplied Parameters

The application supplies the following parameters (See /opt/ibm/sna/include/lua_c.h):

lua_verb
LUA_VERB_RUI

lua_verb_length
The length in bytes of the LUA verb record. Set this to sizeof(LUA_COMMON).

lua_opcode
LUA_OPCODE_RUI_INIT_PRIMARY

lua_correlator
LUA_OPCODE_RUI_INIT_PRIMARY

lua_luname
The name in ASCII of the LU for which you want to start the session. This must match the name of a downstream LU configured for use with SNA Gateway, or an LU created implicitly from a downstream LU template that as a host LU name of #PRIRUI#.

This parameter must be eight bytes long; pad on the right with spaces, 0x20, if the name is shorter than eight characters.

lua_max_length
The length of a buffer supplied to receive a copy of the ACTLU(+RSP) RU received from the downstream PU. If the application does not need to receive this information, it can specify a null pointer in the lua_data_ptr parameter, in which case it does not need to provide a data buffer.

lua_data_ptr
A pointer to the buffer supplied to receive a copy of the ACTLU(+RSP) RU received from the downstream PU. If the application does not need to receive this information, it can specify a null pointer, and the information will not be returned.

lua_post_handle
A pointer to a callback routine. If the verb completes asynchronously, LUA will call this routine to indicate completion of the verb. For more information, see Designing and Writing LUA Applications.

lua_encr_decr_option

0 Session-level cryptography is not used.

128 Encryption and decryption are performed.

Note: Encryption and decryption are performed by the application program.

Any other value will result in the return code LUA_ENCR_DECR_LOAD_ERROR.

6.5.2 Returned Parameters

LUA always returns the following parameter:

lua_flag2.async
This flag is set to 1 if the verb completed asynchronously, or 0 if the verb completed synchronously. RUI_INIT_PRIMARY will always complete asynchronously, unless it returns an error such as LUA_PARAMETER_CHECK.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

If the verb executes successfully, LUA returns the following parameters.

lua_prim_rc
LUA_OK

lua_sid
A session ID for the new session. This can be used by subsequent verbs to identify this session.

lua_data_length
The length of the ACTLU(+RSP) RU received from the downstream PU. LUA places the data in the buffer specified by lua_data_ptr.

Unsuccessful Execution

If the verb does not complete successfully, LUA returns a primary return code to indicate the type of error and a secondary return code to provide specific details about the reason for the unsuccessful execution.

Please refer to the LUA Programmer's Guide section describing Unsuccessful Execution for RUI_INIT to see the list of possible return codes. In addition to these, the following indications may return for RUI_INIT_PRIMARY specific errors:

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
LUA_DUPLICATE_RUI_INIT_PRIMARY

An RUI_INIT_PRIMARY verb is currently being processed for this session.

6.5.3 Interaction with Other Verbs

The RUI_INIT_PRIMARY verb must be the first LUA verb issued for the session.

Until this verb has completed successfully, the only other LUA verb that can be issued for this session is RUI_TERM, which will terminate a pending RUI_INIT_PRIMARY.

All other verbs issued on this session must identify the session using one of the following parameters from this verb:

- The session ID, returned to the application in the lua_sid parameter
- The LU name, supplied by the application in the lua_luname parameter

6.5.4 Usage and Restrictions

The RUI_INIT_PRIMARY verb completes after an ACTLU positive response is received from the downstream LU. If necessary, the verb waits indefinitely. If the application needs to check the contents of this ACTLU positive response, it can do so by supplying a data buffer on RUI_INIT_PRIMARY (using the lua_max_length and lua_data_ptr parameters) in which CS Linux returns the contents of the received message.

Once the RUI_INIT_PRIMARY verb has completed successfully, this session uses the LU for which the session was started. No other LUA session (from this or any other application) can use the LU until the RUI_TERM verb is issued, or until an LUA_SESSION_FAILURE primary return code is received.

If the RUI_INIT_PRIMARY verb returns with an LUA_IN_PROGRESS primary return code then the Session ID will be returned in the lua_sid parameter. This Session ID is the same as that returned when the verb completes successfully and can be used with the RUI_TERM verb to terminate an outstanding RUI_INIT_PRIMARY verb.

7. Administration Guide Updates

There are 2 new commands for the snaadmin administration tool. These commands are:

define_rtp_tuning
define new timers for tuning HPR session connectivity.

query_rtp_tuning
query HPR path switch timers

To use these commands, issue the help command ″snaadmin -d -h define_rtp_tuning″ or ″snaadmin -d -h query_rtp_tuning″ to see the syntax of the commands. You can also refer to the NOF Programmer's Guide updates for more specific information.

8. NOF Programmer's Guide Updates

8.1 DEFINE_RTP_TUNING

DEFINE_RTP_TUNING specifies parameters to be used when setting up RTP connections. After you issue this verb, the parameters you specify will be used for all future RTP connections until you modify them by issuing a new DEFINE_RTP_TUNING verb.

8.1.1 Supplied Parameters

The application supplies the following parameters (See define_rtp_tuning structure in /opt/ibm/sna/include/nof_c.h):

opcode
AP_DEFINE_RTP_TUNING

path_switch_attempts
Number of path switch attempts to set on new RTP connections. Specify a value in the range 1-255. If you specify 0(zero), CS Linux uses the default value of 6.

short_req_retry_limit
Number of times a Status Request is sent before CS Linux determines that an RTP connection is disconnected and starts Path Switch processing. Specify a value in the range 1-255. If you specify 0(zero), CS Linux uses the default value of 6.

path_switch_times
Length of time in seconds for which CS Linux attempts to path switch a disconnected RTP connection. This parameter is specified as four separate time limits for each of the valid transmission priorities in order: AP_LOW, AP_MEDIUM , AP_HIGH, and AP_NETWORK. Each of these must be in the range 1-65535. The value you specify for each transmission priority must not exceed the value for any lower transmission priority.

If you specify 0(zero) for any of these values, CS Linux uses the corresponding default value as follows:

- 480 seconds (8 minutes) for AP_LOW
- 240 seconds (4 minutes) for AP_MEDIUM
- 120 seconds (2 minutes) for AP_HIGH
- 60 seconds (1 minute) for AP_NETWORK

8.1.2 Returned Parameters

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_INVALID_PATH_SWITCH_TIMES The path_switch_times parameter was not valid; for example, you may have specified a value for one transmission priority that exceeds the value specified for a lower transmission priority.

Common Return Codes lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

8.2 QUERY_RTP_TUNING

QUERY_RTP_TUNING returns information about the parameters that will be used for future RTP connections. This information was previously set up using DEFINE_RTP_TUNING.

8.2.1 Supplied Parameters

The application supplies the following parameters (See query_rtp_tuning structure in /opt/ibm/sna/include/nof_c.h):

opcode
AP_QUERY_RTP_TUNING

8.2.2 Returned Parameters

Successful Execution

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

primary_rc
AP_OK

path_switch_attempts
Number of path switch attempts to set on new RTP connections

short_req_retry_limit
Number of times a Status Request is sent before CS Linux determines that an RTP connection is disconnected and starts Path Switch processing.

path_switch_times
Length of time in seconds for which CS Linux attempts to path switch a disconnected RTP connection. This parameter is specified as four separate time limits for each of the valid transmission priorities in order: AP_LOW, AP_MEDIUM , AP_HIGH, and AP_NETWORK.

Other Conditions

Common Return Codes lists further combinations of primary and secondary return codes that are common to all NOF verbs.