TCP62 for IBM eNetwork Communications Server Version 5 for OS/2 Warp

Introduction to TCP62

IBM TCP62 is an Application Programming Interface (API) that simplifies configuration of AnyNet LU6.2 support over TCP/IP.

The TCP62 configuration can be thought of as a filter on top of the existing Communications Server configuration API. It adds function by adding new and extended verbs that are useful in meeting TCP62 requirements. In particular, since many configuration files are identical except for the local LU names, unique local LU names can be defined dynamically based on the local IP address. Since TCP62 always implies LU 6.2 over IP, SNA and AnyNet configurations can be simplified to one or two parameters.

TCP62 consists of the following four verbs:

- START_TN62

  Builds a Communications Server configuration file based on the parameters passed in the **START_TN62** verb and starts Communications Server using that configuration file

- DEFINE_PARTNER_LU_TN62

  Generates a partner LU name and passes the define verb to the Communications Server node

- DEFINE_LOCAL_LU_TN62

  Generates a local LU name and passes the define verb to the Communications Server node

- STOP_TN62

  Stops Communications Server and immediately ends any communication in progress

The **START_TN62** verb can be used when another subsystem (for example, a CICS client) wants to enable LU6.2 over IP communication. Since START_TN62 may involve starting the underlying communications node, it should be used infrequently due to the potentially long processing time. For example, START_TN62 can be used when the subsystem is initialized and should not be used on a per transaction basis. Similarly, it is expected that both the **DEFINE_LOCAL_LU_TN62** and **DEFINE_PARTNER_LU_TN62** verbs would only be used during subsystem initialization. However, the processing time for these two verbs is much less than for the **START_TN62** verb.

Writing TCP62 Programs

Communications Server TCP62 provides a dynamic link library (DLL) file that handles TCP62 verbs.

TCP62 verbs have a straight forward language interface. Your program fills in fields in a block of memory called a verb control block (VCB). Then it calls the TCP62 DLL and passes a pointer to the VCB. When the program is complete, TCP62 returns, having used and then modified the fields in the VCB. Your program can then read the returned parameters from the VCB.

The following table shows source module usage of supplied header files and libraries needed to compile and link TCP62 programs. Some of the header files may include other required header files.

Table 1. Header File and Library for TCP62

Operating System	Header File	Library	DLL Name
OS/2	TN62API.H	ACS32.LIB	TN62API.DLL

TN62API() Entry Point

This provides a synchronous entry point for issuing the following TCP62 API verbs:

- START_TN62
- STOP_TN62
- DEFINE_LOCAL_LU_TN62
- DEFINE_PARTNER_LU_TN62

Syntax

```
void APIENTRY TN62API(PVOID vcb);
```

Parameter	**Description**
vcb	**Pointer to verb control block**

Returns

No return value. The **primary_rc** and **secondary_rc** fields in the verb control block indicate any error.

Remarks

This is the main synchronous entry point for the TCP62 services API. This call blocks until the verb completes.

Migration Considerations

One common migration scenario involves the Communications Server node communicating outside the scope of TCP62. For example, a user may have previously installed Communications Server and used it for emulator or native SNA communication. In this case, the user will have a default Communications Server response file and Communications Server may be running when START_TN62 is invoked. This section discusses the design and use of the TCP62 API using this scenario.

When START_TN62 creates its response file, it takes as its starting point the active or default response file. Therefore, any Communications Server configuration, such as links, modes, or LUs performed outside of TCP62 is not lost.

AnyNet support, (that is, whether or not a node can use AnyNet IP transport), cannot be dynamically changed in a running node. For example, if a node is running that is not configured to support AnyNet, the **START_TN62** verb cannot dynamically update the running node to support AnyNet and START_TN62 will fail. In this case, Communications Server must be stopped before START_TN62 can complete successfully.

The following node parameters cannot or should not be changed in a running node:

- CP name
- CP alias
- SNA domain name suffix
- AnyNet timer values

If the node is running and START_TN62 is issued with parameters from the above list that are different from those in the running node, the START_TN62 completes successfully. However, the values from the running node will be unchanged and these values will be returned in the START_TN62 VCB. The START_TN62 tells the caller that some values were not used by setting **primary_rc** to 0 and **secondary_rc** to TN62_PARAMETERS_NOT_USED.

Dynamic Name Generation

TCP62 dynamically generates local LU or partner LU names if the input name parameter is a template. That is, if it contains one or more replacement characters (“*”). The name generation algorithm is also used by the SXMAP program in Communications Server.

The following example shows how this algorithm is implemented.

```
static void
  SxMap(unsigned char *generatedName,
        unsigned char *nameTemplate,
        unsigned int   templateLength,
        unsigned long  addr,
        unsigned long  mask)
{
    int I;
    unsigned long host_bits;
    unsigned long bit_pos;
    char chars[] = "0123456789ABCDFGHJKLMNPQRSTVWXYZEIOU@#$.";

   addr = ntohl(addr);
    mask = ntohl(~mask);
    host_bits = 0L;
    bit_pos = 0x00000001;
    for (i = 0; i < 32; i++)
    {
      if (mask & bit_pos)
      {
        host_bits |= (addr & bit_pos);
        bit_pos <<= 1;
      }
      else
      {
        addr >>= 1;
        mask >>= 1;
      }
    }
    for (i = templateLength; i >= 0; i--)
    {
      if (nameTemplate[i] == REPLACEMENT_CHAR)
      {
        generatedName[i] = chars[host_bits & 0x1F];
        host_bits >>= 5;
      }
      else
        generatedName[i] = nameTemplate[i];
    }
    return;
}
```

The algorithm selects bits from **addr** which is a local or remote IP address. The selected bits are those where the corresponding bit in **mask** is 0. The selected bits are then taken in groups of 5, right to left, to generate a character for each replacement character in **nameTemplate**.

For example, if **nameTemplate** = “A*NAME*”, **addr** = 0x13.0x8f.0x22.0xa3 and **mask** = 0xff.0xff.0xff.0x00. The bits selected by **mask** are 0x00.0x00.0x00.0xa3. Since there are two replacement characters in **nameTemplate**, the two groups of five bits are 0x05 and 0x03. Using these as indices in **chars** yields a **generatedName** of “A2NAME4”.

TCP62 API Support

Communications Server supports the following verbs using the TCP62 API.

START_TN62

The **START_TN62** verb starts the Communications Server node.

VCB Structure

```
typedef struct start_tn62
{
     unsigned short   opcode;            /* verb operation code  */
     unsigned char    reserv1[6];        /* reserved             */
     unsigned short   primary_rc;        /* primary return code  */
     unsigned char    reserv2[2];        /* reserved             */
     unsigned long    secondary_rc;      /* secondary return code*/
     unsigned char    reserv3[4];        /* reserved             */
     unsigned char    key[8];            /* key (ASCII)          */
     unsigned char    fqcp_name[17];     /* real fully-qualified */
                                         /* name or a template   */
                                         /* for a fully-qualified*/
                                         /* name                 */
     unsigned char    cp_alias[8];       /* ASCII CP alias       */
     unsigned char    reserv4[3];        /* reserved             */
     unsigned long    ip_address_mask;   /* mask used in dynamic */
                                         /* CP name generation   */
     unsigned short   connection_retry_secs;
                                         /*connection retry count*/
     unsigned short   unacked_dg_retry_secs;
                                         /* unacknowledged data- */
                                         /* gram retry interval  */
     unsigned short   unsent_dg_retry_secs;
                                         /* unsent datagram      */
                                         /* retry interval       */
     unsigned short   inactivity_timer_secs;
                                         /* remote node inactiv- */
                                         /* ity poll interval    */
     unsigned short   connwait_secs;     /* connection wait time */
                                         /* limit                */
     unsigned char    domain_name_suffix[220];
                                         /* domain name suffix   */
} START_TN62;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_START_TN62.
key	Specifies either the master or service key if the keylock feature has been activitated. This is an 8-byte ASCII character string. If the name is less than 8 bytes, it must be padded on the right with ASCII blanks.
fqcp_name	This is either a real fully-qualified CP name or a template for a fully-qualified CP name. If there are no template replacement characters ('*'), it is a real name, otherwise it is a template. The net ID must not contain any template replacement characters and the CP name must not begin with a replacement character. Except for the replacement character, this must be a valid EBCDIC fully-qualified CP name.
cp_alias	The ASCII CP alias for the TCP62 node. If the node is running when the START_TN62 is issued, this will contain the CP alias of the running node on return. If this field is all blanks or nulls, and the node is not running when the START_TN62 is issued, the CP alias is set to the (unqualified) CP name on return.
ip_address_mask	This is the mask to be used in dynamic CP name generation. It is ignored if **fqcp_name** is not a template. The mask is encoded as a big-endian long; i.e., high-order byte first to low-order byte last.

connection_retry_secs

The connection retry count is the maximum time, in seconds, for LU6.2 over TCP/IP to set up a multiprotocol transport network (MPTN) connection over TCP/IP. When an MPTN connection setup fails, Communications Server tries every IP address associated with an LU name in the domain name server or HOSTS file until all the addresses are exhausted or until the time specified is reached.

Specify a value between 1 and 65535 seconds.

Default: 300

If you are unsure about what value to enter, use the default.

unacked_dg_retry_secs

The unacknowledged datagram retry interval is the maximum time, in seconds, that LU6.2 over TCP/IP waits to resend an unacknowledged out-of-band (OOB) or MPTN keepalive datagram.

When expedited data is sent over TCP/IP, this interval is used to help control the delivery of expedited data in congested situations. In SNA, some control messages are sent over TCP/IP, this interval is used to help control the delivery of expedited data in congested situations. In SNA, some control messages are sent as expedited data (for example, messages requesting the right to send data or messages taking down a session). Expedited data is not subject to congestion control and can move ahead of normal, non-expedited data. To assure delivery, AnyNet SNA over TCP/IP might send expedited data as normal data and as an OOB datagram.

Specify a value between 1 and 65535 seconds.

Default: 10

If you are unsure about what value to enter, use the default.

unsent_dg_retry_secs

The unsent data gram retry interval is the maximum time, in seconds, that Communications Server waits for an acknowledgement after sending expedited data on a TCP connection, before sending the data as an out-of-band (OOB) datagram.

When expedited data is sent over TCP/IP, this interval is used to help improve the delivery of expedited data in congested situations. In SNA, some control messages are sent as expedited data (for example, messages requesting the right to send data or messages taking down a session). Expedited data is not subject to congestion control and can move ahead of normal, non-expedited data. To assure delivery, AnyNet SNA over TCP/IP might send expedited data as normal data and as an OOB datagram.

Specify a value between 1 and 65535 seconds.

Default: 3

If you are unsure about what value to enter, use the default.

inactivity_timer_secs

The remote node inactivity poll interval is the number of seconds of inactivity allowed between two partner nodes before LU6.2 over TCP/IP tries to determine whether the partner node is still active.

Type a value between 1 and 65535 seconds.

Default: 30

Setting the interval below 10 seconds might seriously affect system performance.

To calculate how long it takes before an inactive partner is detected:

1. Multiply the value of the **unsent datagram retry interval** by 5.

2. Add the remote node inactivity poll interval value.

The resulting value is the number of seconds it takes to detect an inactive partner.

If you are unsure about what value to enter, use the default.

connwait_secs

The connection waittime limit is the maximum time, in seconds, that LU6.2 over TCP/IP waits to receive a multiprotocol transport network (MPTN) connection or connection response packet after the TCP connection is established. This limit prevents the connecting node from waiting too long for a session partner to send a packet.

Specify a value between 1 and 65535 seconds.

Default: 30

If you are unsure about what value to enter, use the default.

domain_name_suffix

The SNA domain name suffix is used when a domain name is created from the fully-qualified partner LU name.
The SNA domain name suffix is a user-defined domain name suffix created using the hierarchical-naming format recognized by TCP/IP. For example, SNA.IBM.COM is an SNA domain name suffix.

Consult your network administrator to obtain an SNA doman name suffix. The suffix consists of strings concatenated with periods. Each string must be less than or equal to 63 characters, with the total length of less than or equal to 237 characters.

Valid characters for each string are:

The first character must be an alphabetic character (A-Z, a-z).

The last character must be an alphanumeric character (A-Z, a-z, 0-9).

The remaining characters can be alphanumeric characters (A-Z, a-z, 0-9) or the special character (-).

Default: SNA.IBM.COM

Returned Parameters

primary_rc
secondary_rc
See TCP62 Return Codes for details on **primary_rc** and **secondary_rc** verbs.

fqcp_name
The real fully-qualified CP name. If the supplied fqcp_name was a template, it is replaced with the generated name.

STOP_TN62

The **STOP_TN62** verb stops the Communications Server node. Any communications that are in progress will end.

VCB Structure

```
typedef struct stop_tn62
{
     unsigned short   opcode;            /* verb operation code  */
     unsigned char    reserv1[6];;       /* reserved             */
     unsigned short   primary_rc;        /* primary return code  */
     unsigned char    reserv2[2];;       /* reserved             */
     unsigned long    secondary_rc;      /* secondary return code*/
     unsigned char    reserv3[4];        /* reserved             */
     unsigned char    key[8];            /* key (ASCII)          */
} STOP_TN62;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_STOP_TN62.

key Specifies either the master or service key if the keylock feature has been activitated.

This is an 8-byte ASCII character string. If the name is less than 8 bytes, it must be padded on the right with ASCII blanks.

Returned Parameters

primary_rc Success or failure of the verb. The primary and

secondary_rc secondary return codes are always zero on return.

DEFINE_LOCAL_LU_TN62

TCP62 extends the **DEFINE_LOCAL_LU** verb to allow definitions of LU names that are generated dynamically, based on a supplied template, mask, and the local IP address.

VCB Structure

```
typedef struct define_local_lu_tn62
{
     unsigned short   opcode;            /* verb operation code  */
     unsigned char    reserv1[6];;       /* reserved             */
     unsigned short   primary_rc;        /* primary return code  */
     unsigned char    reserv2[2];;       /* reserved             */
     unsigned long    secondary_rc;      /* secondary return code*/
     unsigned char    reserv3[4];        /* reserved             */
     unsigned char    key[8];            /* key (ASCII)          */
     unsigned char    lu_name[8];        /* local Lu name        */
     unsigned char    lu_alias[8];       /* Lu alias (ASCII)     */
     unsigned char    nau_address;       /* network addressable  */
                                         /* unit address         */
     unsigned char    external_support;                          */
                              /*external support for sync point */
                              /*AP_NONE x'00'                    */
                              /*AP_SYNCPT_PROVIDER x'80'         */
                              /*AP_REMOTE_TP_PROVIDER x'40'      */
                              /*AP_SYNCPT_AND_REMOTE_TP x'C0'    */
     unsigned char    host_link_name[8]; /* host link name       */
                                         /* 0 or 1-8 bytes       */
                                         /* (EBCDIC type A)      */
     unsigned char    lu_model_name[7];  /* self-defining dep LU */
                                         /* model name           */
     unsigned char    reserv4[35];       /* reserved             */
     unsigned long    ip_address_mask;   /* mask used in CP name */
     unsigned char    reserv5;           /* reserved             */
     } DEFINE_LOCAL_LU_TN62;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_LOCAL_LU_TN62.

key Specifies either the master or service key if the keylock feature has been activitated.

This is an 8-byte ASCII character string. If the name is less than 8 bytes, it must be padded on the right with ASCII blanks.

lu_name This is either a real LU name or an LU name template. If there are no template replacement characters (‘*’), it is a real name, otherwise it is a template. Except for replacement characters, lu_name must be a valid, EBCDIC LU name. The first character must not be the replacement character.

lu_alias The 8-byte ASCII name used locally for the LU. The name is not sent outside the local node.

nau_address Specifies the network addressable unit (NAU) address of the LU.

You can specify a value from 0 to 254, where:

0 Specifies that the NAU address is not used, and the LU is an independent LU. Sessions among APPN end nodes and network nodes must use independent Lus. An LU type 6.2 is the only type of SNA LU that supports independent sessions. An independent session does not depend on an SSCP (that is, the LU can send a BIND directly without the help of an SSCP).

1-254 Specifies the NAU address of the LU, and that the LU is a dependent LU for sessions to a subarea node. If your network contains a subarea that does not support an independent session from a peripheral node, you will be restricted to a single dependent session between your APPC LU and that subarea. In this case, the LU will need to be assigned a unique NAU address.

An LU’s NAU address is the address used by a subarea node for an LU dependent session. A dependent session is a session that depends on an SSCP to initiate it (that is, an optional INIT_SELF request that flows on the LU to SSCP session, the SSCP sends a CINIT request to a subarea LU, and the subarea LU sends the BIND).

On a given link, every LU that uses a dependent session must be assigned a unique NAU address. Every LU defined for an LUA session, every LU defined for a 3270 session, and every LU defined for a 3270 gateway session, uses a dependent session and must be assigned a unique NAU address.

host_link_name The 8-byte EBCDIC name of the local link station. This logical link can be activiated by specifying this name on the **ACTIVATE_LOGICAL_LINKS** verb.

Note: The host link name is ignored if **nau_address** equals 0 is specified. Otherwise, this parameter is optional. If a NAU address is specified, the LU definition is assigned to the CP-PU.
special characters $, #, and @.

This is a Type A EBCDIC character string. If the name is less than 8 bytes, it must be padded on the right with EBCDIC blanks. In addition, the string cannot begin with an EBCDIC "@" (X'7C').

LU_model_name The 7-byte EBCDIC model name of the dependent LU 6.2 local LU. This is a 7-byte EBCDIC character string consisting of uppercase A-Z and 0-9 only. If the **lu_model_name** is not used, it must be filled with EBCDIC blanks. It should match a model name on a Host Model definition.

ip_address_mask This is the mask to be used in dynamic LU name generation. It is ignored if lu_name is not a template. The mask is encoded as a big-endian long; i.e., high-order byte first to low-order byte last.

Returned Parameters

primary_rc
secondary_rc See TCP62 Return Codes for details on **primary_rc** and **secondary_rc** verbs.

lu_name The real name of the defined LU. If the supplied lu_name was a template, it is replaced with the generated name.

lu_alias If all blank or nulls on input, this will contain the LU name (in ASCII) on output.

DEFINE_PARTNER_LU_TN62

TCP62 extends the **DEFINE_PARTNER_LU** verb to allow the definition of LU names that are generated dynamically, based on a supplied template, mask, and IP address. The verb is extended by using a new overlay structure. The DEFINE_PARTNER_LU_TN62_OVERLAY is specified after all the alt_alias_overlay structures (if any) in the DEFINE_PARTNER_LU VCB.

The opcode field in the DEFINE_PARTNER_LU VCB must be set to AP_DEFINE_PARTNER-LU_TN62 before invoking the TCP62 API. For a description of a **DEFINE_PARTNER_LU** verb, see *Communications Server for OS/2 Warp: System Management Programming Reference*.

OVERLAY Structure

```
typedef struct define_partner_lu_tn62_overlay
{
     unsigned long    ip_address_mask;   /* mask used in dynamic */
                                         /* name generation      */
     unsigned long    partner_ip_addr;   /* IP address of the    */
                                         /* partner LU           */
     unsigned char    partner_hostname[220];
                                         /* host name of the     */
                                         /* partner LU           */
} DEFINE_PARTNER_LU_TN62_OVERLAY;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_PARTNER_LU_TN62.

fq_partner_lu_name

This is either a real fully-qualified partner LU name or a template for a fully-qualified partner LU name. If there are no template replacement characters (‘*’), it is a real name, otherwise it is a template. The net ID must not contain any template replacement characters. Note that the replacement character is ‘*’ instead of ‘.’ to avoid confusion with the ‘.’ separating the net ID and partner LU name.

ip_address_mask This is the mask to be used in dynamic partner LU name generation. It is ignored if **fqplu_name** is not a template. The mask is encoded as a big-endian long, that is, high-order byte first to low-order byte last.

If the **fqplu_name** is a template, one of the following two parameters must be specified. If both are specified, **partner_ip_addr** takes precedence.

partner_ip_addr This is the IP address of the partner LU. It is ignored if **fqplu_name** is not a template. All zeros indicate "unspecified". The IP address is encoded as a big-endian long; that is, high-order byte first to low-order byte last.

partner_hostname This is the host name of the partner LU. It is ignored if the **fqplu_name** is not a template. All blanks indicate "unspecified".

Returned Parameters

primary_rc
secondary_rc
fq_partner_lu_name

See **TCP62 Return Codes** for details on **primary_rc** and **secondary_rc** verbs.

The real fully-qualified LU name. If the supplied **fq_partner_lu_name** was a template, it is replaced with the generated name.

TCP62 Return Codes

The following section summarizes the unique TCP62 return codes. Note, DEFINE_LOCAL_LU_TN62 and DEFINE_PARTNER_LU_TN62 may also have return codes that are described in *Communications Server for OS/2 Warp: System Management Programming Reference*. Each subsection heading lists both the primary and secondary return codes in parenthesis (primary_rc, secondary_rc), using defined symbols located in **tn62api.h** or **appcdef.h**.

(TN62_ERROR, TN62_NODE_RUNNING_NO_ANYNET)

returned by	START_TN62
cause	The Communications Server node is running and the running configuration does not support AnyNet. START_TN62 did not complete successfully.
corrective action	Stop the node and reissue the START_TN62.

(TN62_ERROR, TN62_CONFIGURATION_FILE_ERROR)

returned by	START_TN62
cause	A TCP62 call to the Communications Server configuration API failed. The most likely cause is a TCP62 or Communications Server program defect. START_TN62 did not complete successfully.
corrective action	None. Collect problem determination data by turning on all tracing and recreating the problem. Also, capture any log data.

(TN62_ERROR, TN62_NODE_NOT_STARTED)

returned by	START_TN62
cause	The node failed to start. The most likely cause is a TCP62 program defect. START_TN62 did not complete successfully.
corrective action	None. Collect problem determination data by turning on all tracing and recreating the problem. Also, capture any log data.

(TN62_ERROR, TN62_NODE_START_INCOMPLETE)

returned by	START_TN62
cause	The node started, but the configuration was not successful. The most likely cause is the AnyNet program is not installed. The node is running.
corrective action	Ensure the AnyNet component is installed. If the problem persists, collect problem determination data by turning on all tracing and recreating the problem. Also, capture any log data.

(AP_OK, TN62_PARAMETERS_NOT_USED)

returned by	START_TN62
cause	The AnyNet parameters (timers and **domain_name_suffix**) or both the CP name and CP alias from START_TN62 were not used. The node was already running using different parameters. The parameter values used by the running node are returned in the START_TN62 VCB. START_TN62 completes successfully.
corrective action	None.

(TN62_ERROR, TN62_NAME_GENERATION_ERROR)

returned by	START_TN62, DEFINE_LOCAL_LU_TN62, DEFINE_PARTNER_LU_TN62
cause	Dynamic name generation failed due to gethostname (for START_TN62 and DEFINE_LOCAL_LU_TN62) or gethostbyname (for DEFINE_PARTNER_LU_TN62). The most likely causes are TCP/IP was not installed, configured and active, or an incorrect **partner_hostname** passed on DEFINE_PARTNER_LU_TN62. The verb does not complete successfully.
corrective action	Ensure that TCP/IP is configured and active, and that the **partner_hostname** on DEFINE_PARTNER_LU_TN62 is correct.

(AP_PARAMETER_CHECK, INVALID_CP_NAME)

returned by	START_TN62
cause	The **fqcp_name** is not valid. The net ID must not contain any template replacement characters and the CP name must not begin with a replacement character. Except for the replacement character, this must be a valid EBCDIC fully-qualified CP name. START_TN62 does not complete successfully.
corrective action	Correct the **fqcp_name** parameter.

(AP_PARAMETER_CHECK, INVALID_LU_NAME)

returned by	DEFINE_LOCAL_LU_TN62
cause	The **lu_name** is not valid. The LU name must not begin with a replacement character. Except for the replacement character, this must be a valid EBCDIC LU name. DEFINE_LOCAL_LU_TN62 does not complete successfully.
corrective action	Correct the **lu_name** parameter.

(AP_PARAMETER_CHECK, INVALID_FQ_LU_NAME)

returned by	DEFINE_PARTNER_LU_TN62
cause	The **fqplu_name** is not valid. The net ID must not contain any template replacement characters and the partner LU name must not begin with a replacement character. Except for the replacement character, this must be a valid EBCDIC fully-qualified LU name. This return code will also occur if the **fqplu_name** is a valid template, but both **partner_ip_addr** and **partner_hostname** are unspecified. DEFINE_PARTNER_LU_TN62 fails.
corrective action	Correct the **fqplu_name** parameter or the (**partner_ip_addr**, **partner_hostname**) combination.