Guide to OpenVMS Alpha PPP Privileged Interfaces

November 7, 1996

This document describes the privileged interfaces for the Point-to-Point Protocol (PPP) support provided in OpenVMS Alpha Version 7.1.

Revision/Update Information: This a new document.

Software Version: OpenVMS Alpha Version 7.1

Digital Equipment Corporation
Maynard, Massachusetts

November 1996

© Digital Equipment Corporation 1996. All rights reserved.

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

Preface

This document describes the privileged interfaces for the Point-to-Point Protocol support provided in OpenVMS Alpha Version 7.1. It assumes that the reader has a knowledge of the OpenVMS VCI programming interface and is familiar with OpenVMS device drivers.

Overview of OpenVMS Alpha PPP Support

The Point-to-Point Protocol (PPP) is an internet layer protocol which is now the widely accepted replacement for the Serial Line Interconnect Protocol (SLIP). PPP support enhances the connectivity and internet-readiness of OpenVMS Alpha by providing a way to establish a dynamic Internet Protocol (IP) network connection over a serial line–without extensive router or server hardware.

Based on code originated at Carnegie Mellon University, this serial protocol features:

- A way to encapsulate datagrams an a serial link. Specifically, PPP supports an asynchronous link with 8 bits of data and no parity.
- A link control protocol (LCP) that establishes and verifies the data link connection. This protocol enables each host, network, or transport to configure various communication options.

To support this protocol, OpenVMS Alpha Version 7.1 offers a Point-to-Point Protocol utility (PPPD) and related device driver.

The Point-to-Point Protocol utility (PPPD) is a DCL-level tool that you can use to initiate and manage an IP network connection over an asynchronous, serial data line. PPPD extends the networking capability of OpenVMS Alpha by enabling you to:

- Establish temporary, high-speed network connections between remote hosts. This includes both *dial-in* capability from a remote host to an OpenVMS Alpha host and *dial-out* capability from an OpenVMS Alpha host to a remote system or server box that supports PPP.
- Establish permanent, low-speed network connections between local hosts, such as between a laptop computer and an Alpha workstation connected by a serial data line.
- Set and display communication characteristics, such as address compression, flow control, and line speed.

Intended Audience

This information in this guide is intended for system and application programmers. It presumes that its readers are familiar with the OpenVMS Alpha programming environment and concepts.

Related Documents

Refer to the *TCP/IP Networking on OpenVMS Systems* for a detailed description of this utility and its related subcommands. Sample PPPD connection procedures are also provided online in an ASCII (.TXT) and PostScript (.PS) file. Once you install the OpenVMS Alpha Version 7.1 kit, you can retrieve either file from the following location:

```
SYS$SYSROOT:[SYSHLP.EXAMPLES.PPPD.DOC]PPP_CONNECT.TXT (or .PS)
```

For additional information on the Open Systems Software Group (OSSG) products and services, access the Digital OpenVMS World Wide Web site. Use the following URL:

```
http://www.openvms.digital.com
```

Reader's Comments

Digital welcomes your comments on this manual.

Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT and send us your comments by:

Internet	**openvmsdoc@zko.mts.dec.com**
Fax	603 881-0120, Attention: OSSG Documentation, ZKO3-4/U08
Mail	OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Conventions

The name of the OpenVMS AXP operating system has been changed to OpenVMS Alpha operating system. Any references to OpenVMS AXP or AXP are synonym ous with OpenVMS Alpha or Alpha.

1
Protocols Interface

This chapter describes how to write a network control protocol and how to use the VCI interface to transfer packets. It is this interface that the network developers need to read and understand in order to use PPP.

Protocols gain access to the PPP driver via a VCI interface. The VCI (VMS Communications Interface) is an OpenVMS internal interface which may be used between any system level Kernel mode modules, called VCMs (VMS Communication Modules). Messages are passed between VCMs via the VCRP (VMS Communication Request Packet) structure. Although the VCI interface permits VCRPs to be chained, to simplify the design, the PPP driver supports unchained requests only

A structure called the VCIB (VMS Communication Interface Block) maintains the context of the interface and is shared between both the upper and lower VCMs. The services which comprise the VCI for the PPP driver are described in the following sections. For more information on VCI refer to the VCI Specification. All routines must be called at IPL8 with IOLOCK8 as the highest lock acquired. Calls into the upper VCM will also adhere to these rules.

The PPP VCM provides the following VCI services (see subsequent sections for explanations).

- CREATE_PORT
- DELETE_PORT
- PORTMGMT_INITIATE
  - ENABLE_PORT
  - DISABLE_PORT
- TRANSMIT_INITIATE

The upper VCM is expected to provide the following services

- PORTMGMT_COMPLETE
- TRANSMIT_COMPLETE
- RECEIVE_COMPLETE
- REPORT_EVENT

The descriptions of the service routines in the following sections use C programming notation.

The interface to the Report Event routine is as follows:

The Calling Sequence is:

```
@VCIB$A_REPORT_EVENT(R4)                    JSB
INPUTS:
                        R1 = Event Code
                        R2 = Reason Code (optional)
                        R4 = Address of VCIB
OUTPUTS:
                        R0 and R1 may be destroyed
```

1.1 VCIB Extension Fields

The VCIB used to connect the upper VCM to the PPP VCM has three additional fields added to the standard VCIB. Table 1–1 describes the extra fields in the VCIB.

Table 1–1 VCIB Extensions

Field Name	Client	PPP Access	Size Access	Description
PORT_ID	No Access	IRWZ	Long Word	Private to PPP
FCS_SIZE	R	IRWZ	Byte	Space to leave for FCS
HDR_SIZE	R	IRWZ	Byte	Space to leave for PPP header
MTU	R	IRWZ	Word	Maximum

Key to Access:

I: The module initializes this field when the VCI port is ENABLED.
R: The module has read access to the field.
W: The module has write access to the field.
Z: The module will zero the field when the port is disabled

A detailed description of the VCIB extension field follows:

- VCIB$L_PPP_PORT_ID

  This field is intended for use entirely by the PPP VCM. It provides a one-to-one mapping from the VCI Port to the PPP Port. The field is initialized when the client enables the VCI Port.

- VCIB$B_PPP_FCS_SIZE

  This field is the amount of space in bytes to leave at the end of a transmit request so PPP can append it's FCS.

- VCIB$B_PPP_HDR_SIZE

  This field is the amount of space in bytes to leave at the front of a transmit request so PPP can backfill its headers.

- VCIB$W_PPP_MTU

  This is the maximum number of bytes of data that can be transmitted in a single request.

1.2 Registration of the PPP VCM

The PPP VCI is made available to the upper VCMs by registering its CREATE_PORT and DELETE_PORT services with the VCI registry. The PPP VCM is identified by the constant VCI$K_ID_PP which is defined in the VCIBDEF header module. Registration takes place when the VMS executive image is loaded. The interface to the registry is provided by a jacket routine and is described below.

```
Procedure   u_int VCI$PPP_REGISTRY(u_int Request, u_int
              VCM_ID,
                                           u_int
              (*CreatePort)(), u_int (*DeletePort)() )

  Description Register the PPP VCM with the VCI registry to
              make it available to upper VCMs

  Returns     SS$_NORMAL          Successful completion
              SS$_OPINCOMPL       User is not a registered
              VCM
              SS$_INSFEM          Insufficient memory to
                                  register VCM.

  Parameters   Request             VCIB$K_FC_REGISTER
               VCM_ID              VCI$K_ID_PPP
               CreatePort          Address of Create Port service
                                   entry point
               DeletePort          Address of
                                   Delete Port service entry point
```

1.3 VCI Port Management Services

CREATE_PORT Service

```
Procedure      u_int VCI$PPP_CREATE_PORT(VCIB *Port)

Description    The VCI$PPP_CREATE_PORT entry point is used
               by the VCI registry when an upper VCM wants
               to create a port to this VCM.  It is called
               with one parameter, the VCIB which the
               upper VCM has already initialised its
               fields and entry points. The PPP driver
               will initialise PPP's VCI entry points and
               the relevant fields in the VCIB.

Returns       SS$_NORMAL          Port successfully created

Parameters     Port      A pointer to the VCIB.
```

DELETE_PORT Service

Procedure u_int VCI$PPP_DELETE_PORT(VCIB *Port)

Description The VCI$PPP_DELETE_PORT entry point is used by the VCI registry when a upper VCM wants to delete a VCI port to this VCM. It is called with one parameter which is the VCIB that describes the connection to the lower VCM. This routine performs all decommissioning for this VCI port and zero's the fields earlier initialised in the CREATE_PORT(). The upper VCM is responsible for deallocating the VCIB when this call completes.

NOTE: Although this call can be issued in any port state, it is recommended the port be disabled before deleting it.

Returns	SS$_NORMAL	Port was successfully deleted
	SS$_DEVACTIVE	The port has not been disabled
Parameters	Port	A pointer to the VCIB

PORTMGMT_INITIATE Service

Procedure u_int VCI$PPP_PORTMGMT_INITIATE(VCIB *Port, VCRP *Request)

Description This entry point is called to perform functions on the VCI port. The table below lists the valid function codes and the action they perform. The action taken by the routine depends on the value in the VCRP$L_FUNCTION field. The function value also determines: the format of the item lists passed in (if any) and the meaning of return values.
The call to the PORTMGMT_INITIATE service will always return a success. The service completes it requests asynchronously by calling the upper VCM's PORTMGMT_COMPLETE routine. The status of the call is determined by the VCRP$Q_REQUEST_STATUS field in the VCRP returned by PORTMGMT_COMPLETE.

Returns	SS$_NORMAL	Successful completion
Parameters	Port	A pointer to the VCIB
	Request	A pointer to the VCRP containing the portmgmt request

****

Table: Supported PORTMGMT_INITIATE functions.

Function Code	Description
ENABLE_PORT	Initiate actions to allow data transmission and reception of protocol packets

```
DISABLE_PORT   Set the port to a state where it can no
               longer transmit/receive protocol packets

******************************************************************
```

VCRP$K_DISABLE_PORT

```
Description       The VCRP$K_FC_PORT_DISABLE_PORT is
                  used by the upper VCM to request a
                  protocol no longer be run over a
                  communications port.
                  Unlike the LAN VCI, no PORT_UNUSABLE
                  event will result from calling this
                  service.

Data Structures   None

Returns           SS$_NORMAL         Successful
                  completion
```

VCRP$K_ENABLE_PORT

```
Description   The VCRP$K_FC_PORT_ENABLE_PORT is used by the
               upper VCM  to request  a protocol  to  be run
               over a communications port.   Notification of
               successful completion  of  this  function via
               the PORTMGMT_COMPLETE  service does  not mean
               the upper VCM is able  to transmit or receive
               data; only that PPP has  committed to put the
               port  into  an  appropriate  state  for  data
               transfer.   If   the  protocol   requires  a
               matching control  protocol  and  one has  not
               been registered the  function will  fail.
               If the PORTMGMT_COMPLETE indicates success, the protocol
               must wait for a PORT_USABLE event  before data transmission
               is permitted.

             Data Structures  The input itemlist is used to specify the
                              communications port and the protocol to run.
                              The following table lists the supported input
                              item list fields.

              Returns    SS$_NORMAL         Successful completion
******************************************************************************
              Table:  ENABLE_PORT input itemlist
```

Item Code	Type	Mandatory	Description
PPP$K_NAME	ASCII	Yes	Name of PPP line to start protocol on.
PPP$K_PROTOCOL	WORD	Yes	Number of the protocol to start.

```
******************************************************************************
```

1.4 VCI Transmit Service

```
Procedure   u_int VCI$PPP_TRANSMIT_INITIATE(VCIB *Port VCRP
            *Request)
```

```
Description   This routine is called by the upper VCM when it
              has data to send.  The upper VCM must leave
              HDR_SIZE bytes at the start of the VCRP to
              allow the PPP protocol header to be backfilled.
              Must also leave FCS_SIZE at the back of the
              VCRP to allow an FCS to be appended to the
              frame. The FCS_SIZE and HDR_SIZE fields  live
              in the VCIB and are filled in when a
              PORT_USABLE event is received.  Completion of
              the VCRP is done via the TRANSMIT_COMPLETE
              service.
          NOTE: Only unchained transmit requests are
          supported.

Returns   SS$_NORMAL          Successful completion

Parameters   Port Request     A pointer to the VCIB
                              containing the request.
```

1.4.1 Upper VCM Services

This section describes the entry points that must exist in a PPP VCI Client.

PORTMGMT_COMPLETE

```
Procedure       void VCI$xxx_PORTMGMT_COMPLETE(VCIB
                *Port, VCRP *Request)

Description     This entry point is called to signal to
                the upper VCM the completion a
                PortMgmt_Initiate request.  The upper VCM
                determines the status of the
                PortMgmt_Initiate request by examining
                the VCRP$Q_STATUS field in the returned
                VCRP.

 Returns        None

Status          ALL FUNCTION CODES:

                SS$_NORMAL        Normal successful completion
                SS$_ILLIOFUNC     The function code field of the
                                  VCRP had a value that did not
                                  represent a function known
                                  by the PPP driver.

                ENABLE_PORT:

                SS$_DEVACTIVE    The VCI port is already enabled
                SS$_DEVOFFLINE   No Control Protocol registered
                SS$_DEVALLOC     The protocol is already in use on this line by
                SS$_NOSUCHDEV    No such line.
                SS$_INSFMEM      Insufficient memory.
                SS$_INSFARG      A protocol and/or line name were
                                 not supplied in the input
                                 itemlist.
                SS$_BADPARAM     A specified item
                                 code is not valid. The code of
                                 the offending item is
                                 returned in the
                                 VCRP$L_REQUEST_STATUS_QUAL field.
                SS$_VALNOTVALID  The value specified for
                                  an item code is not valid.
                                  The code of the offending item
                                  is returned in the
                                 VCRP$L_REQUEST_STATUS_QUAL field.
```

DISABLE_PORT:

SS$_DEVINACT	The VCI port is already disabled
SS$_OPINCOMPL	A previous disable request hasn't been completed

Parameters

Port	A pointer to the VCIB
Request	A pointer to the VCRP containing the original PORTMGMT request

TRANSMIT_COMPLETE

Procedure

```
void VCI$xxx_TRANSMIT_COMPLETE(VCIB *Port,
VCRP *Request)
```

Description

This routine is called when a packet has been successfully queued to the communications port (e.g. ASN2) for transmission or data transmission has failed for some reason (e.g. the communications port is no longer in a state to send data).

The upper VCM determines the status of the TRANSMIT_INITIATE request by examining the VCRP$Q_STATUS field in the returned VCRP.

Returns None

Status

SS$_NORMAL	Packet successfully sent
SS$_DEVINACT	Port is not in a state permitting data transfer.
SS$_IVBUFLEN	The buffer length exceeds MTU

Parameters

Port	A pointer to the VCIB
Request	A pointer to the VCRP containing the original transmit request

RECEIVE_COMPLETE

Procedure

```
void VCI$xxx_RECEIVE_COMPLETE(VCIB *Port,
VCRP *Request)
```

Description

When PPP receives a packet for a usable port of an upper VCM, it calls this service in the upper VCM to pass up the packet. The upper VCM is responsible for deallocating the VCRP when it has finished processing the packet. The connection_id field of the VCRP will contain the protocol type of the received packet

NOTE: When the IP protocol is enabled on PPP port, the upper VCM will receive both IP and VJ compressed packets if compression has been negotiated by the control protocol The upper VCM uses the protocol type field to distinguish between IP and VJ compressed packets..

Returns None

Port	A pointer to the VCIB Parameters
Request	A pointer to the VCRP containing the received packet

REPORT_EVENT

Procedure void VCI$xxx_REPORT_EVENT(VCIB *Port, u_int Event, u_int Reason)

Description The PPP driver reports events by calling this service in the upper VCM. The VCRP$L_FUNCTION field of the VCRP indicates the type of event that occurred. Table 1 lists the events that may be reported. Table 2 lists the reasons that can be associated with each event.

Returns None

Parameters

Port	A pointer to the VCIB
Event	The event that occurred
Reason	The action that generated the event.

****

Table 1 Events Reported to PPP VCI Client

Event	Description
PPP$K_PORT_USABLE	The port is now in a state where it can be used for data transfer. No Reason is supplied with this event.
PPP$K_PORT_UNUSABLE	The port can no longer be used for data transfer. A protocol can elect to disable and then re-enable the port to make another attempt to bring up the connection. Table 2 lists the reasons that can be generated with this event.
PPP$K_PORT_FAILED	The port can no longer be used. The only action a protocol can take is to delete the port.

****

****

Table 2 Reasons Associated with Events

Reason	Description
PPP$K_LCP_FAILED	The Link Control Protocol has been terminated
PPP$K_NCP_FAILED	The Control Protocol has been terminated
PPP$KDEV_FAILED	The communications port being used by the PPP line has failed
PPP$K_LINE_DELETED	Network management has deleted the PPP line
PPP$K_NO_REASON	No reason associated with this event.

****

2

Network Control Protocols Interface

This chapter describes the interface provided for implementing network control protocols.

The implementer of the control protocol is responsible for supplying and checking the configuration options for each communications port the control protocol is run over. The PPP driver maintains the finite state machine associated with the control protocol running on each communications port. A vector of routines for calling into the PPP finite state machine is provided. This method was chosen because it requires the least amount of changes to the pppd code. The interface routines use the standard calling sequence and are callable by kernel modules only. A function which returns the start of the vector is provided for modules to link against. The vector provides the following routines:

- fsm_register — Notifies PPP that the module is responsible for a control protocol.
- fsm_deregister —- Notifies PPP that the module is no longer responsible for a control protocol.
- fsm_start — Starts a control protocol on a connection.
- fsm_stop - Stops a control protocol on a connection.

The routines must be called in kernel mode with IOLOCK8 held. The callback routines are executed in kernel mode with IOLOCK8 held, therefore code that implements a network control protocol must be loaded into nonpaged pool in the form of an EXEC loadable image or device driver.

A module registers itself as willing to take care of a control protocol by calling the fsm_register routine. The module is then responsible for handling all connections using this control protocol across all communications ports using PPP. That is, there can only be one module registered as being responsible for the control protocol on the system at any time. The fsm_register routine takes as an input parameter a structure that contains pointers to routines which the PPP driver calls to notify the module of state transitions and requests to accept/reject configuration options. If a control protocol does not want to be notified of a particular event, it does not supply a callback routine for it. If no callback routine is supplied for an event a default action is taken.

A control protocol module is notified of requests to start the control protocol. The module itself does not initiate requests to start a control protocol. The PPP driver calls the control protocol's open callback routine. The call contains a which identifies this connection to the state id machine, and the VCIB of the associated protocol port. The id is used for calling into the state machine. The module calls the fsm_open routine to accept the connection and kickstart the PPP state machine. The module supplies a which is the modules identifier for the connection unit (.e.g. this could be the index of this connection into an array of connections). Once a module has started the state machine it will be notified of requests to accept/reject configuration options if it has elected to be notified of these events. The unit is used by callbacks to the module, to identify the particular connection for the request.

If the control protocol provider does not want to start the control protocol on this connection it calls the fsm_close routine. The fsm_close routine can also be called at any time to stop a control protocol.

If the state machine closes a connection the module will be notified via a call to its close( ) routine.

If a module no longer wants to be responsible for a control protocol it calls the fsm_deregister routine. Deregistering a control protocol closes any connections running this control protocol.

2.1 Calls into the PPP driver by the Control Protocol

This section describes the calls a module implementing a control protocol can make into the state machine.

fsm_register

```
             Procedure   u_int fsm_register( u_int ctrlProt, protentry
                         *callbacks , u_int *error )

             Description Notifies PPP that the module will process
                         requests to start the given control protocol.
                         The module is then responsible for handling
                         all connections using this control protocol
                         across all communications ports using PPP.

             Returns     TRUE    protocol successfully registered
                         FALSE   protocol could not be registered.
                                 The error contains the reason
                                 for the failure. The table below lists the
                                 errors that can be returned on
                                 failure.

            Parameters   ctrlProt        control protocol number
                         callbacks       structure containing pointers
                                         to callback routines for the control
                                         protocol
*****************************************************************************
     Table:  Errors Returned by the fsm_register Routine

            Error           Description

          SS$_VALNOTVALID   The ctrlProtocol parameter does not specify
                            a valid control protocol. That is, it is not in
                            the range 0x8000 to 0x3FFFF

          SS$_DEVALLOC      The  control  protocol   has  already  been
                            registered.

          SS$_INSARG        The values  for the  open and/or the close
                            callbacks have not been supplied.
*****************************************************************************
```

fsm_deregister

```
Procedure   void fsm_deregister( u_int ctrlProt )

Description Notify PPP that the module is no longer
            responsible for a control protocol

Returns     none

Parameters   ctrlProt       control protocol number
```

fsm_start

```
Procedure   void fsm_start( fsmId id, int unit )

Description Accept the request to start a control
            protocol on a connection.

Returns     none

Parameters  id    state machines identifier for the connection
            unit  module's identifier for the connection
```

fsm_stop

```
Procedure     void fsm_stop( fsmId id )

Description   Stop a control protocol running on a connection

Returns        none

Parameters   id state  machines  identifier  for  the connection
```

fsm_send

```
Procedure   void fsm_send( fsmId id, u_char code, u_char
            newPktId, u_char *data, int dataLen )

Description Get the state machine to send a control packet of
            type code

NOTE:  This routine is only supplied for control
       protocols which need to send a control packet.
       This would be in response to a control packet
       being received which the state machine doesn't
       process. That is, control protocols which need to
       supply the extcode() callback.  In most cases a
       control protocol will not need to use this routine.

Returns     none

Parameters
            id       state machines identifier for the connection

            code     type of control packet to sent

            newPktId if TRUE the state machine will generate a
                     new packet identifier for this
                     control packet; otherwise it will use the
                     identifier of the last received packet.

            data    This is the control packet (minus the
                    code, identifier and length fields.
                    These will be added by the state machine).

            dataLen Length in bytes of the data
```

2.2 Callbacks to the Control Protocol by the PPP Driver

The following callbacks can be provided by the module when registering a control protocol. The PPP driver calls these routines in response to different control packets being received. The PPP driver only handles control packets with codes 1-7 (That is, Configure-Request, Configure-Ack, Configure-Nak, Configure-Reject, Terminate-Request, Terminate-Ack, and Code-Reject). All other control packets will be handed to the control protocol via the supplied extcode callback.

The open and close routines must be specified. The cilen, addci, ackci, nakci, and rejci routines determine the host's negotiation of configuration options. Either none or all of these routines must be supplied. All the other routines are optional. The default action is taken if the routine is not supplied. All the routines are called with a unit parameter to identify the connection this call is for.

The PPP driver verifies that the identifier field of a control packet is valid. For callbacks which accept or reject configuration options (e.g. ackci) the control protocol module is responsible for:

- Checking the options have not been reordered
- Updating its copy of the options

There is no need for the control protocol module to ask that request with the updated options be sent. The PPP driver will callback to the module to obtain the updated options when it attempts to send a Configure-Nak or Configure-Reject.

The supported callback routines and the default action taken if they are not supplied are described in the following subsections.

open

```
Procedure   void open( fsmId id, PPPvcib *vcib )

Description   This function notifies the client of a
              request to start a control protocol on a
              communications port.  The control protocol
              uses the fsmId parameter for calls back into
              the state machine.
              When a control protocol receives an open
              request in MUST respond with either an
              fsm_open call to accept the connection or an
              fsm_close call to reject the connection.
              Failure to do so will leave the state machine
              in an undefined state.

Returns     none

Parameters
            id      Identifier for fsm for this control protocol.
                     Used for calls into the state machine

             vcib    Pointer to PPP vcib (i.e. port) associated
                     with this control protocol
```

close

Procedure `void close( int unit )`

Description This function notifies a client that the control protocol is no longer needed on this communications port (there is no corresponding protocol using the communications port) or the protocol can no longer be run on this communications port (for example, the modem has hung up). No more calls to the id matching this unit are permitted.

Returns none

Parameters unit Client’s reference for this connection

resetci

Procedure `void resetci( int unit )`

Description This routine is called to tell a client to set its configuration options to their initial values. No action results if this routine is not supplied.

Returns none

Parameters unit Client’s reference for this connection

cilen

Procedure `int cilen( int unit )`

Description This routine is called to obtain the length of client’s of the current configuration options. This is the length of the options as they would appear in the a PPP packet. The state machine calls this routine when building Configure-Requests Not supplying this routine implies the control protocol won’t negotiate any options and results in a Configure-Request with no options being sent.

Returns Length of client’s configuration information

Parameters unit Client’s reference for this connection

addci

Procedure `void addci( int unit, u_char *options, int *len )`

Description The state machine calls this routine when building Configure-Requests. The routine adds each of desired configuration options to the buffer. The options should be formatted in accordance with the control protocol being implemented. i.e. as they would appear in a Configure Request.

Not supplying this routine implies the control protocol won’t negotiate any options and results in a Configure-Request with no options being sent.

Returns none

Parameters

unit Client’s reference for this connection

options Pointer to buffer to copy configuration options to

len On input - length of buffer
On output - length of configuration options in buffer

ackci

Procedure `int ackci( int unit, u_char *options, int len )`

Description The state machine calls this routine when it has received a Configure-Ack for the control protocol, i.e. the remote end has accepted the clients configuration options.
The routine parses each option in the options buffer and checks that it is formatted correctly and that each option appears in the same order as it was specified in the addci routine. If all the options meet these requirements then the routine returns 1 (to indicate a valid Ack); otherwise 0 is returned.

Not supplying this routine implies the control protocol won’t negotiate any options and any Configure-Ack received containing options is ignored by the state machine.

Returns 1 if Ack is valid
0 invalid ack

Parameters

unit Client’s reference for this connection

options Pointer to buffer containing configuration options

len Length of configuration options

nakci

Procedure `int nackci( int unit, u_char *buffer, int len )`

Description The state machine calls this routine when it has received a Configure-Nak for the control protocol. i.e. the remote end doesn’t like some of the values in the requested configuration options.
This routine is responsible for checking that this is a valid Nak. The Nak is valid: if all the options appearing in the buffer are formatted according to the control protocol.

If an option was specified by the addci routine, then option appears in the same order as specified by the addci routine.

If an option was not specified by the addci routine it appears after any of the options that were specified by the addc routine.

If the Nack is invalid the routine returns 0. If the Nack is valid the routine returns 1, for any options in the buffer that were specified by the addci routine, the nakci routine updates the client's value of the options (preferably to the value the option has in the buffer); for any options appearing in the buffer that weren't specified by the addci routine, the nackci routine updates the client's options so that it will be specified the next time the addci routine is called.

Not supplying this routine implies that the control protocol will not negotiate any options and any Configure-Nak received containing options is ignored by the state machine

Returns
- 1 if nack is valid
- 0 invalid nack

Parameters
- unit Client's reference for this connection
- buffer Pointer to buffer containing configuration options
- len Length of configuration options

rejci

Procedure `int rejci( int unit, u_char *buffer, int len)`

Description The state machine calls this routine when it has received a Configure-Rej for the control protocol. That is, the remote has returned the options it is NOT willing to negotiate. The routine parses each option in the options buffer and checks that it is formatted correctly and that each option in the buffer appears in the same order as it was specified in the routine. If all the options meet these requirements then the routine returns 1 (to indicate a valid)and the routine updates the copy of the client's options so that none of the options appearing in the buffer are negotiated (i.e. will not be inserted by the i routine). If one or more options are formatted incorrectly or are out of order 0 is returned (to indicate an invalid Rej)

Not supplying this routine implies the control protocol won't negotiate any options and any Configure-Rej received containing options is ignored by the state machine

Returns
- 1 if reject is valid
- 0 invalid reject

Parameters

unit	Client’s reference for this connection
buffer	Pointer to buffer containing configuration options
len	Length of configuration options

reqci

Procedure

```
int reqci( int unit, u_char *options, int
*len, int reject_if_disagree )
```

Description

The state machine calls this routine when it has received a Configure-Request for the control protocol from the peer.
The routine parses each option in the buffer and decides whether to accept/reject the option.. If all the options in the buffer are formatted correctly and acceptable to the client, then the routine returns CONFACK.
If the client finds one or more options it is not willing to negotiate on then it returns CONFREJ and updates the options buffer to contain the options it is not willing to negotiate. If the client finds an incorrectly formatted option, the routine returns CONREJ and options parameter is updated to point to all options from the bad option onwards i.e. the remainder of the original options buffer.
If the client is prepared to negotiate on all the options but finds one or more values specified in the options unacceptable, the routine returns CONFNAK and updates the option in buffer to contain an acceptable value.
If the processing results in a NAK and the reject_if_disagree flag is set, then the routine must return CONFREJ.

Not supplying this routine is equivalent to not allowing any options to be negotiated by the peer, that is, if the peer’s request contains any options a Configure-Rej will be sent

Returns

CONFACK	All the options are acceptable to the client
CONFNAK	One or more options is not acceptable
CONFREJ	The client is not willing to negotiate on one or more options

Parameters

unit	Client’s reference for this connection options On Input: Pointer to buffer containing configuration options On Output: Pointer to buffer with options to Ack/Reject/Nak
len	On Input: Length of configuration options On Output: Length of options to Ack/Rej/Nak

reject_if_disagree	The Client must return REJ instead of a NAK when this flag is set to TRUE.

up

Procedure void up(int unit)

Description This routine is called to notify a client that the control protocol has entered the OPENED state i.e. the corresponding protocol can now be used on the connection. The routine should take any action it deems necessary. No action results if this routine is not supplied

NOTE : The protocol using the VCI interface will be notified separately that it can be now be used.

Returns none

Parameters unit Client's reference for this connection

down

Procedure void down(int unit)

Description This routine is called to notify a client that the control protocol has left the OPENED state i.e. the corresponding protocol can no longer be used on the connection. The routine should take any action it deems necessary. No action results if this routine is not supplied. NOTE: The protocol using the VCI interface will be notified separately that it can no longer be used.

Returns none

Parameters unit Client's reference for this connection

extcode

Procedure int extcode(int unit, int code, int id, u_char *pkt, int len)

Description The state machine calls this routine when it has received a packet for the control protocol that the state machine doesn't handle (i.e. the packet isn't a Configure-Request, Configure-Ack, Configure-Nak, Configure-Reject, Terminate-Request, Terminate-Ack or Code-Reject). It is up to the routine to decided what to do with the packet. If the routine elects to handle the packet it returns 1; otherwise it returns 0. It the client doesn't handle the packet, the state machine will send a Code-Reject.

If this return is supplied a Code-Reject will be sent.

Returns

1	packet recognised and handled by client
0	packet not handled by client

Parameters	unit	Client's reference for this connection
	code	type of control packet
	id	identifier field of control packet
	pkt	pointer to control packet data
	len	length of control packet data

3 Programming Notes

This chapter describes how to use the programming interfaces to implement protocols and control protocols for PPP.

3.1 Control Protocols

3.1.1 Registering a Control Protocol

Call fsm_register( ), supplying the control protocol and pointers to the callback routines as input parameters. Pointers to the callback routines are provided by filling in the relevant fields in the entry parameter.

Once a control protocol has been registered the module will receive notification of requests to start a control protocol

The registering of a control protocol MUST take place before any requests to start the corresponding protocol occur. It is therefore recommended that the registering of a control protocol takes place when the module implementing is initialised. If a request to start a protocol occurs before a corresponding control protocol has been registered, the request will fail.

3.1.2 Deregistering a Control Protocol

Call the fsm_deregister0 routine.

The state machine will stop any connections using the control protocol. The module will receive notification that the connections have been stopped.

3.1.3 Receiving Notification of a New Connection

1. The module's open( ) routine is called. The id parameter identifies the connection to the state machine and must be used for calls into the state machine.
2. The module can accept this connection by call the fsm_start( ) routine. The connection is identified by the id parameter. The module supplies it's own reference to this connection (e.g. the index into an array of connections) by filling in the unitparameter. When the state machine calls back to the module the ref is used by the module to identify the connection.
3. If the module accepts the connection, the state machine will negotiate the configuration options with the peer and will call various callback routines as part of this process. The section Example operation of Negotiating Configuration Options below shows a typical exchange.

3.1.4 Example operation of Negotiating Configuration Options

A typical exchange (of a registered control protocol) is as follows:

- PPP driver calls the open( ) routine to signal to the module of a request to start the control protocol on a communications port. The opencall contains the name of the device and idto identify this connection to the state machine.
- The module accepts the request by calling fsm_open( ).
- PPP driver calls the cilen( ) and addci( ) routines to obtain the configuration options to request.
- The remote end accepts the configuration options. PPP driver calls the module's ackci routine.
- The remote end sends its configuration options. PPP driver calls the module's reqci( ) routine.
- The module finds all the options acceptable and indicates this in the return value of the reqci( ) call. (Isn't this amazing - neither peer disagreed with any of the options)
- The control protocol enters the OPENED state. PPP signals this to the module by calling its up( ) routine.

3.1.5 Receiving Notification that a Connection has Closed

The modules' close( ) routine will be called. No further action is required from the module.

3.1.6 Stopping a Connection

Call the fsm_close( ) routine supplying the id of the connection to be closed. The module does not receive notification that the connection was closed.

3.2 Protocols

3.2.1 Starting a Protocol

1. The Protocol VCM allocates a VCIB and initialises the required fields of the VCIB.
2. The Protocol VCM calls the CREATE_PORT routine (through the registry) to open a port to the PPP driver.
3. The Protocol VCM call the PORTMGMT_INITIATE requesting an ENABLE_ PORT function. The item list supplied as part of calling this routine will contain the protocol to start and the name of the PPP line to start it on. For this function to succeed a corresponding control protocol must have been registered if one is required.
4. The Protocol VCM receives notification of the status of the ENABLE_PORT function through a call to it's PORTMGMT_COMPLETE routine. If success is indicated the Protocol VCM waits for a PORT_USABLE event.
5. The Protocol VCM receives a PORT_USABLE event. It can now send data and will receive data from the peer.

3.2.2 Stopping a Protocol

1. The protocol VCM calls PORTMGMT_INITIATE routine specifying a DISABLE_PORT function.
2. The protocol VCM calls the DELETE_PORT routine (through the registry) to close the port to PPP. When this routine returns the Protocol VCM should delete the VCIB being used for the connection.

3.2.3 Events

```
Port_Usable Event
            This events signals that data reception and transmission can
            now take place.  When a Protocol VCM see this event it
            should be prepared to receive data from that time on.  The
            Protocol VCM can send data at any time after receiving this
            event.


Port_Unusable Event
            This events signals to the Protocol VCM that it MUST stop
            sending data and that no more data will be handed up to the
            Protocol VCM.  The event indicates a temporary condition
            only.  A PORT_USABLE event will be generated when the port
            is available for data transmission/reception again.

            A Protocol VCM can disable and re-enable the port to restart
            the protocol.

Port_Failed Event
            This event signals to the Protocol VCM that protocol can no
            longer be run because of a permanent condition.  The
            Protocol VCM should take the actions outlined in Stopping a
            Protocol on page 41.
```

3.3 Typical Sequence for Starting a Protocol

This section lists the sequence of events that occur when starting a protocol and corresponding control protocol on an asynchronous device. They are discussed in reference to a "protocol module which implements both the protocol and corresponding control protocol.

1. The protocol module registers the control protocol via a call to the fsm_register routine.
2. A user types the "CONNECT" command in the PPP utility to start PPP on the asynchronous device. The utility calls the protocol modules's registered network callback routine. The network callback routine contains the name of the asynchronous device.
3. The protocol module creates and initialises a PPP VCIB. The protocol module calls the PPP VCM's CREATE_PORT and ENABLE_PORT routines to start the protocol on the asynchronous device. The ENABLE_PORT call specifies the name of the asynchronous and the protocol to start. It is the ENABLE_PORT call which causes the PPP driver to commence the negotiation of the corresponding control protocol.
4. The protocol module's open is called to notify it of a request to start the control protocol. The open routine has a pointer to the VCIB of the corresponding protocol as one of it's parameters.
5. The protocol module calls the fsm_open routine to accept the connection.

6. The configuration options for the control protocol are negotiated. (These are documented in the preceding section.)
7. The protocol module is notified that data transmission is now possible via a call to the REPORT_EVENT routine with a PORT_USABLE event.
8. The protocol module now sends and receives data.

4
Compiling and Linking

An implementation gains access to the protocol (VCI) , control protocol and management interfaces through a vectors of routines. There is a separate vector for each interface. Each vector contains pointers to the routines for that interface. The routine PPPD$LIBRTNS returns a pointer to the vector for each of the interfaces. This function takes as an input parameter the a constant for the vector to be returned. The header file ppp_lib.h defines the prototype of the PPPD$LIBRTNS routine and constants for each of the vectors. The header files ppp_vci_if.h, fsm_if.h, and ppp_mgmt_if.h contain definitions of the vectors and the routines in them for the protocol, control protocol and management interfaces respectively.

Sample code for implementing and protocol and a control protocol can be found in SYS$EXAMPLES.

4.1 Interface Header Files

The following table lists the header files needed for implementing a protocol and/or a control protocol. The header file required for implementing an alternate management utility is also listed.

Table 4–1 PPP Interface Header Files

Header File	Description
ppp_vci_if.h	Contains definitions and prototypes for writing to the PPP VCI interface.
fsm_if.h	Contains definitions and prototypes for writing to the control protocol interface.
ppp_lib.h	Contains definitions and prototype for interfacing to PPP library routine.
pppusr.h	Contains definitions of PPP constants eg. PPP protocol types.
ppp_mgmt_if.h	Contains definitions and prototypes for to the management interface. Used to implement an alternate utility to manage PPP.

4.2 Linking

An implementation needs to link with the PPP_LIB.OBJ object module if it calls the PPPD$LIBRTNS functions This function is defined in the object module PPP_LIB, which is contained in the object library VMS$VOLATILE_PRIVATE_INTERFACES.OLB. VMS$VOLATILE_PRIVATE_INTERFACES.OLB is found in the SYS$SHARE: directory.