PSF - Q. 930/Q.931 (FT/HA)

Service Definition
1092144 1.2

PSF - Q.930/Q.931 (FT/HA)

Service Definition
1092144 1.2

Trillium Digital Systems, Inc.
12100 Wilshire Blvd., Suite 1800
Los Angeles, CA 90025-7118
Phone: +1 (310) 442-9222
Fax: +1 (310) 442-1162
Web: http://www.trillium.com

PSF - Q.930/Q.931 (FT/HA)
Service Definition
1092144 1.2

Preface

Objective

This document provides a detailed description of the services provided by the PSF - Q.930/Q.931 (FT/HA) software (p/n 1000144) designed by Trillium Digital Systems, Inc. This product is referred to as PSF - Q.930/Q.931 in the rest of the document.

Audience

Trillium assumes that the readers of this document are familiar with telecommunication protocols, specifically ISDN, Trillium's Fault Tolerant/High Availability (FT/HA) concepts, and Trillium's Q.930/Q.931 product.

Document Organization

This document is organized into the following sections:

Section	Description
1 Introduction	Provides an overview of the product. It also contains some terms used throughout the document.
2 Environment	Describes assumptions about the operating environment for the PSF - Q.930/Q.931 software
3 Interface Primitives	Explains the interface primitives at the PSF - Q.930/Q.931 layer interfaces
4 Interface Procedures	Defines the interface procedures at the PSF - Q.930/Q.931 layer interfaces
5 Porting	Describes the portation requirements for the PSF - Q.930/Q.931 software

Document Set

The suggested reading order for the PSF - Q.930 / Q.931 document set is:

1. *PSF - Q.930/Q.931 (FT/HA) Functional Specification*

   Highlights and describes the protocol and system characteristics of the software, including the memory characteristics and conformance details.

2. *PSF - Q.930/Q.931 (FT/HA) Service Definition*

   Describes the procedures and the layer manager interface used to pass information between the software and other software elements. The Interface Primitives section describes the services of the software. The Interface Procedures section describes and illustrates the flow of primitives and messages across the interfaces.

   Note: *Information on porting the software is contained in the Service Definition.*

3. *PSF - Q.930/Q.931 (FT/HA) Software Test Sample*

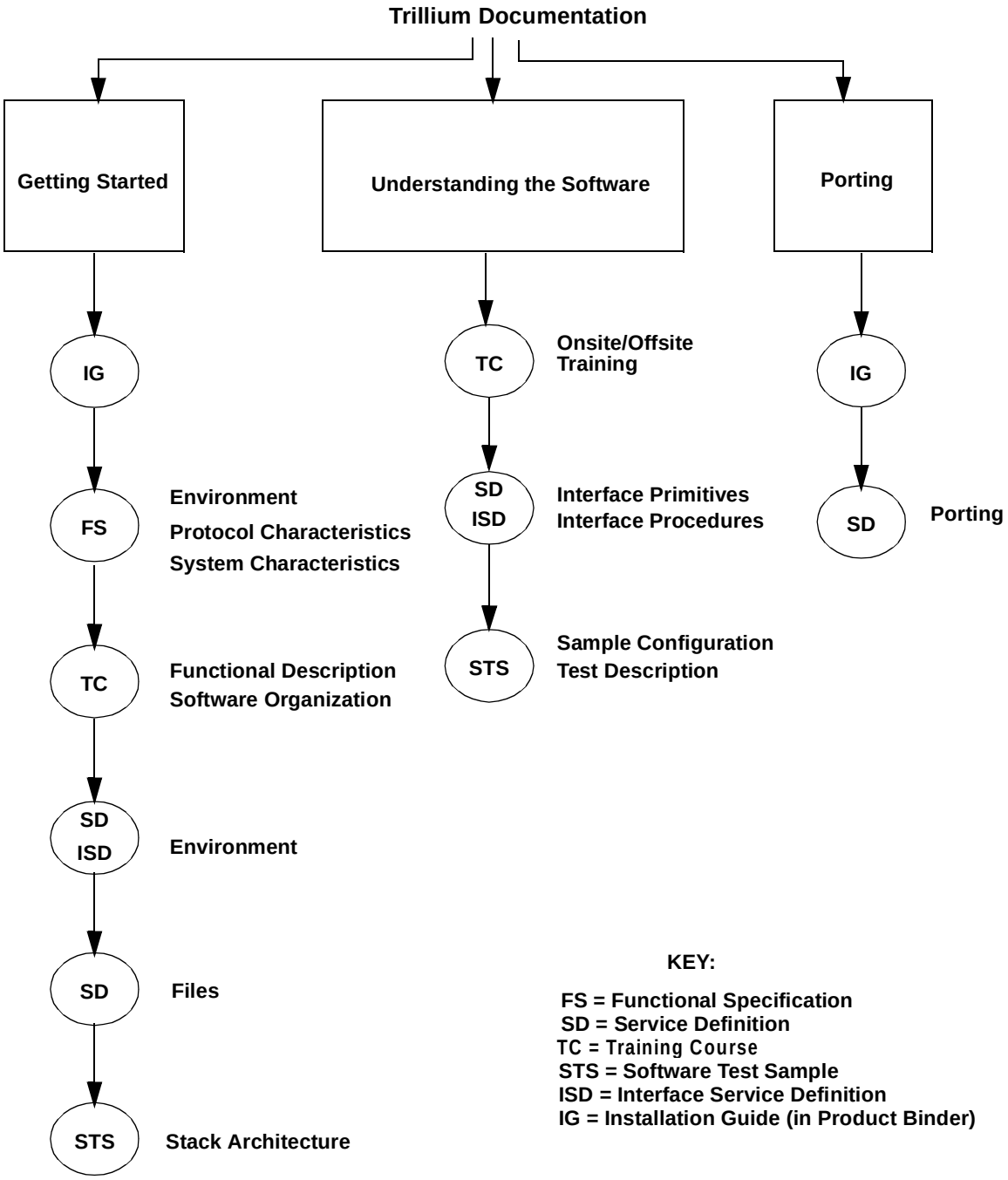
   Describes the sample files delivered with the product and the procedures to build a sample test. This test partially demonstrates the initialization, configuration, and execution of the product. It may contain data flow diagrams illustrating the correct operation of the software.

In addition to the above PSF documents, the following documents should also be read for a better understanding of the fault tolerant system:

4. *Fault Tolerant/High Availability (FT/HA) Core Functional Specification* (p / n 1091133).
5. *Fault Tolerant/High Availability (FT/HA) Core Service Definition* (p / n 1092133).

Using Trillium Documentation

The figure below illustrates the various approaches the user can take when utilizing the software documentation. First time users should read the documents under the **Getting Started** column; important sections and subsections are listed to the right of each document. For users familiar with the documentation but who need to look up certain points concerning the use of the software, the **Understanding the Software** column is suggested. The **Porting** column is for those users who are familiar with Trillium software and related telecommunications protocols and who wish to install the software immediately onto their operating systems.



Notations

This table displays the notations used in this document:

Notation	Explanation	Examples
Arial	**Titles**	**1.1 Title**
Palatino	Body text	This is body text.
Bold	**Highlights information**	**Loose coupling, tight coupling**, **upper layer interface**
ALL CAPS	CONDITIONS, MESSAGES	AND, OR CONNECT ACK
Italics	*Document names, emphasis*	*PSF - Q. 930/Q.931 (FT/HA) Service Definition* This adds *emphasis.*
Courier New Bold	**Code Filenames, pathnames**	**PUBLIC S16 ZqMiLzqCfgReq(pst, cfg)** **Pst *pst;** **CmPFthaMngmt *cfg;**

Release History

This table lists the history of changes in successive revisions to this document:

Version	Date	Initials	Description
1.2	12/28/99	nm	Update for software version 1.2
1.1	09/30/99	pk	Initial release

Contents

Illustrations

1 INTRODUCTION

This document provides a description of the services provided by the Protocol Specific Function (PSF) - Q.930 / Q.931 Fault Tolerant / High Availability (FT / HA) software designed by Trillium Digital Systems, Inc.

PSF - Q.930 / Q.931 adds the FT / HA functionality to Trillium's Q.930 / Q.931 (ISDN) product. From the Q.930 / Q.931 standpoint, PSF - Q.930 / Q.931 is a library of functions that is invoked only in a fault-tolerant environment. This document describes the primitives and the procedures supported by PSF - Q.930 / Q.931 at the layer management interface.

The PSF - Q.930 / Q.931 software provides interfaces to perform the following functions:

- Run-time state update of standby
- Warmstart of an Out-Of-Service (OOS) node to make it standby
- Controlled switchover of active and standby Q.930 / Q.931 nodes
- Forced switchover on the failure of an active Q.930 / Q.931 node

PSF implements functions as required to make the Q.930 / Q.931 layer fault tolerant. In a fault-tolerant environment, two copies of Q.930 / Q.931 are maintained, one as the active node and the other as the standby. Each of these nodes has a copy of PSF - Q.930 / Q.931. Only the active copy of PSF - Q.930 / Q.931 participates in the protocol execution. While handling an event, the active node can modify its internal states. The active PSF - Q.930 / Q.931 updates the state changes to the standby to keep the standby synchronized with the active. The active PSF updates only high-level, stable states to the standby during run time. This is to reduce the run-time update overhead and the complexity involved in the update procedure. Transient states (that is, the states that have a small duration) are only updated at the time of a controlled switchover.

The state update procedure can be initiated either by an active node or by the system manager. The active node initiates the state update procedure when its state changes as part of the normal protocol operation. The system manager initiates the state update procedure while performing warmstart or controlled switchover procedures. The PSF module on the standby node updates the protocol state information received from the active node onto the standby Q.930 / Q.931 to ensure that the two nodes are synchronized during run time.

During a forced switchover the standby resumes operation with its current, stable states. Any existing transient state information is lost during a forced switchover; therefore, the calls in the transient state are lost.

To initiate a switchover (controlled / forced) or other fault tolerance activities, the system manager sends a sequence of control requests to the PSF and the protocol layer of both the active and standby nodes. All such control requests supported by Q.930 / Q.931 and the PSF are described in detail in the following sections. The sequence of system manager control requests is beyond the scope of this document and is described in the *FT/ HA Core Service Definition*.

1.1 Terms and Definitions

The following terms are used in this document:

Term	Definition
Active node	A node that executes software to provide the necessary protocol functionality. The active node processes the protocol messages and updates the new state information in the standby node.
Controlled switchover	A procedure that makes a standby node active and an active node standby
Fault-tolerant node	A pair of nodes with replicated protocol layers. A fault-tolerant node can be in an active, standby, or Out-Of-Service (OOS) state.
Forced switchover	A procedure that makes a standby node active when an active node goes OOS
Node	A unit that has a processor(s) with private volatile memory inaccessible to all other nodes, and a private clock governing the execution of instructions on this processor. A node also has a network interface connecting it to a communication network using communication channels. The software governs the sequence of instructions executed on a node.
OOS node	An off-line node that has the ability to become an active or standby node
Run-time state update	The active Q.930 / Q.931 handles protocol events, which can result in internal state changes. The active PSF updates the standby with the state changes to keep the standby synchronized.
Standby node	A node that acts as a backup to an active node
Warmstart	A procedure that makes an OOS active node standby. An active node updates this new standby node with current information using a bulk update procedure.

1.2 Recommended Reading: FT/HA Service Definition

The *FT/HA Service Definition* describes system level fault-tolerant scenarios (for example, switchover and controlled switchover). This document can be referred to for a complete understanding of the functioning of various fault-tolerant layers in a protocol stack residing on different nodes. It describes various system entities that together achieve fault tolerance functionality, as well as the sequence of events that perform fault-tolerant procedures between the protocol layers and other system entities, such as controlled switchover and forced switchover.

Within the *FT/HA Service Definition*, the layer manager functions are split into three parts: system manager, system agent, and stack manager. Trillium provides both the system manager and the system agent components. If the user is not using Trillium's system manager and system agent, all interfaces with the system manager and system agent in the *FT/HA Service Definition* should be viewed as layer manager interfaces. The system manager, system agent, and stack manager can be implemented in a proprietary fashion by the user.

2 ENVIRONMENT

This section describes the environment in which the PSF - Q.930 / Q.931 software is designed to operate.

The Q.930 / Q.931 product conforms to Trillium Advanced Portability Architecture (TAPA). TAPA can be visualized as a box surrounded by four outer boxes — the box in the center represents the Q.930 / Q.931 (ISDN) software (see Figure 2-1 below). The four outer boxes represent other software to which Q.930 / Q.931 can be connected. The separation between the center box and outer boxes defines the interfaces across which Q.930 / Q.931 interacts with the other software.

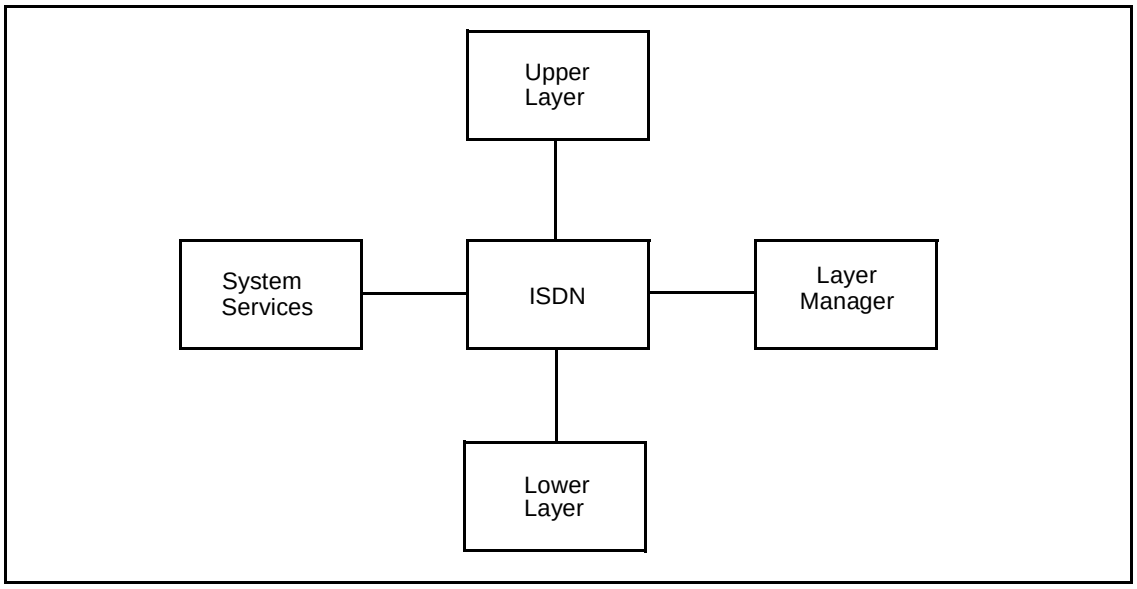


Figure 2-1: Trillium Advanced Portability Architecture (TAPA)

The PSF - Q.930 / Q.931 architecture differs from the architecture of other Trillium products that conform to TAPA, as it does not have an upper or lower layer. PSF - Q.930 / Q.931 does, however, provide the standard Layer Manager (LM) and System Services (SS) interfaces.

Figure 2-2 illustrates the architecture of the PSF - Q.930 / Q.931 software environment:

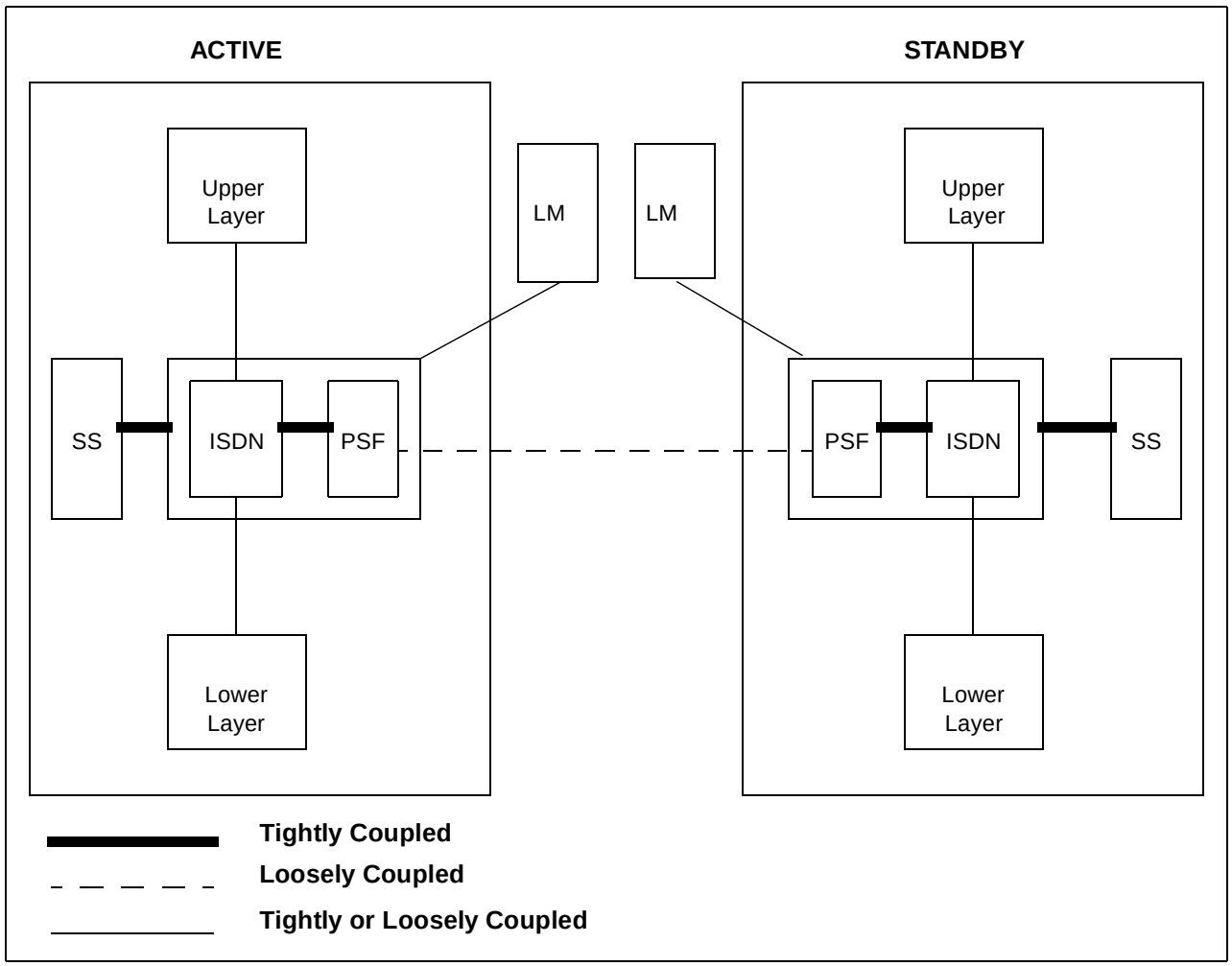


Figure 2-2: PSF - Q.930/Q.931 (FT/HA) software environment

The Q.930 / Q.931 — system services interface provides initialization, timer management, memory management, message, queue management, date and time management, and resource checking. The system services interface is always tightly coupled.

The PSF interface with the Q.930 / 931 protocol layer is an internal interface. The Q.930 / Q.931 invokes PSF functions to carry out run-time updates. This interface is always tightly coupled.

The PSF interface with the peer PSF - Q.930 / Q.931 is always loosely coupled. This interface allows the active PSF - Q.930 / Q.931 to communicate with the standby PSF - Q.930 / Q.931, and vice versa. The functions provided by this interface involve state updates from active to standby during run time, warmstart, and controlled switchover.

PSF - Q.930 / Q.931 interacts with the layer manager and the peer PSF using a set of primitive functions. These primitives take the form of requests, indications, responses, and confirms, and completely define the interaction between layers.

3 INTERFACE PRIMITIVES

This section describes the primitives at the PSF - Q.930 / Q.931 interfaces.

3.1 General

The data types and common structures for the PSF - Q.930 / Q.931 software are described in this section.

3.1.1 Data Types

The sizes of the data types in the primitives are defined as:

Data Type	Number of 8-bit bytes	Sign
S8	1	Signed
U8	1	Unsigned
S16	2	Signed
U16	2	Unsigned
S32	4	Signed
U32	4	Unsigned
PTR	As required	Unsigned

Note: *The size of* **PTR** *depends on the specific machine to which the software is ported.*

Other **typedef**s used are:

Mnemonic	Data Type	Description
Bool	**U8**	Boolean
Cntr	**S32**	Statistics counter

3.1.2 Common Structures

This section describes the common structures used by all the interface primitives.

3.1.2.1 Post

The destination post structure is used at all the interface primitives by the sending layer to indicate to system services the identity of the destination layer and the route. This field is used by system services for intertask communication.

Once the buffer reaches the destination layer, this parameter can be used to identify the sending layer. The post structure has the following format:

```
typedef struct pst          /* parameters for SPstTsk */
{
   ProcId dstProcId;        /* destination processor id (U16) */
   ProcId srcProcId;        /* source processor id      (U16) */

   Ent dstEnt;              /* destination entity       (U8)  */
   Inst dstInst;            /* destination instance     (U8)  */
   Ent srcEnt;              /* source entity            (U8)  */
   Inst srcInst;            /* source instance          (U8)  */

   Prior prior;             /* priority                 (U8)  */
   Route route;             /* route                    (U8)  */
   Event event;             /* event                    (U8)  */
   Region region;           /* region                   (U8)  */

   Pool pool;               /* pool                     (U8)  */
   Selector selector;       /* selector                 (U8)  */
   U16 spare1;              /* spare 1                  (U16) */
} Pst;
```

dstProcId

Destination processor ID. This parameter is reconfigurable.

srcProcId

Source processor ID. This parameter is used for posting messages across a loosely coupled interface.

dstEnt

Destination layer entity ID. This parameter is reconfigurable.

dstInst

Destination layer instance ID. This is reconfigurable.

srcEnt

Source entity ID. This parameter is used for posting messages across a loosely coupled interface.

srcInst

Source instance ID. This parameter is used for posting messages across a loosely coupled interface.

prior

Priority. This parameter is reconfigurable and is not used currently.

route

Route. This parameter is reconfigurable and is not used currently.

event

Posted event ID.

region

Memory region.

pool

Pool number.

selector

Selector for resolving tight coupling or loose coupling with the destination entity.

spare1

Spare for alignment. Not used.

3.2 System Services Interface

This section provides a brief description of the System Services Interface (SSI) functions that are used by PSF - Q.930 / Q.931. PSF - Q.930 / Q.931 uses a subset of the functions that system services provides. SSI is described in detail in the *System Services Interface Service Definition.* The system services interface functions used by the PSF - Q.930 / Q.931 are described below.

Initialization

The initialization function is called by system services to initialize a task (layer). The following functions are used for initialization:

Name	Description
SRegInit	Register initialization task
inActvInit	Initialization task for Q.930 / Q.931
zqActvInit	Initialization task for PSF - Q.930 / Q.931

Task Scheduling

The task scheduling functions are called to register, activate, and terminate a task. These functions are required only if the PSF - Q.930 / Q.931 is loosely coupled with any of the interfaces. The following functions are used for task scheduling:

Name	Description
SRegActvTsk	Register layer activation task
SPstTsk	Post a message buffer to a destination task
SExitTsk	Cleanup and exit a task
SFndProcId	Find the processor ID on which a task is running
inActvTskNew	Layer activation task for Q.930 / Q.931

Note: *The PSF - Q.930/Q.931 layer is not registered separately. When the* **inActvTskNew** *function is invoked, it is checked to see if it is a PSF - Q.930/Q.931 event. If it is, the* **inActvTskNew** *function invokes* **zqActvTsk**.

Structure Memory Management

The structure memory management functions allocate and deallocate variable sized buffers to be used as structures. The following functions are used for memory management:

Name	Description
SGetSMem	Allocate structure memory
SGetSBuf	Allocate buffer from structure memory
SPutSBuf	Deallocate buffer and return it to structure memory

Timer Management

Timer functions are called by system services periodically to activate tasks. The following functions are used for timer management:

Name	Description
SRegTmr	Register timer activation task
SDeregTmr	Deregister timer activation task
zqActvTmr	PSF - Q.930 / Q.931 timer activation function. This function is provided in PSF - Q.930 / Q.931 and is called back by system services.

Queue Management

The queue management functions initialize, add, and remove messages to and from queues. The structure of a queue is not known to a layer and is specific to system services. The following functions are used for queue management:

Name	Description
SInitQueue	Initialize a queue
SQueueLast	Add a message to the tail of a queue
SQueueFirst	Add a message to the head of a queue
SDequeueFirst	Remove a message from the head of a queue
SDequeueLast	Remove a message from the tail of a queue
SFndLenQueue	Find the length of a queue
SFlushQueue	Flush queue

Message Management

The message management functions initialize, add, and remove data to and from messages utilizing dynamic buffers. The structure of a message is not known to a layer and is specific to system services. The following functions are used for message management:

Name	Description
SGetMsg	Allocate message from dynamic memory
SPutMsg	Deallocate message from dynamic memory
SAddPstMsg	Add a byte to the tail of a message
SAddPreMsg	Add a byte to the head of a message
SAddPreMsgMult	Add multiple bytes to the head of a message
SAddPstMsgMult	Add multiple bytes to the tail of a message
SRemPreMsg	Remove a byte from the head of a message
SRemPreMsgMult	Remove multiple bytes from the head of a message
SRemPstMsg	Remove a byte from the tail of a message
SExamMsg	Examine message
SFndLenMsg	Find length of message
SPrntMsg	Print the contents of the message
SCatMsg	Concatenate two messages
SRepMsg	Replace message
SInitMsg	Initialize message
SSegMsg	Segment message
SCpyMsgMsg	Copy from message to message

Packing/Unpacking

These routines are used for the packing and unpacking of interface primitive parameters. These functions are required only if any of the interfaces are loosely coupled. The following packing and unpacking functions are used:

Name	Description
SPkU8	Add an unsigned 8-bit value to the head of a message
SPkU16	Add an unsigned 16-bit value to the head of a message
SPkU32	Add an unsigned 32-bit value to the head of a message
SpkS8	Add a signed 8-bit value to the head of a message
SPkS16	Add a signed 16-bit value to the head of a message
SPkS32	Add a signed 32-bit value to the head of a message
SUnpkU8	Remove an unsigned 8-bit value from the head of a message
SUnpkU16	Remove an unsigned 16-bit value from the head of a message
SUnpkU32	Remove an unsigned 32-bit value from the head of a message
SUnpkS8	Remove a signed 8-bit value from the head of a message
SUnpkS16	Remove a signed 16-bit value from the head of a message
SUnpkS32	Remove a signed 32-bit value from the head of a message

Miscellaneous

The miscellaneous functions are used for date and time management, error handling, and resource availability checking. The following miscellaneous functions are used:

Name	Description
SLogError	Handle a fatal system error
SChkRes	Report available system resources such as message buffer memory
SPrint	Print a pre-formatted string
SGetDateTime	Get current date and time
SGetSysTime	Get system time in system ticks

For a detailed description of the system services listed above, refer to the *System Services Interface Service Definition*. The functions required by PSF - Q.930/Q.931 are also specified in the *PSF - Q.930/Q.931 (FT/HA) Release Notes.*

3.3 Interface with Layer Manager

This section provides a brief description of the layer manager (**Lzq**) interface functions that are used by PSF - Q.930 / Q.931.

3.3.1 General

The layer manager interface provides the following functions:

Configuration

Functions to configure protocol layer resources:

Function	Description
ZqMiLzqCfgReq	Configuration request
ZqMiLzqCfgCfm	Configuration confirm

Control

Functions to activate and deactivate protocol resources:

Function	Description
ZqMiLzqCntrlReq	Control request
ZqMiLzqCntrlCfm	Control confirm

Solicited Status

Functions to indicate current state of the protocol layer:

Function	Description
ZqMiLzqStaReq	Status request
ZqMiLzqStaCfm	Status confirm

Unsolicited Status

A function to indicate change in status of the protocol layer:

Function	Description
ZqMiLzqStaInd	Status indication

3.3.2 Common Structures

In addition to the specific structures for the respective primitive, each layer manager primitive uses the following common structures.

3.3.2.1 Header Structure

Each management primitive consists of a header structure followed by a structure specific to the type of primitive invoked. The header structure has the following format:

```
typedef struct tds_header       /* header */
{
   U16 msgLen;                  /* message length   - optional */
   U8 msgType;                  /* message type     - mandatory */
   U8 version;                  /* version          - optional */
   U16 seqNmb;                  /* sequence number  - optional */
   EntityId entId;              /* entity id        - mandatory */
   ElmntId elmId;               /* element id       - mandatory */
#ifdef LMINT3
   TranId transId;              /* transaction Id - mandatory */
   Resp response;               /* response parameters - mandatory */
#endif /* LMINT3 */
} Header;
```

The **msgLen**, **msgType**, **version**, **seqNmbr**, and **entId** fields are not used by PSF - Q.930/Q.931

elmId

Element ID. Identifies a specific element to which the management message is to be applied.

```
typedef struct elmntId      /* element id */
{
   Elmnt      elmnt;
   ElmntInst1 elmntInst1;
   ElmntInst2 elmntInst2;
   ElmntInst3 elmntInst3;
} ElmntId;
```

elmnt

Element. Allowable values:

Value	Description
STGEN	General
STPEERSAP	Peer SAP

elmntInst1, elmntInst2, and elmntInst3

These parameters are not used in PSF - Q.930/Q.931.

transId

Transaction ID. Specifies the sequence number of the layer manager message. The layer manager assigns a transaction ID to every request issued to the PSF and PSF sends back the same transaction ID to the layer manager in the response. The layer manager can use this field to correlate the responses from with the requests previously sent.

response

Response structure. This field is used by PSF - Q.930 / Q.931 to send replies to the sender of the layer manager requests. The response structure has the following format:

```
typedef struct resp
{
   Selector selector;           /* selector */
   Priority prior;              /* priority */
   Route route;                 /* route */
   MemoryId mem;                /* memory */
}Resp;
```

selector

Response post structure selector field.

prior

Response post structure prior field.

route

Response post structure route field.

mem

Response post structure region and pool.

3.3.2.2 Date and Time

The date and time structure is used in many layer manager primitives to report the date and time when the primitive is called. It has the following format:

```
typedef struct dateTime           /* date and time */
{
   U8 month;                      /* month */
   U8 day;                        /* day */
   U8 year;                       /* year */
   U8 hour;                       /* hour - 24 hour clock */
   U8 min;                        /* minute */
   U8 sec;                        /* second */
   U8 tenths;                     /* tenths of second */
} DateTime;
```

month

The current month.

day

The current day.

year

The current year. Allowable values: 0 to 255.

Trillium products represent the year as an offset from the year 1900. For example, 1999 is represented as 99, 2003 is represented as 103, and so on. Thus, the years from 1900 to 2155 can be represented without causing an overflow. If necessary, the customer can add the value 1900 to the **year** field to get the actual calendar year.

hour

The current hour.

min

The current minute.

sec

The current second.

tenths

The tenths of a second.

3.3.2.3 Common Status

This structure is used only in the primitives sent from PSF - Q.930/Q.931 to the layer manager as a response to the management request. The common status structure has the following format:

```
typedef struct cmStatus
{
   U16 status;       /* status of request */
   U16 reason;       /* failure reason */
}CmStatus;
```

status

Status to indicate success or failure of a layer manager request coming to PSF - Q.930/Q.931. Allowable values:

```
#define LCM_PRIM_OK           0       /* OK, activity completed */
#define LCM_PRIM_NOK          1       /* NOK, activity completed */
#define LCM_PRIM_OK_NDONE     2       /* OK, activity not completed */
```

reason

Reason for the failure of the layer manager request. Allowable values:

```
#define LCM_REASON_NOT_APPL          0  /* not applicable */
#define LCM_REASON_MEM_NOAVAIL       4  /* memory not avail */
#define LCM_REASON_INVALID_ELMNT     5  /* invalid hdr.elmnt */
#define LCM_REASON_REGTMR_FAIL       7  /* timer registration
                                           failed */
#define LCM_REASON_GENCFG_NOT_DONE   8  /* gene config not done */
#define LCM_REASON_INVALID_ACTION    9  /* invalid control action */
#define LCM_REASON_INVALID_SUBACTION 10 /*invalid control subaction */
#define LCM_REASON_INVALID_STATE     11 /* invalid state */
#define LCM_REASON_NEG_CFM           16 /* negative confirmation */
#define LCM_REASON_UPDTMR_EXPIRED    17 /* update timer expired */
#define LCM_REASON_MISC_FAILURE      18 /* miscellaneous failures */
#define LCM_REASON_PEERCFG_NOT_DONE  21 /* swft peer sap not cfgd */
#define LCM_REASON_PRTLYRCFG_NOT_DONE   22 /* swft -portable lyr not
                                              configured */
#define LCM_REASON_LYR_SPECIFIC      256   /* protocol specific
                                              presents */
```

3.3.3 Layer Manager Interface Primitives

This section describes the primitives at the PSF - Q.930 / Q.931 - layer manager interface.

3.3.3.1 ZqMiLzqCfgReq

Name:

Configuration Request

Direction:

Layer manager to PSF - Q.930 / Q.931

Supplied:

Yes

Synopsis:

```
PUBLIC S16 ZqMiLzqCfgReq(pst, cfg)
Pst      *pst;
CmPFthaMngmt *cfg;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

cfg

Pointer to the configuration structure. The configuration structure has the following format:

```
typedef struct cmPFthaMngmt
{
   Header hdr;
   CmStatus cfm;   /* confirm structure */
   union
   {
     struct
     {
        union
        {
           CmPFthaGenCfg genCfg;          /* general */
           CmPFthaSAPCfg peerSAPCfg;      /* peer sap */
        }s;
     }cfg;  /* configuration */
     .
     .

   }t;
}CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure." The following fields must be initialized as shown:

```
hdr.msgType              - TCFG
hdr.entId                - entity id of Q.930/Q.931
hdr.inst                 - instance of Q.930/Q.931
hdr.elmnt.elmnt          - STGEN or STPEERSAP
hdr.transId              - transaction Id for confirm
hdr.response.selector    - selector to be used for confirm
hdr.response.route       - route to be used for confirm
hdr.response.prior       - priority to be used for confirm
hdr.response.mem.pool    - memeory pool for confirm
hdr.response.mem.region  - region for confirm
```

cfm

See Section 3.3.2.3, "Common Status."

cfg

The structure that contains the union of all the specific configuration structures. The format of this structure is:

```
struct
{
   union
   {
      CmPFthaGenCfg genCfg;          /* general */
      CmPFthaSAPCfg peerSAPCfg;      /* peer sap */
   }s;
}cfg;                               /* configuration */
```

genCfg

General configuration structure.

```
typedef struct cmPFthaGenCfg
{
   S16 timeRes;                /* timer resolution */
   ProcId vProcId;             /* virtual processor Id */
   MemoryId mem;               /* self region and pool */
   U8  updateOption;           /* update option */
   Pst smPst;                  /* stack manager post structure */
} CmPFthaGenCfg;
```

timeRes

Timer resolution in ticks. System services invokes the PSF - Q.930 / Q.931 timer function, **ziPrcPeerSapTq**, in every **timeRes**. This field is not reconfigurable.

Allowable values: up to 32767 ticks.

vProcId

Virtual processor ID of the node on which Q.930/Q.931 and PSF - Q.930/Q.931 reside. The upper and lower layers use the **vProcId** of Q.930/Q.931 to send messages to the layer for loosely coupled interfaces. An external entity is responsible for converting **vProcId** to the physical processor ID of the currently active Q.930/Q.931 node before delivering the message. PSF - Q.930/Q.931 initializes the source processor ID of all the lower SAPs and the upper SAPs of Q.930/Q.931 with the **vProcId**.

Allowable values: 0 to 65535. This field is not reconfigurable.

mem

Region and pool to be used by PSF - Q.930/Q.931 to mail a message to itself during warmstart and controlled switchover.

Allowable values: 0 to 255. This field is not reconfigurable.

updateOption

Configuration parameter to determine if state updating is to be performed for unanswered calls. This field is reconfigurable.

Allowable values: **0** - Do not update **1** - Update

smPst

Post structure for sending alarms to the layer manager. This field is reconfigurable.

peerSAPCfg

Peer SAP structure.

```
typedef struct cmPFthaSAPCfg
{
   Region region;              /* memory region for peer */
   Pool pool;                  /* memory pool for peer */
   ProcId dstProcId;           /* peer processor id */
   Ent dstEnt;                 /* peer entity id */
   Inst dstInst;               /* peer instance id */
   Priority prior;             /* peer post priority */
   Route route;                /* peer post route */
   Selector selector;          /* peer selector */
   TmrCfg tUpdCompAck;         /* update completion timer */
   U32 maxUpdMsgSize;          /* maximum size of the update message */
} CmPFthaSAPCfg;
```

region

Memory region for allocating message buffers for mailing a message to the peer PSF - Q.930/Q.931. This field is reconfigurable.

Allowable values: 0 to 255.

pool

Memory pool for allocating message buffers for mailing a message to the peer PSF - Q.930 / Q.931. This field is reconfigurable.

Allowable values: 0 to 255.

dstProcId

Peer to PSF - Q.930 / Q.931 physical processor ID. This field is reconfigurable. Allowable values: 0 to 65535.

dstEnt

Peer PSF - Q.930 / Q.931's entity ID. This field is reconfigurable. Allowable values: 0 to 255.

dstInst

Peer PSF - Q.930 / Q.931's instance ID. This field is reconfigurable. Allowable values: 0 to 255.

prior

Priority for post structure of the peer SAP. This field is reconfigurable. Not currently used.

route

Route for the post structure of the peer SAP. This field is reconfigurable. Allowable value: **RTESPEC** (default value).

selector

Selector for the post structure of the peer SAP. This field is reconfigurable. Not currently used.

tUpdCompAck

Update completion acknowledgment timer. Value for the timer started by the active PSF - Q.930 / Q.931 to wait for a confirm from the standby for warmstart or controlled switchover state update. This field is reconfigurable.

Allowable values: 0 to 65535.

```
typedef struct tmrCfg /* timer configuration */
 {
     Bool enb; /* flag to enable/disable timer */
      U16 val; /* value of the timer */
}TmrCfg;
```

maxUpdMsgSize

Maximum message update size. This field is reconfigurable.

Allowable values: *x* to (2^{32} - 1)

Note: *The minimum size of the update message must be greater than the maximum size of a table to be packed by PSF - Q.930/Q.931.*

Description:

The stack manager uses this primitive to send the general and peer SAP configuration to the PSF - Q.930 / Q.931 module. The stack manager configures the timer resolution and the timer values for warmstart and controlled switchover procedures. PSF - Q.930 / Q.931 calls **SRegTmr** with the configured timer resolution to register the timer function with the system services. For more information, see Section 4.2.1, "Management Configuration."

Returns:

ROK OK

RFAILED Failed

3.3.3.2 ZqMiLzqCfgCfm

Name:

Configuration Confirm

Direction:

PSF - Q.930 / Q.931 to layer manager

Supplied:

No

Synopsis:

```
PUBLIC S16 ZiMiLziCfgCfm(pst, cfm)
Pst *pst;
CmPFthaMngmt *cfm;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

cfm

Common management structure. Only **hdr** and **cfm** are used by this primitive.

```
typedef struct cmPFthaMngmt
{
   Header hdr;
   CmStatus cfm;
   union
   {
      ...
      ...
   }t;
 }CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure."

cfm

See Section 3.3.2.3, "Common Status." Allowable values for its fields are:

status

Status to indicate success or failure of a layer manager request coming to the PSF - Q.930 / Q.931 . Allowable values:

```
LCM_PRIM_OK           /* configuration request is successful */
LCM_PRIM_NOK          /* configuration request is not successful */
```

reason

Reason for the failure. Allowable values:

```
LCM_REASON_REGTMR_FAIL        /* Timer registeration failed */
LCM_REASON_GENCFG_NOT_DONE    /* General configuration not done */
LCM_REASON_INV_PAR_VAL        /* Upd msg size parameter invalid */
LCM_REASON_QINIT_FAIL         /* Queue init failed */
LCM_REASON_INVALID_ELMNT      /* Invalid element in config req */
LCM_REASON_NOT_APPL           /* Reason not applicable: used
                                 with LCM_PRIM_OK status */
LCM_REASON_PRTLYRCFG_NOT_DONE /* Portable layer configuration
                                 not done */
```

Description:

PSF - Q.930 / Q.931 uses this primitive to acknowledge the configuration request primitive (**ZqMiLzqCfgReq**) sent by the stack manager. PSF - Q.930 / Q.931 indicates the success or failure of the configuration request received from the layer manager. For more details on the configuration procedure, see Section 4.3.1, "Management – Configuration."

Returns:

ROK OK

RFAILED Failed

3.3.3.3 ZqMiLzqStaReq

Name:

Status Request

Direction:

Layer manager to PSF - Q.930 / Q.931

Synopsis:

```
PUBLIC S16 ZqMiLzqStaReq(pst, sta)
Pst          *pst;
CmPFthaMngmt *sta;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

sta

Pointer to management structure. Status structure has the following format:

```
typedef struct cmPFthaMngmt
{
   Header   hdr;      /* header */
   CmStatus cfm;      /* confirm */
   union              /* values are returned in sta structure*/
   {
    .
    .
    .
   } t;
 } CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure."

Description:

The stack manager uses this primitive to get the status of the protocol layer and the status of the peer SAP from the PSF - Q.930 / Q.931. For more information see Section 4.3.2, "Management – Solicited Status."

Returns:

00 OK

3.3.3.4 ZqMiLzqStaCfm

Name:

Status Confirm

Direction:

PSF - Q.930/Q.931 to layer manager

Supplied:

No

Synopsis:

```
PUBLIC S16 ZqMiLzqStaCfm(pst, sta)
Pst *pst;
CmPFthaMngmt *sta;
```

Parameters:

pst

See Section 3.1.2.1, "Post." Confirmation is sent to the sender of **ZqMiLzqStaReq**.

sta

Pointer to management structure. Status structure has the following format:

```
typedef struct cmPFthaMngmt
{
   Header    hdr;
   CmStatus  cfm;
   union
   {
   .
   .
      struct
      {
         DateTime dt;
         union
         {
            U8 genSta;                      /* general status */
            CmPFthaPeerSapSta peerSapSta;  /* peer sap status */
            SystemId sysId;                 /* system id */
          }s;
      }sta;
   .
   .
   }t;
}CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure." **transId** field should be the same as that received in **ZqMiLzqStaReq**.

cfm

See Section 3.3.2.3, "Common Status." Allowable values for its fields are:

status

Status to indicate success or failure of a layer manager request coming to the PSF - Q.930 / Q.931. Allowable values:

```
LCM_PRIM_OK           /* configuration request is successful */
LCM_PRIM_NOK          /* configuration request is not successful */
```

reason

If the status is **LCM_PRIM_NOK**, this field specifies the reason for the request's failure.

```
LCM_REASON_INVALID_ELMNT  /* Invalid element in config req */
LCM_REASON_NOT_APPL       /* Reason not applicable: used with
                             LCM_PRIM_OK status */
```

dt

See Section Section 3.3.2.2, "Date and Time."

genSta

Status of the Q.930 / Q.931 protocol layer. Allowable values:

```
00   ACTIVE
01   STANDBY
02   OOS
```

peerSapSta

Specifies the status of the peer SAP. It has the following format:

```
typedef struct cmPFthaPeerSapSta
{
   U8 bndState;             /* bind state */
   U8 updState;             /* update state */
}CmPFthaPeerSapSta;
```

bndState

Bind status of the peer SAP. Allowable values:

```
CMPFTHA_BND             /* SAP is bound */
CMPFTHA_UBND            /* SAP is not bound */
```

updState

Update status of the peer SAP. Allowable values:

```
CMPFTHA_IDLE            /* No activity at peer SAP */
CMPFTHA_WRMSTRT         /* Peer PSF under warmstart */
CMPFTHA_SYNC            /* Peer PSF under synchronization */
CMPFTHA_WRMSTRT_WAIT    /* waiting for ack from peer PSF for
                           warmstart completion */
CMPFTHA_SYNC_WAIT       /* waiting for ack from peer PSF for
                           synchronization completion */
```

sysId

Specifies the current version of the PSF - Q.930 / Q.931 software. It has the following format:

```
typedef struct systemId
{
   S16 mVer;
   S16 mRev;
   S16 bVer;
   S16 bRev;
   S8 *ptNmb;
}SystemId;
```

mVer

Main version.

mRev

Main revision.

bVer

Branch version.

bRev

Branch revision.

ptNmb

Pointer to the part number.

Description:

The PSF - Q.930 / Q.931 uses this primitive to return the status of the protocol layer, the status of the peer SAP, or the version of the current PSF - Q.930 / Q.931 software to the layer manager. If the status request is successful, then **`cfm.status`** is **`LCM_PRIM_OK`** in **`ZqMiLzqStaCfm`**; otherwise, **`cfm.status`** is **`LCM_PRIM_NOK`**. For more information, see Section 4.3.2, "Management – Solicited Status."

Returns:

`ROK` OK

`RFAILED` Failed

3.3.3.5 ZqMiLzqStaInd

Name:

Status Indication

Direction:

PSF - Q.930/Q.931 to layer manager

Supplied:

No

Synopsis:

```
PUBLIC S16 ZqMiLzqStaInd(pst, usta)
Pst *pst;
CmPFthaMngmt *usta;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

usta

Pointer to unsolicited status indication structure. It has the following format:

```
typedef struct cmPFthaMngmt
{
   Header hdr;
   CmStatus cfm;
   union
   {
   .
   .
   .
     struct
     {
        CmAlarm alarm;                        /* alarm structure */
        U8  evntParm[CMPFTHA_USTA_EP_MAX];    /* event parameters */
     }usta;
   .
   .
   .
   }t;
}CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure."

cfm

Not used in this primitive.

alarm

Common alarm structure. It has the following format:

```
typedef struct cmAlarm
{
   DateTime dt;      /* data and time */
   U16 category;     /* alarm category*/
   U16 event;        /* alarm event */
   U16 cause;        /* alarm cause */
}CmAlarm;
```

dt

See Section 3.3.2.2, "Date and Time."

category

Category of alarm generated by the PSF - Q.930 / Q.931. Allowable values:

Allowable Values	Description
LCM_CATEGORY_RESOURCE	System resources category. An alarm in this category is generated if an error occurs during allocation / deallocation / addition / subtraction of system resources.
LCM_CATEGORY_INTERFACE	Interface category. This category indicates the occurrence of an error during the handing of interface events. An alarm in this category is generated if any interface parameter is in error or the interface event is not expected in the current state of the protocol.
LCM_CATEGORY_PSF_FTHA	PSF category. An alarm in this category is generated if an error occurs during normal PSF operations.

evnt

Type of alarm generated by PSF - Q.930/Q.931. Allowable values:

Allowable Values	Description
LCM_EVENT_INV_TMR_EVT	Generated by the standby PSF when a protocol timer expires on the standby node. The standby should not run any timer.
CMPFTHA_SEQERR	Reported by the standby PSF when it detects a run-time sequence error. The stack manager should make the standby OOS on getting this event.
CMPFTHA_MEM_FAILURE	Reported by active or standby PSF whenever it encounters memory allocation failures
CMPFTHA_UPDMSG_ERR	Reported by the standby PSF to indicate that a wrong update message has been received from the active PSF

cause

This field specifies the cause of alarm generated by the PSF. Allowable values:

```
LCM_CAUSE_PROT_NOT_ACTIVE     /* protocol not active */
LCM_CAUSE_UNKNOWN             /* unknown */
```

Possible combinations of **category**, **event**, and **cause** of each alarm are shown in the table below:

Category	Event	Cause
LCM_CATEGORY_RESOURCE	CMPFTHA_MEM_FAILURE	CAUSE_UNKNOWN
LCM_CATEGORY_PSF_FTHA	CMPFTHA_SEQERR	CAUSE_UNKNOWN
LCM_CATEGORY_PSF_FTHA	CMPFTHA_UPDMSG_ERR	CAUSE_UNKNOWN
LCM_CATEGORY_INTERFACE	LCM_EVT_INV_TMR_EVT	LCM_CAUSE_PROT_NOT_ACTIVE

evntParm

Event parameters. This field provides more details about the alarm. This field is not currently used.

Description:

PSF - Q.930 / Q.931 uses this primitive to report alarms to the stack manager. For example, when the standby PSF - Q.930 / Q.931 detects sequence error in the run-time state update messages, it sends **CMPFTHA_SEQERR** to the stack manager. For more information, see Section 4.3.3, "Management – Unsolicited Status."

Returns:

ROK OK

RFAILED Failed

3.3.3.6 ZqMiLzqCntrlReq

Name:

Control Request

Direction:

System manager to PSF - Q.930/Q.931

Supplied by:

Yes

Synopsis:

```
PUBLIC S16 ZqMiLzqCntrlReq(pst, cntrl)
Pst *pst;
CmPFthaMngmt *cntrl;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

cntrl

Pointer to control structure. It has the following format:

```
typedef struct cmPFthaMngmt
{
   Header hdr;
   CmStatus cfm;
   union
   {
   .
   .
      struct
      {
         DateTime dt;              /* date & time */
         U8 action;                /* action */
         U8 subAction;             /* subaction */
         union
         {
            CmPFthaDbgCntrl umDbg;  /* debug control */
         }ctlType;
      }cntrl;
   .
   .
   .
   }t;
}CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure."

cfm

Not used in this primitive.

dt

See Section 3.3.2.2, "Date and Time."

action

Control action.

subAction

Subaction.

The following table lists the allowable values for the **action**, **subaction**, and **elmnt** (**hdr.elmId.elmnt**) parameters.

The destination column shows the allowable combination values for the three parameters. Unlisted combinations in the following table are not allowed by PSF - Q.930 / Q.931:

Action	Subaction	elmnt	Destination	Description
AENA	**SAUSTA**	**STGEN**	Active, standby	Enable alarms
	SADBG	**STGEN**	Active, standby	Enable debug prints for the specified type **(see Note 1)**
ADISIMM	**SAUSTA**	**STGEN**	Active, standby	Disable alarms
	SADBG	**STGEN**	Active, standby	Disable debug prints for the specified type **(see Note 1)**
AUBND_DIS		**STPEERSAP**	Active	Disable the peer SAP
AGO_ACT	**SAENA_PEER_SAP**	**STGEN**	Standby	Make the protocol layer active and enable the peer SAP
	SADIS_PEER_SAP		Standby, OOS	Make the protocol layer active and disable the peer SAP
AGO_SBY	**SAENA_PEER_SAP**	**STGEN**	Active, OOS	Make the protocol layer standby and enable the peer SAP

Action	Subaction	elmnt	Destination	Description
ASYNCHRONIZE		**STGEN**	Active	Synchronize the peer to perform controlled switchover
AWARMSTART		**STGEN**	Active	Warmstart the peer to make it standby from OOS
AABORT		**STGEN**	Active	Abort the ongoing warmstart or synchronization procedure
ASHUTDOWN		**STGEN**	Active, standby	Shutdown operations. PSF - Q.930 / Q.931 deallocates all the allocated resources

Note 1: *Multiple debug levels can be specified in a single control request. For example, (***DBGMASK_MI | DBGMASK_PI***) is a valid* **dbgMask** *value.*

Note 2: *When the stack manager sends a control request to the active PSF - Q.930/Q.931 to warmstart or synchronize the peer layer, the active responds with two confirmations. The first confirmation indicates that the control request is accepted but not yet completed, and the second indicates the final result in terms of success or failure. The stack manager can send an abort control request before getting the final response.*

umDbg

PSF - Q.930 / Q.931 debug structure. It has the following format:

```
typedef struct cmPFthaDbgCntrl
{
   U32 dbgMask;   /* debug mask */
}CmPFthaDbgCntrl;
```

dbgMask

Debug mask. The default mask is **0**—that is, all the debug prints are disabled. The prints for the following debug levels can be enabled:

```
0                    /* all debug prints are disabled */
DBGMASK_MI           /* layer management interface */
DBGMASK_PI           /* peer interface */
DBGMASK_PLI          /* PSF protocol layer interface */
LZI_DBGMASK_PACK     /* debug mask for packing functions */
LZI_DBGMASK_UNPACK   /* debug mask for unpacking functions */
```

Description:

The system manager sends control requests to PSF - Q.930 / Q.931 for performing the fault tolerance procedures described in Section 4.3.4, "Management–Control." PSF - Q.930 / Q.931 sends confirmation for the control request via the **`ZqMiLzqCntrlCfm`** primitive.

Returns:

`ROK` OK

`RFAILED` Failed

3.3.3.7 ZqMiLzqCntrlCfm

Name:

Control Confirm

Direction:

PSF - Q.930 / Q.931 to system manager

Supplied:

No

Synopsis:

```
PUBLIC S16 ZqMiLzqCntrlCfm(pst, cfm)
Pst *pst;
CmPFthaMngmt *cfm;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

cfm

Common management structure. Only the **hdr** and **cfm** fields are used by this primitive.

```
typedef struct cmPFthaMngmt
{
   Header hdr;
   CmStatus cfm;
   union
   {
      ...
      ...
   }t;
 }CmPFthaMngmt;
```

hdr

See Section 3.3.2.1, "Header Structure."

cfm

See Section 3.3.2.3, "Common Status." Allowable values for its fields are:

status

Status to indicate the success or failure of a layer manager request coming to the PSF - Q.930 / Q.931. Allowable values:

```
LCM_PRIM_OK           /* control request is accepted */
LCM_PRIM_NOK          /* control request is not accepted */
LCM_PRIM_OK_NDONE     /* control request is accepted but the
                         activity is not complete */
```

Note: **LCM_PRIM_OK_NDONE** *status is returned only in response to warmstart or synchronization control requests. After this, the stack manager should wait for the final response:* **LCM_PRIM_OK** *or* **LCM_PRIM_NOK.**

reason

Reason of the failure. Allowable values:

```
LCM_REASON_INVALID_SUBACTION  /* Invalid subaction */
LCM_REASON_INVALID_ACTION     /* Invalid action */
LCM_REASON_PEER_SAP_NOT_CFG;  /* Peer SAP is not configured */
LCM_REASON_INVALID_STATE      /* The PSF state is invalid */
LCM_REASON_INVALID_ELMNT      /* Element in the control request
                                 is invalid
```

Description:

PSF - Q.930 / Q.931 uses this primitive to acknowledge the control request sent by the system manager.

Returns:

ROK OK

RFAILED Failed

3.4 Peer Layer Interface

The interface with the peer PSF - Q.930 / Q.931 is used to transfer updated information from the active node to the standby. This interface is always loosely coupled.

The primitives between the PSF - Q.930 / Q.931s are characterized use the prefix **ZqPiOub** (PSF - Q.930 / Q.931 peer interface outbound) or **ZqPiInb** (PSF - Q.930 / Q.931 peer interface inbound). The following primitives are used for data transfer:

Name	Description
ZqPiOubDatReq	Outbound data request
ZqPiInbDatReq	Inbound data request
ZqPiOubDatCfm	Outbound data confirm
ZiPiInbDatCfm	Inbound data confirm

These primitives are described in detail in the following sections.

3.4.1 ZqPiOubDatReq

Name:

Outbound Data Request

Direction:

Active PSF to standby PSF

Supplied:

Yes

Synopsis:

```
PUBLIC S16 ZqPiOubDatReq(pst, mBuf)
Pst *pst;
Buffer *mBuf;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

mBuf

Pointer to the PSF - Q.930/Q.931 update message. Each update message contains the state information of one or more control blocks. The PSF - Q.930/Q.931 on the active node copies the state information into this message as part of warmstart, controlled switchover, or run-time state update procedures.

Description:

This primitive is used by the PSF - Q.930/Q.931 on the active node to transfer state information to the standby. The active PSF copies the state information for various control blocks (for example, upper SAP control block or lower SAP control block) in one update message. The active PSF can copy a fixed maximum number of bytes into the message; therefore, it can send multiple update messages to send the complete state information. PSF - Q.930/Q.931 posts this message to system services (outbound). System services then routes the message to the PSF - Q.930/Q.931 on the standby node.

For more information, see the following sections:

- Section 4.4.1, "Run-time State Update"
- Section 4.4.2, "Warmstart State Update"
- Section 4.4.3, "Controlled Switchover State Update"

Returns:

00 ROK

3.4.2 ZqPiInbDatReq

Name:

Inbound Data Request

Direction:

Active PSF to standby PSF

Supplied:

Yes

Synopsis:

```
PUBLIC S16 ZqPiInbDatReq(pst, mBuf)
Pst *pst;
Buffer *mBuf;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

mBuf

Pointer to the PSF - Q.930/Q.931 update message. Each update message contains the state information of one or more control blocks. The PSF - Q.930/Q.931 on the active node copies the state information into this message as part of warmstart, controlled switchover, or run-time state update procedures.

Description:

Using this primitive, the standby PSF - Q.930/Q.931 receives the updated state information of the Q.930/Q.931 control blocks from the active PSF - Q.930/Q.931. The direction of this primitive is from the operating system to the standby PSF (inbound). For more information, see the following sections:

- Section 4.4.1, "Run-time State Update"
- Section 4.4.2, "Warmstart State Update"
- Section 4.4.3, "Controlled Switchover State Update"

Returns:

00 ROK

3.4.3 ZqPiOubDatCfm

Name:

Outbound Data Confirm

Direction:

Standby PSF to active PSF

Supplied:

Yes

Synopsis:

```
PUBLIC S16 ZqPiOubDatCfm(pst, status)
Pst *pst;
U8 status;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

status

Status. Allowable values:

CMPFTHA_OK
CMPFTHA_NOK

Description:

This primitive is used by the standby PSF - Q.930/Q.931 to acknowledge the receipt of all the update messages (data requests) sent by the active PSF - Q.930/Q.931 as part of warmstart or controlled switchover. The standby PSF sends only one data confirm in response to all the data requests received during warmstart or controlled switchover. The standby PSF - Q.930/Q.931 does not send any confirmation for run-time updates.

The active PSF - Q.930/Q.931 places a sequence number in all the data requests. If the standby PSF - Q.930/Q.931 finds a data request that has an out-of-order sequence number, it immediately sends a data confirm with a **CMPFTHA_NOK** status to the active PSF - Q.930/Q.931. When the standby PSF - Q.930/Q.931 detects the last data request, it sends data confirm with a **CMPFTHA_OK** status to the active PSF - Q.930/Q.931. The standby PSF - Q.930/Q.931 posts this message to outbound system services. System services then routes the message to PSF - Q.930/Q.931 on the active processor.

Returns:

00 ROK

3.4.4 ZqPiInbDatCfm

Name:

Inbound Data Confirm

Direction:

Standby PSF to active PSF

Supplied:

Yes

Synopsis:

```
PUBLIC S16 ZiPiInbDatCfm(pst, status)
Pst *pst;
U8 status;
```

Parameters:

pst

See Section 3.1.2.1, "Post."

status

Status. Allowable values:

CMPFTHA_OK
CMPFTHA_NOK

Description:

This primitive is used by the standby PSF - Q.930/Q.931 to send a confirmation to the active PSF - Q.930/Q.931 for the data requests received during the warmstart or controlled switchover procedure. The direction of this primitive is from the operating system to the active PSF - Q.930/Q.931 (inbound).

For run-time state update messages, the standby PSF - Q.930/Q.931 does not send a confirmation. For more information, see the following sections:

- Section 4.4.2, "Warmstart State Update"
- Section 4.4.3, "Controlled Switchover State Update"

Returns:

00 ROK

4 INTERFACE PROCEDURES

This section describes the interface and protocol procedures defined for the PSF - Q.930/Q.931 software. The interface procedures define the mechanisms by which the software interacts, via primitives, with any adjacent software within the system in which it resides.

The procedures in this section explain only the PSF - Q.930/Q.931 behavior in various fault-tolerant scenarios. Refer to the *FT/HA Core Service Definition* for details about these scenarios and for the time sequencing of primitives among entities other than PSF - Q.930/Q.931.

The procedures differ depending on whether the interface is tightly coupled or loosely coupled. The system services interface is *always* tightly coupled.

A *tightly coupled interface* implies that the interface between two protocol layers consists of direct function calls between the two layers. A *loosely coupled interface* implies that the interface between two protocol layers consists of passing messages between the two layers via queues maintained by system services.

4.1 Message Format & Flow

If a tightly coupled interface is used, the primitives referenced within the flow diagrams are directly called.

The flow and message format for the steps applicable to the tightly coupled interface are:

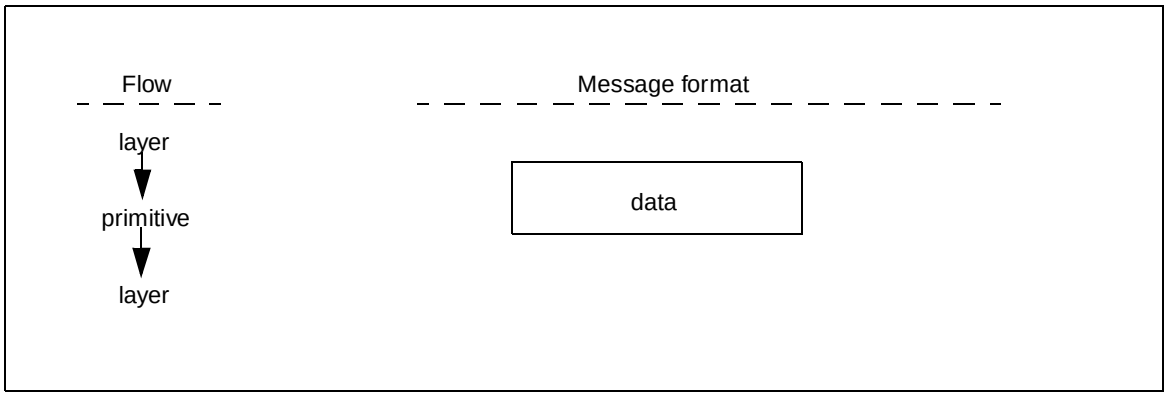


Figure 4-1: Message format and flow in tightly coupled interfaces

The data is the message buffer (if applicable) passed in the primitive.

If a loosely coupled interface is used, a set of packing (**SPkxxx**) and unpacking (**SUnpkxxx**) functions sit between the primitive and the associated message to / from system services.

Figure 4-2 illustrates the flow and message format for the steps applicable to the loosely coupled interface.

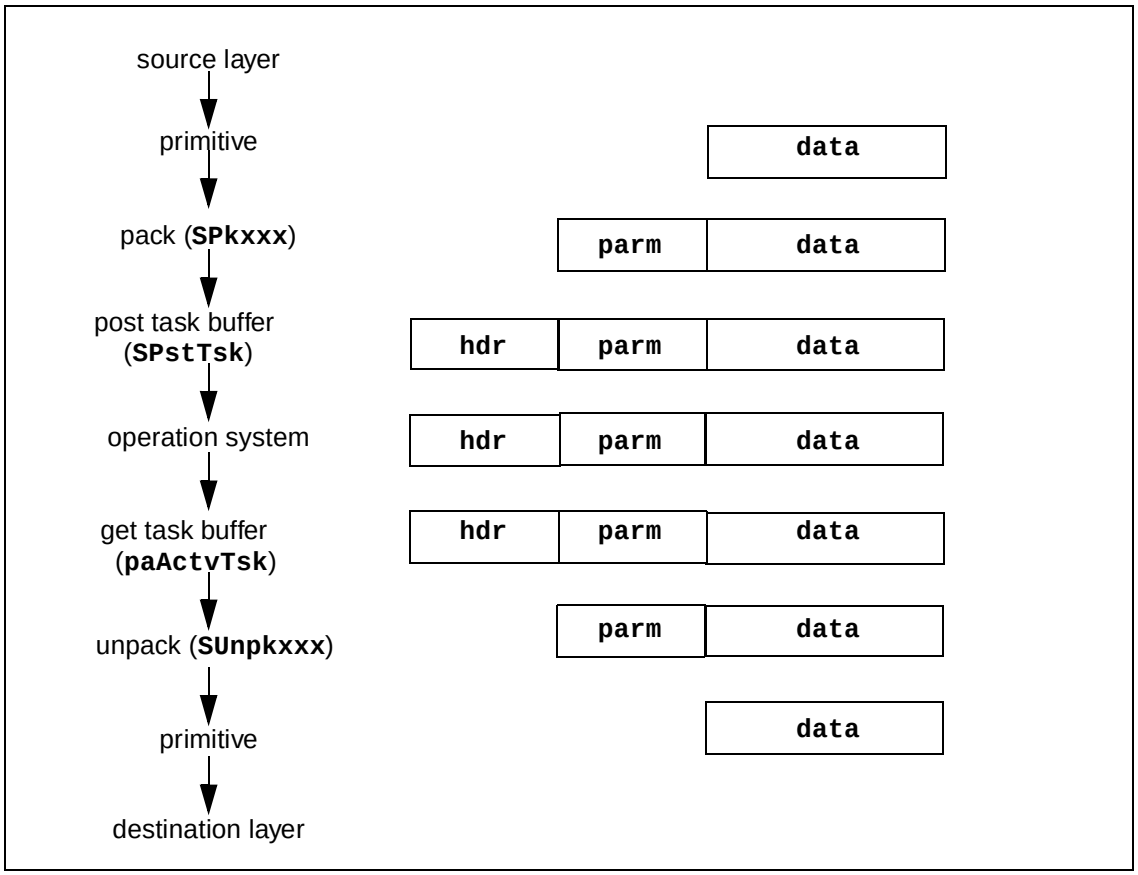


Figure 4-2: Message format and flow in loosely coupled interfaces

The header (**hdr**) is completely independent of the protocol layer and represents any information that must be placed in the message by system services (priority, routing, destination entity, source entity, and so on) to ensure proper routing of the message to the destination entity.

The parameters (**parm**) are an encoded version of all of the parameters passed in the primitive.

The data (**data**) is the message buffer (if applicable) passed in the primitive.

A PSF primitive that causes a message to flow to another layer calls the appropriate packing function and then the post task (**SPstTsk**) system services primitive to send the message to the layer. A message received from a layer causes system services to call the activate task primitive, which calls the appropriate unpacking function and then the PSF primitive.

For clarity, the packing, unpacking, **SPstTsk**, and **zqActvTsk** primitives are not included in the flow diagrams for a loosely coupled interface.

The following rules apply to the flow diagrams in this section:

- Time flows towards the bottom of the page.
- The mnemonic above a line represents a function call or primitive.
- The mnemonic below a line represents a message type.
- A **+** indicates an OR condition, that is, one path or another may be taken.
- A **o** indicates an AND condition, that is, both paths are taken in parallel.

The labels above each flow diagram have the following meaning:

Name	Description
SS	System Services
IN	Q.930 / Q.931
PSF	Protocol Specific Function
Peer PSF	Peer Protocol Specific Function
LM	Layer Manager

The following interface procedures are described:

Name	Description
System services interface	Procedures related to initializing the software, such as timer activation
Layer manager interface	Procedures related to the control and monitoring of the PSF - Q.930 / Q.931 software, such as configuration, solicited status, unsolicited status, and control
Peer layer interface	Procedures related to run-time, warmstart, and controlled switchover state update

4.2 System Services Interface

PSF - Q.930 / Q.931 is not registered as a separate task with system services. It has the same entity ID as Q.930 / Q.931. Q.930 / Q.931 calls both the activation initialization function and the task activation function of the PSF - Q.930 / Q.931. The system services interface has several components, including:

- Task initialization
- Task activation
- Timer activation
- Error checking and recoveries

4.2.1 Task Initialization

The initialization procedure initializes the Q.930 / Q.931 software. This procedure is initiated by the layer manager when it registers the initialization function for Q.930 / Q.931, using the **SRegInit** primitive. System services then calls the initialization function, **inActvInit**. This function, in turn, invokes the initialization activation function in PSF - Q.930 / Q.931 to initialize the global variables and structures of PSF.

The data flow is:

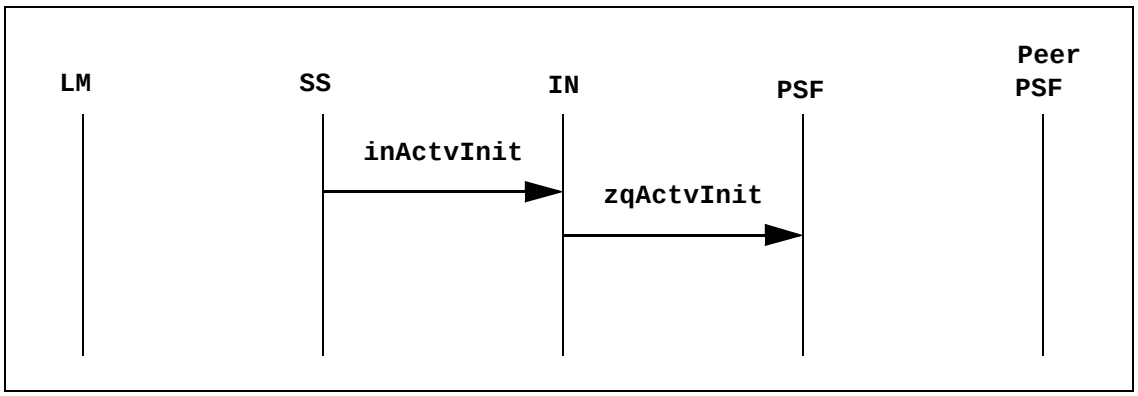


Figure 4-3: Data flow—initialization procedure

4.2.2 Task Activation

The task activation procedure is used to provide activation to the Q.930 / Q.931 software whenever there is a message waiting for from a loosely coupled interface. If the message received is not for the Q.930 / Q.931 layer, the layer invokes the task activation function of PSF - Q.930 / Q.931.

The data flow is:

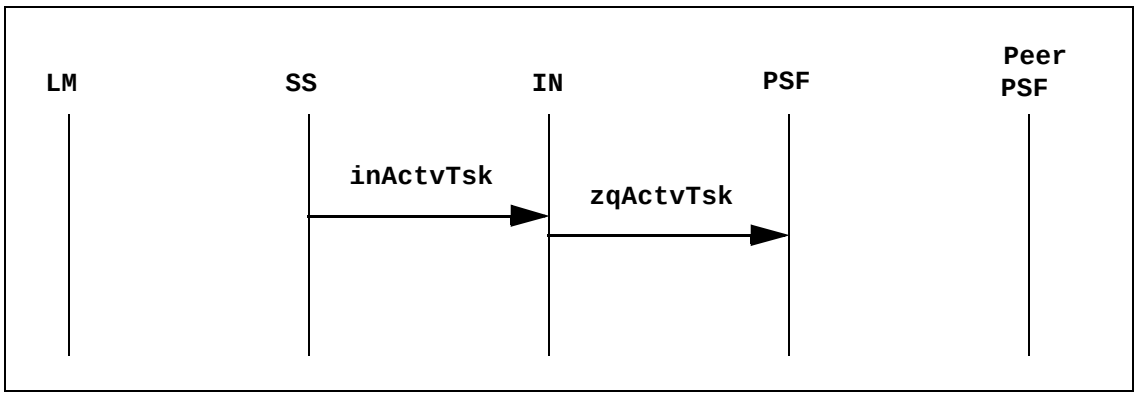


Figure 4-4: Data flow—task activation procedure

4.2.3 Timer Activation

The timer activation procedure is used to provide periodic activation to the PSF - Q.930 / Q.931 software so that it can manage its own timers. This procedure is initiated by PSF - Q.930 / Q.931 when it registers the timer activation function using the **SRegTmr** primitive. This primitive also specifies the time period between successive timer activations. System services then calls the timer activation functions periodically. Typically, **SRegTmr** is called during general configuration, once PSF - Q.930 / Q.931 has obtained the required time resolution from the layer manager.

The data flow is:

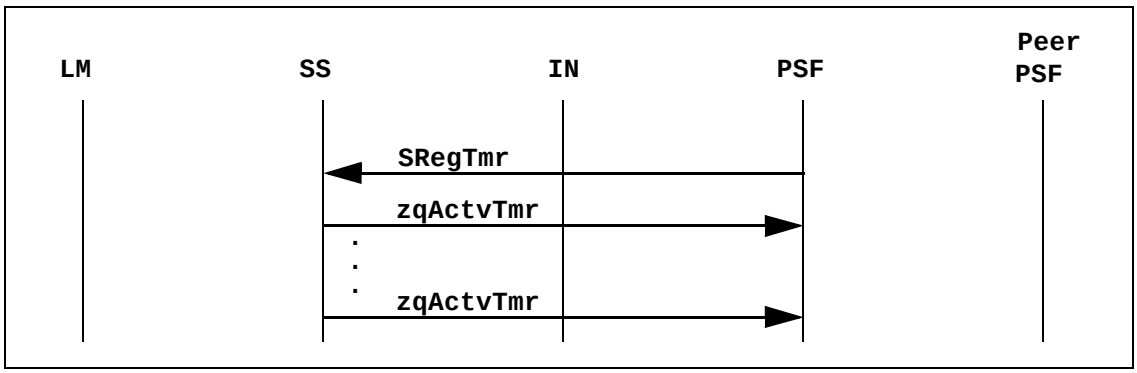


Figure 4-5: Data flow—timer activation procedure

4.2.4 Error Checking and Recoveries

Intensive error checking and recovery mechanisms are provided in the software to make it robust enough to deal with normal error conditions. Whenever any error is detected in the protocol layer, recovery actions are taken. The error recoveries depend upon the type of error. For example, if there is any error at the upper layer interface or at the layer manager interface, an error is logged to the system services.

If the PSF is not able to allocate the resources, it is returned to the stable state and the stack manager is informed. In addition to informing the layer manager, system services is informed via the **SLogError** function call. To make the systems robust and time efficient, the errors are checked under compile-time flags. These compile-time flags can be enabled / disabled on a file-by-file basis.

The error checks are classified into the following categories:

Get Resources

Get resources errors are defined to mean system service primitives where memory / resources must be allocated in order for the primitive to succeed. These system service calls are always validated unconditionally. When return values are in error, remedial actions are taken so that the software remains in the stable state.

Examples of such primitives are: **SGetSBuf**, **SGetDBuf**, **SGetMsg**, **SCpyMsgMsg**, **SRegTmr**, **SRegTsk**, **SRegActvTsk**, **SRegDrvrTsk**, and so on.

Add Resources

Add resource errors are defined to mean system services primitives where memory / resources may be allocated in order for the primitive to succeed (depending upon the implementation). These system service call returns are validated under the **ERRCLS_ADD_RES** conditional compile flag. When return values are in error, remedial actions are taken so that the protocol layer goes back into the stable state.

Examples of such routines are: **SAddPreMsg**, **SAddPstMsg**, **SAddPreMsgMult**, **SAddPstMsgMult**, **SPkU8**, **SPkU16**, **SPkU32**, **SPkS8**, **SPkS16**, **SPkS32**, **SCpyFixMsg**, **SPstTsk**, **SQueueLast**, **SqueueFirst**, and so on.

Interface Parameters

These errors are related to invalid parameters that are passed into primitive functions, including configuration. The validation of these attributes are handled under the **ERRCLS_INT_PAR** conditional compile flag. All the interface errors are reported to the layer manager and to the system services.

Consistency Checks

These errors are related to the failure of routines that happen because of improper usage. This type of error should not occur in an internally consistent software module. No recovery action is taken on the occurrence of such errors. The validation of these routines are done under the **ERRCLS_DEBUG** conditional compile flag. The examples of such routines are: **SFndLenMsg**, **SCatMsg**, **SPrntMsg**, **SFndLenQueue**, **SUnpkU8**, **SUnpkU16**, **SInitMsg**, **SPutSBuf**.

4.3 Layer Manager Interface

This section describes the components of the layer manager interface. The layer manager interface allows the management entity to execute various fault-tolerant procedures on Q.930/Q.931, such as warmstart and controlled switchover.

4.3.1 Management – Configuration

The layer manager initiates the management - configuration procedure to configure the various elements of the PSF - Q.930/Q.931 software. The PSF - Q.930/Q.931 configuration request primitive (**`ZqMiLsiCfgReq`**) can be called more than once, **before** Q.930/Q.931 binds with its provider and its user.

The following PSF - Q.930/Q.931 configuration request primitive types may be called:

Name	Description
General	Passes parameters that apply to the PSF - Q.930/Q.931 software as a whole. Used primarily to tune the PSF software for the most efficient use of its resources. This configuration is performed only once.
Peer SAP	Passes parameters that apply to the peer SAP. Used primarily to configure the post structure towards the peer PSF - Q.930/Q.931. It can be called more than once for reconfiguration at run time.

The configuration request primitive type is specified by the **`cmPFthaMngmt.hdr.elmId`** field.

For proper operation, the configuration request primitive types must be called in the following order:

1. General
2. Peer SAP

System services primitives are called during the management - configuration procedure.

The register timer (**`SRegTmr`**) system services primitive is called during the general configuration request primitive to register the PSF - Q.930/Q.931 timer activation (**`zqActvTmr`**) function.

The **`cmPFthaMngmt.t.cfg`** structure is used to specify parameters used by the configuration request primitive. The PSF responds with a configuration confirm to the layer manager after processing the configuration request.

The data flow is:

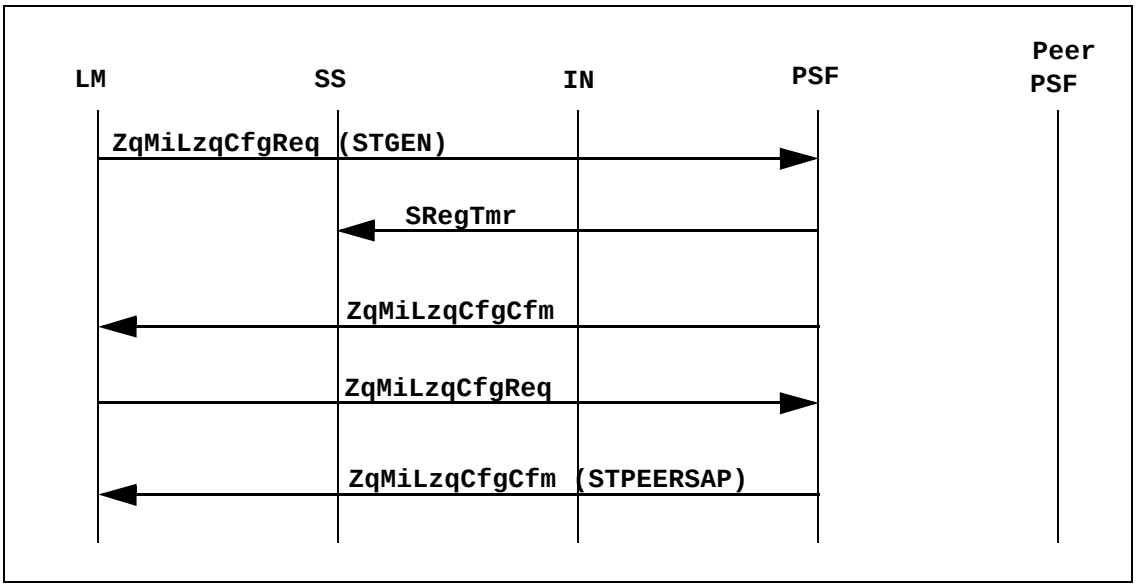


Figure 4-6: Data flow—management - configuration procedure

Note: *The PSF - Q.930/Q.931 must be configured only after the Q.930/Q.931 general configuration.*

4.3.1.1 Retry of Configuration Request

The layer manager can resend the same configuration request if it does not receive a configuration confirm from the PSF - Q.930/Q.931. The configuration request or confirm can get lost due to unreliable communication media between the layer manager and PSF - Q.930/Q.931. If PSF receives a configuration request that is already in a configured state, it handles the request as reconfiguration request (that is, it configures only those parameters that are reconfigurable at run time). Therefore, repeating the same configuration request more than once does not affect the PSF operation.

4.3.2 Management – Solicited Status

The layer manager initiates the management - solicited status procedure to gather solicited status information about the various elements of the PSF - Q.930 / Q.931 software. The status request primitive (**ZqMiLzqStaReq**) can be called more than once, any time after the management - configuration procedure.

The following PSF status request primitive types can be called:

1. General
2. System ID
3. Peer SAP

The status request primitive type is specified by the **cmPFthaMngmt.hdr.elmId** field. The **cmPFthaMngmt.t.sta** structure is used to specify parameters used by the status request and status confirm primitives.

The status confirm (**ZqMiLsiStaCfm**) primitive and other system services primitives are called during the status procedure if a loosely coupled layer manager interface is used. The status confirm (**ZqMiLsiStaCfm**) primitive is not called during the status procedure if a tightly coupled layer manager interface is used.

The data flow is:

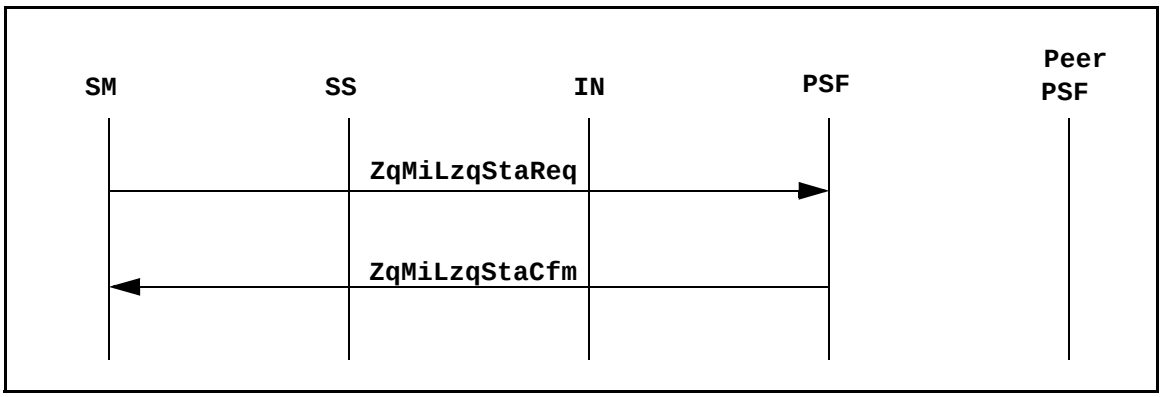


Figure 4-7: Data flow—management - solicited status procedure

4.3.2.1 Retry of Status Request

The layer manager may resend the same status request if it does not receive a status confirm from the PSF - Q.930 / Q.931 in response. The status request or confirm can get lost due to unreliable communication media between the layer manager and PSF - Q.930 / Q.931. The PSF handles the status request in the same manner each time, responding with a status confirm.

4.3.3 Management – Unsolicited Status

The management–unsolicited status (alarm) procedure indicates the occurrence of an abnormal condition in PSF - Q.930 / Q.931. This procedure is initiated by the PSF - Q.930 / Q.931 software. The unsolicited status indication primitive (**ZqMiLsiStaInd**) can be called at any time after the configuration procedure, if the unsolicited status has been enabled. The status indication primitive is not called if the unsolicited status has been disabled. The unsolicited status can be enabled or disabled with the management - control procedure.

System services primitives are called during the unsolicited status procedure if a loosely coupled layer manager interface is used.

The **cmPFthaMngmt.t.usta** structure is used to specify parameters used by the status indication primitive.

The data flow is:

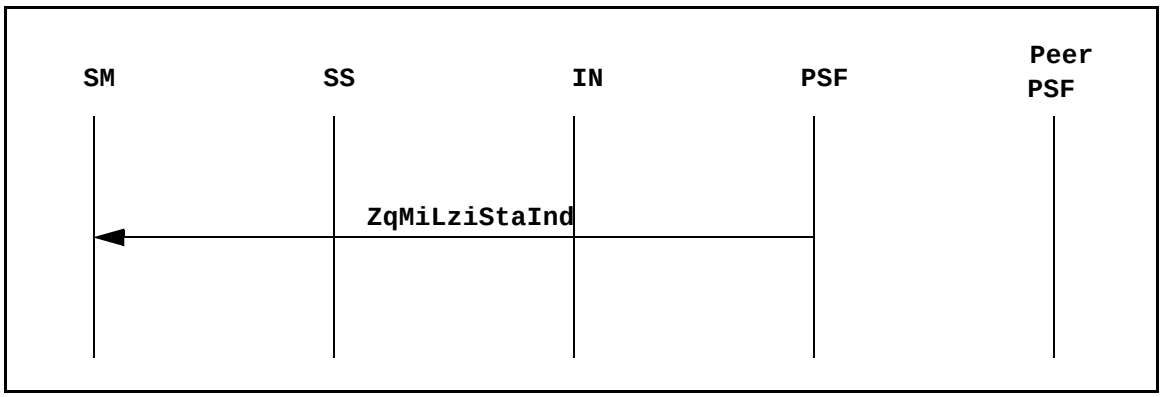


Figure 4-8: Data flow—management - unsolicited status procedure

4.3.4 Management–Control

The management–control procedure is used by the layer manager to control various elements of the PSF - Q.930 / Q.931 software. This procedure is initiated by the layer manager. The PSF control request primitive (**ZqMiLzqCntrlReq**) can be called more than once, any time after the management–configuration procedure.

In response to every control request, the PSF - Q.930 / Q.931 sends a control confirm (**ZqMiLzqCntrlCfm**), indicating success or failure of the control request in the status field of the **cfm** structure.

The data flow is:

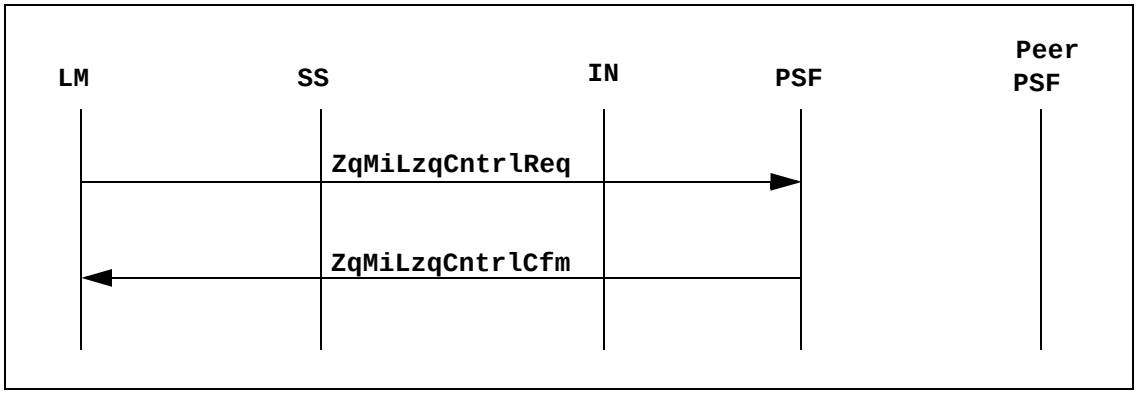


Figure 4-9: Data flow—management - control procedure

The following control actions can be performed with the control request:

1. Go active
2. Go standby
3. Disable peer SAP
4. Shutdown
5. Disable / enable alarms
6. Disable / enable debug prints
7. Warmstart
8. Synchronize
9. Abort

The following sections describe each of these control requests in detail.

4.3.4.1 Go Active

This action is used by the layer manager to make a Q.930/Q.931 layer active. The following combinations of **action** and **subaction** fields in the **cntrl** (**cmPFthaMngmt.t.cntrl.action**) structure are possible:

action	subAction	Scenario
AGO_ACT	ADIS_PEER_SAP	If the peer SAP is OOS. For example, in a forced switchover scenario when an OOS peer needs to be made active.
AGO_ACT	ADIS_ENA_PEER_SAP	If the peer is standby. For example, in a controlled switchover when the peer is already in standby and the manager wants to make it active.

The layer manager can send this control request to PSF - Q.930/Q.931 under the following conditions:

1. When the status of Q.930/Q.931 is OOS:
   - During system initialization, after the configuration of the Q.930/Q.931 layer and the PSF - Q.930/Q.931, the layer manager requests that PSF make the Q.930/Q.931 layer active. The Q.930/Q.931 is originally in the OOS state. It can start its operations only after it becomes active.
2. When the status of Q.930/Q.931 is standby:
   - During controlled switchover the layer manager requests the active to update the standby PSF with the current states. The manager then sends a Go Active control request to the standby PSF to make the Q.930/Q.931 layer active. The new active layer then starts handles the protocol events from the upper or lower layer.
   - During a forced switchover, the layer manager sends a Go Active control request to the standby when the current active Q.930/Q.931 layer becomes OOS. To keep the loss minimal, the layer manager holds the traffic coming to PSF - Q.930/Q.931 from the upper and lower layers before making the standby active.
3. When the status of Q.930/Q.931 is active
   - The layer manager can abort a controlled switchover procedure before it has been completed (even if the active PSF has updated the standby with the transient state information). Before the current active Q.930/Q.931 resumes operation, the layer manager must send a control request to the active to become active again.

4.3.4.1.1 Retry of Go Active Control Request

If the layer manager does not receive a control confirm from the PSF for the Go Active control request, the layer manager can retry the request. If the PSF - Q.930/Q.931 has already received the control request, it does not take any action but sends a control confirm indicating success. Therefore, this control request can be retried any number of times in case of lost primitives.

4.3.4.2 Go Standby

This action is used by the layer manager to make the Q.930/Q.931 standby. The **action** and the **subAction** fields in the **cntrl** (**cmPFthaMngmt.t.cntrl**) structure are as follows.

action	subAction	Scenario
AGO_SBY	SAENA_PEER_SAP	The PSF must always exist to be made standby. The PSF can be OOS or active.

The layer manager can send this control request to PSF - Q.930/Q.931 under the following conditions:

1. When the status of Q.930/Q.931 is OOS:
   - After configuring Q.930/Q.931 and PSF - Q.930/Q.931, the layer manager sends a Go Standby control request to make the PSF standby from OOS state.
2. When the status of Q.930/Q.931 is active:
   - During controlled switchover, the layer manager can request the active PSF to update the standby PSF with the current transient states of Q.930/Q.931. Subsequently, the layer manager sends a Go Standby control request to the active PSF to make the Q.930/Q.931 standby. The new standby Q.930/Q.931 starts receiving run-time state updates from the active PSF.
3. When the status of Q.930/Q.931 is standby:
   - The layer manager may can abort a controlled switchover procedure at any time even when the procedure is not completed. At the time the procedure is aborted, it may not have swapped the status of active and standby even though active PSF - Q.930/Q.931 has updated the standby with the transient state information. Before the current active Q.930/Q.931 resumes operation, the layer manager has to send a request to the standby to remain standby.

For data flow, see Figure 4-9.

4.3.4.2.1 Retry of Go Standby Control Request

If the layer manager does not receive a control confirm for the Go Standby control request, it can retry the request. If the PSF - Q.930/Q.931 has already received the control request, it does not take any action but it sends a control confirm indicating success. Therefore, this control request can be retried any number of times in case of lost primitives.

4.3.4.3 Disable Peer SAP

The layer manager sends this control request to the active PSF - Q.930 / Q.931 when the standby PSF - Q.930 / Q.931 goes OOS. The active PSF then stops sending run-time state update messages to the peer. The **action** and the **subAction** fields in the **cntrl** (**cmPFthaMngmt.t.cntrl**) structure are as follows:

action	subAction
AUBND_DIS	UNUSED

For data flow, see Figure 4-9.

4.3.4.3.1 Retry of Disable Peer SAP Control Request

If the layer manager does not receive the control confirm for the control request to disable the peer SAP, it can retry the request. The PSF - Q.930 / Q.931 takes the same action again without affecting PSF states and sends a control confirm indicating success. This control request can, therefore, be retried any number of times in case of lost primitives.

4.3.4.4 Shutdown

The layer manager sends this control request to the PSF - Q.930 / Q.931 to shut down its operation. The PSF deletes all its dynamic resources, as well as any timers that are running. It can be fully reconfigured after the shutdown. The **action** and the **subAction** fields in the **cntrl** (**cmPFthaMngmt.t.cntrl**) structure are as follows:

action	subAction
ASHUTDOWN	UNUSED

For data flow, see Figure 4-9.

4.3.4.4.1 Retry of Shutdown Control Request

If the layer manager does not receive the control confirm for the shutdown control request, it can retry the request. If the PSF - Q.930 / Q.931 has already received the control request, it takes no action and sends a control confirm indicating success.

4.3.4.5 Disable/Enable Alarms

The layer manager can use this control request to disable or enable alarm generation from the PSF - Q.930 / Q.931. By default, the alarms are disabled. To enable the alarm generation, the **action** and the **subAction** fields in the **cntrl** structure are as follows:

action	subAction
AENA	SAUSTA

To disable the alarm generation, the **action** and **subAction** fields in the **cntrl** structure are as follows:

action	subAction
ADISIMM	SAUSTA

For data flow, see Figure 4-9.

4.3.4.5.1 Retry of Disable/Enable Alarms Control Request

If the layer manager does not receive the control confirm for the control request to disable / enable alarms in PSF - Q.930 / Q.931, the layer manager can retry the request. PSF - Q.930 / Q.931 takes the same action again, without affecting PSF states, and sends a control confirm indicating success. This control request can, therefore, be retried any number of times in case of lost primitives.

4.3.4.6 Disable/Enable Debug Prints

The layer manager uses this control request to disable or enable debug prints for various debug levels from PSF - Q.930 / Q.931. By default, the debug prints are disabled. To enable debug prints, the **action** and **subAction** fields in the **cntrl** structure are as follows:

Action	subAction
AENA	SADBG

To disable the debug prints, the **action** and **subAction** fields in the **cntrl** structure are as follows:

Action	subAction
ADISIMM	SAUSTA

4.3.4.6.1 Retry of Disable/Enable Debug Prints Control Request

If the layer manager does not receive the control confirm for the control request to disable / enable debug prints in PSF, it can retry the request. The PSF - Q.930 / Q.931 takes the same action again, without affecting PSF states, and sends a control confirmation indicating success. This control request can, therefore, be retried any number of times in the case of lost primitives.

4.3.4.7 Warmstart

The layer manager initiates the warmstart procedure when an OOS node must be made standby. The layer manager follows these steps to warmstart an OOS node.

1. Configures the standby Q.930/Q.931 and the corresponding PSF - Q.930/Q.931
2. Sends a control request to PSF - Q.930/Q.931 to make the new Q.930/Q.931 standby
3. Sends a control request to the active PSF to warmstart the standby with the **action** and **subAction** fields in the **cntrl** structure as follows:

action	subAction
AWARMSTART	NOTUSED

When the active PSF receives a warmstart control request, it sends a control confirm to the layer manager indicating that the warmstart has started.

If the active PSF - Q.930/Q.931 successfully sends the state update messages to the peer, it sends a second control confirmation primitive to the layer manager with the **status** in the **cfm** structure as **LCM_PRIM_OK**, indicating that warmstart was successful.

If the active PSF - Q.930/Q.931 encounters a failure during the state update procedure (for example, a state update message gets lost because of unreliable communication media between the active and the standby node) it sends a second control confirmation primitive to the layer manager with the **status** in the **cfm** structure as **LCM_PRIM_NOK**, indicating that warmstart failed. The active PSF also disables the peer SAP so that the run-time state update to the peer is stopped.

The data flow for warmstart is illustrated in Figure 4-10.

The data flow is:

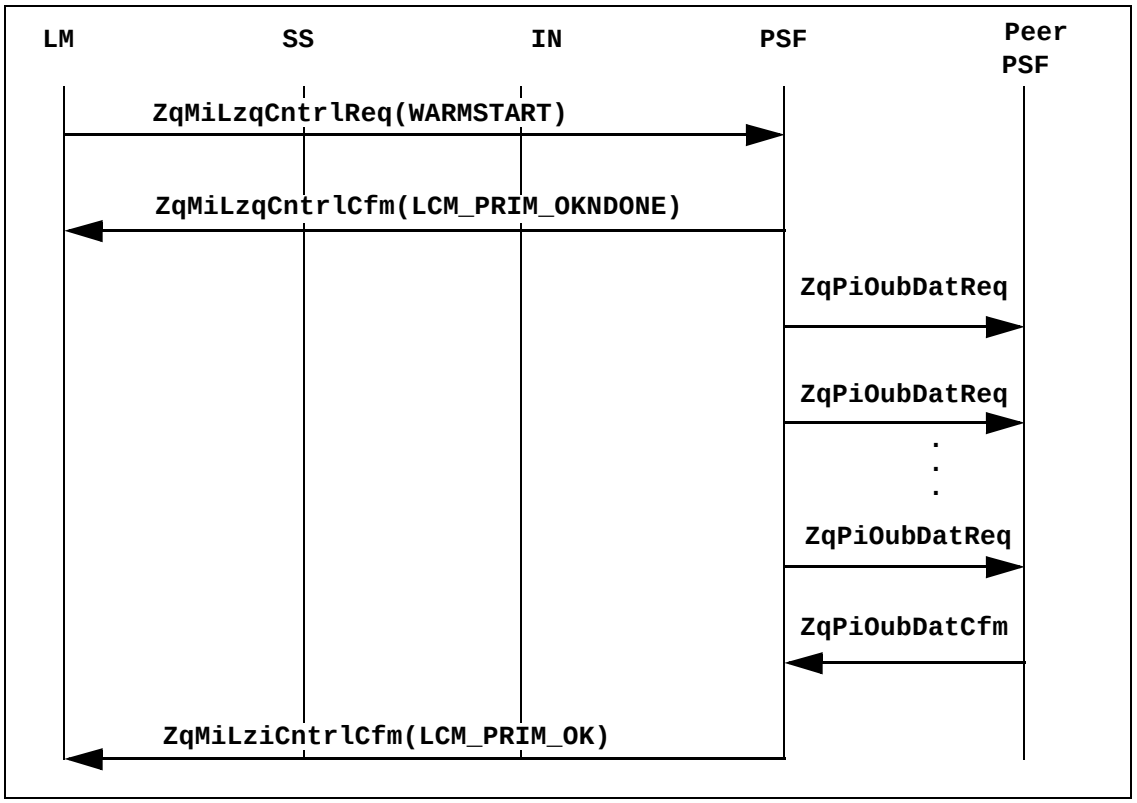


Figure 4-10: Data flow—warmstart procedure

4.3.4.7.1 Retry of Warmstart Control Request

If the warmstart control request or control confirm is lost, the layer manager can resend the control request. The layer manager can lose either the first or second control confirm, resulting in the layer manager resending the control request. If the layer manager retries, PSF can receive the control request in any of the following states:

- The PSF - Q.930/Q.931 has completed the warmstart successfully (which is the case when the second control confirm gets lost) and, thus, the peer SAP is enabled. In this case, the PSF returns a control confirmation indicating that the warmstart is complete.
- The warmstart is in progress (which is the case when the first control confirm gets lost). In this case, the PSF sends a control confirm indicating that the warmstart is continuing.
- The peer SAP is disabled—that is, either the warmstart never happened (which is the case when the control request gets lost) or the warmstart failed (which is the case when the second control confirm gets lost, indicating failure). In this case, the PSF starts the warmstart procedure and sends a control confirm indicating warmstart is continuing. After completing the warmstart, the PSF sends a second control confirm indicating that the warmstart is complete.

4.3.4.8 Synchronize

The layer manager initiates the synchronize procedure during the controlled switchover of a Q.930 / Q.931 node. Before swapping the status of the active and standby nodes, the layer manager sends the synchronize control request to the active PSF to update the standby with the current transient state information. The layer manager sends this control request to the active only after freezing the traffic coming to Q.930 / Q.931 layer, so that Q.930 / Q.931 states do not change while the active is updating the standby.

The layer manager sends the synchronize control request to the active PSF with the **action** and the **subAction** fields in the **cntrl** structure as follows:

action	subAction
ASYNCHRONIZE	NOTUSED

On receiving the control request the active PSF immediately sends a control confirmation to the layer manager with **status** in the **cfm** structure as **LCM_PRIM_OKNDONE**, indicating that synchronization is progressing.

The active PSF - Q.930 / Q.931 freezes its timers so that timer firing does not result in any state changes. The active then updates the peer with all its transient state update information.

If the active PSF - Q.930 / Q.931 successfully sends the state update messages to the peer, it sends a second control confirmation to the layer manager with **status** in the **cfm** structure as **LCM_PRIM_OK**, indicating that the synchronize was successful.

If the synchronize procedure failed for any reason (for example, a state update message becomes lost due to unreliable communication media between the active and the standby node), the active PSF sends a control confirmation primitive to the layer manager with **status** in the **cfm** structure as **LCM_PRIM_NOK**, indicating that synchronization has failed. In this case, the active does not disable the peer SAP so that the active node can continue to send run-time state updates to the standby node. Before restarting the traffic to Q.930 / Q.931 layer, the layer manager must send a control request to the active PSF to remain active and to the standby PSF to remain standby.

The data flow is:

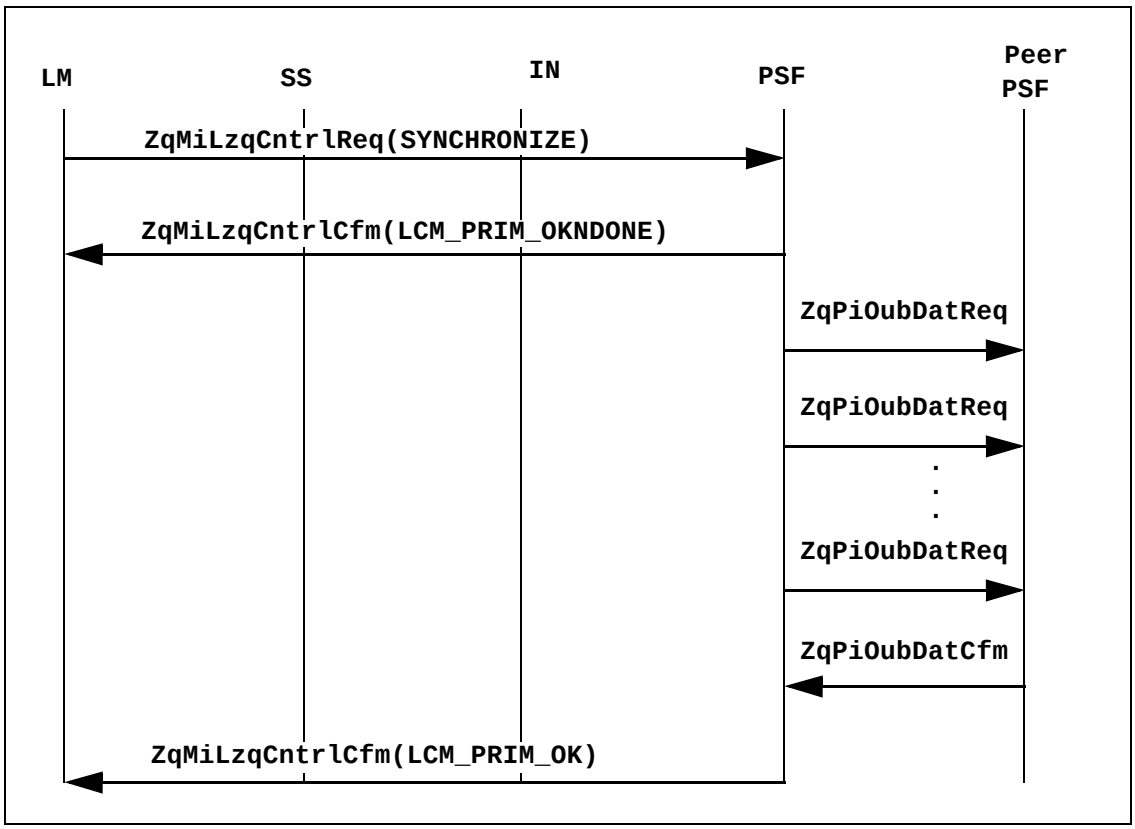


Figure 4-11: Data flow—synchronization procedure

4.3.4.8.1 Retry of Synchronize Control Request

If the synchronize control request or control confirm is lost, the layer manager can resend the control request. The layer manager can lose either the first control confirm or the second, resulting in the resend. If the layer manager retries the send, PSF can receive the control request in one of the following states:

- The synchronization procedure is ongoing:

  This occurs when the first control confirm gets lost. In this case, the PSF - Q.930 / Q.931 sends a control confirm to the layer manager indicating that the synchronization is continuing.

- The PSF - Q.930 / Q.931 is not currently carrying out the synchronization:

  This occurs when either the synchronization never happened (which is the case when the control request itself gets lost) or the synchronization was completed (which is the case when the second control confirm gets lost). In this case, the PSF starts the synchronization procedure and sends a control confirmation indicating that synchronization is continuing. After completing the synchronization, PSF sends the second control confirm indicating that synchronization was completed.

4.3.4.9 Abort

The warmstart or controlled switchover state update can be a long procedure, depending on the Q.930/Q.931 data to be updated. While the warmstart or controlled switchover state update is continuing (that is, the layer manager has received the first control confirm indicating that state update is continuing), the layer manager can send this control request to the active PSF to abort the state update procedure. The active PSF - Q.930/Q.931 responds with a control confirm for the stop request.

If a warmstart state update is aborted, the active PSF disables the peer SAP. If the synchronize state update is aborted, the layer manager must send a control request to the active PSF to remain active and to the standby PSF to remain standby before releasing the traffic to Q.930/Q.931.

If the active PSF receives this request in idle state (that is, there is no ongoing warmstart state update or controlled switchover state update), the PSF sends a control confirm to the layer manager indicating success. The PSF does not perform any other action.

The layer manager sends the abort control request to the active PSF with the **`action`** and **`subAction`** fields in the **`cntrl`** structure as follows:

`action`	**`subAction`**
`AABORT`	**`NOTUSED`**

For data flow, see Figure 4-9.

4.3.4.9.1 Retry of Abort Control Request

If the layer manager does not receive the control confirm for the control request to abort warmstart or the controlled switchover, the layer manager can retry the request. The PSF - Q.930/Q.931 takes the same action again, without affecting PSF states, and sends a control confirm indicating success. This control request can be retried any number of times in case of primitive loss.

4.4 Peer Layer Interface

This interface consists of procedures to carry out state updates from the active PSF - Q.930 / Q.931 to the standby PSF - Q.930 / Q.931. The procedures related to this interface are as follows:

- Run-time state update
- Warmstart state update
- Controlled switchover state update

4.4.1 Run-time State Update

The active Q.930 / Q.931 can change its internal stable states while handling protocol events. When the active layer changes its internal states, the active PSF sends the modified state information to the standby (if the peer SAP is enabled) in the update messages. Each run-time state update message carries an incremented sequence number. If the standby PSF detects a sequence error in the run-time update messages, it sends an alarm to the layer manager (**ZqMiLinStaInd**)

The data flow is:

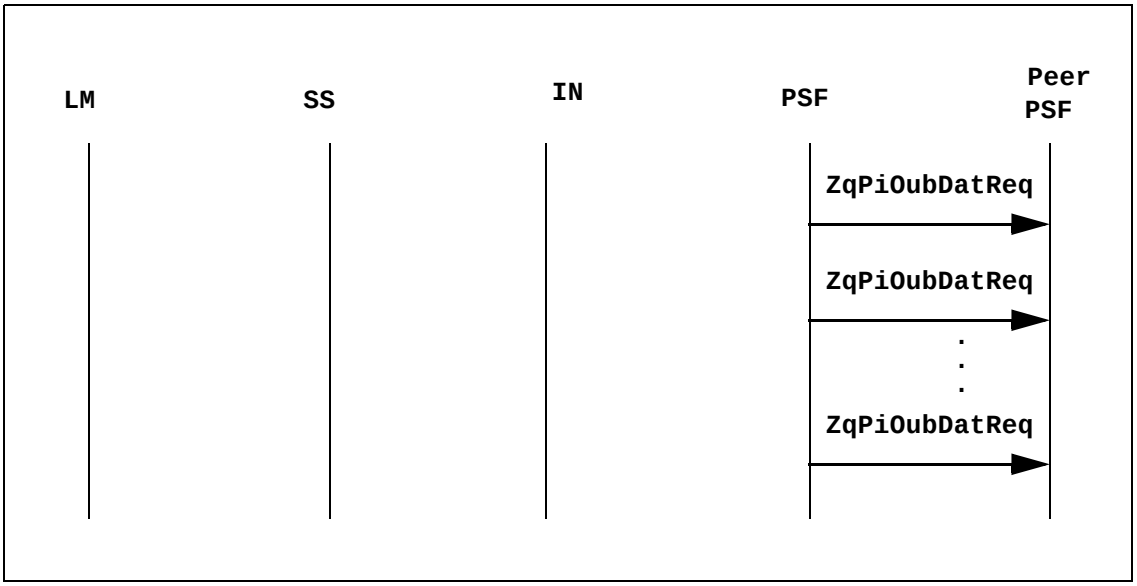


Figure 4-12: Data flow—run-time state update procedure

4.4.2 Warmstart State Update

When the layer manager decides to warm start the peer, it sends a warmstart control request to the active PSF - Q.930 / Q.931. The active PSF then sends the entire stable state information to the peer. The information can be sent as a single message or as multiple update messages. The active PSF sends all warmstart messages with an incremented sequence number. If the standby receives all the update messages, it sends a confirmation to the active indicating success. If the standby finds a sequence error in the update messages, it sends a confirmation to the active indicating failure.

For the data flow, see Figure 4-10.

4.4.3 Controlled Switchover State Update

When the layer manager decides to initiate a controlled switchover it sends a synchronize control request to the active PSF - Q.930 / Q.931. The active PSF then sends the entire transient state information to the peer. The active PSF can send multiple update messages to accommodate the complete transient state information. The active PSF sends all synchronization messages with an incremented sequence number. If the standby receives all the update messages, it sends a confirmation to the active indicating success. If the standby finds a sequence error in the update messages, it sends a confirmation to the active indicating failure.

For the data flow, see Figure 4-11.

5 PORTING

This section describes the process for porting the PSF - Q.930/Q.931 software.

PSF - Q.930/Q.931 is not an independent product. It compiles only with Trillium's Q.930/Q.931 product. Most of the general portation issues that affect PSF - Q.930/Q.931 are described in the *Q.930/Q.931 Portation Guide*. See the References section of this manual for more information.

The following table lists the PSF - Q.930/Q.931 source files:

Name	Description
zq_bdy1.c	Contains all the incoming primitives to PSF - Q.930/Q.931 from the layer manager, peer PSF - Q.930/Q.931, and protocol layer
zq_bdy2.c	Contains the support functions. This file also has functions that implement management control. This part of the code will be utilized by both the active and standby copies.
zq_bdy3.c	Contains all the packing functions invoked by the active PSF - Q.930/Q.931
zq_bdy4.c	Contains support functions to initialize the update structures/messages on the active side
zq_bdy5.c	Contains all the unpacking functions invoked by the standby PSF - Q.930/Q.931
zq_bdy6.c	Contains support functions to decode messages received on the standby side
zq_ptmi.c	Contains all the packing functions invoked by PSF - Q.930/Q.931 for sending primitives to the layer manager
zq_ptpi.c	Contains all the packing functions invoked by PSF - Q.930/Q.931 for sending primitives to the peer PSF - Q.930/Q.931
zq_ex_ms.c	Contains all the unpacking functions invoked by PSF - Q.930/Q.931 when it receives primitives from the layer manager or peer PSF - Q.930/Q.931
zq_id.c	Contains the current version of the PSF - Q.930/Q.931 software
zq.h	The header file for PSF - Q.930/Q.931 internal defines
zq.x	The header file for PSF - Q.930/Q.931internal structures and function prototypes
lzq.h	The header file containing defines for the events between PSF - Q.930/Q.931 and the layer manager
lzq.x	The header file containing the prototypes of the primitives between PSF - Q.930/Q.931 and the layer manager
zq_err.h	The header file for PSF - Q.930/Q.931 error defines
zq_acc.c	Contains the test engine for running PSF - Q.930/Q.931 acceptance tests
zq_acc.h	The header file for PSF - Q.930/Q.931 - acceptance test-specific defines
zq_acc.x	The header file for PSF - Q.930/Q.931 - acceptance test-specific **typedefs**

Name	Description
zq.mak	**Makefile** for building portable (PSF - Q.930/Q.931 and portable files) and acceptance tests executables for various environments
cm_pftha.h	Common header file for portable FT/HA defines
cm_pftha.x	Common header file for portable FT/HA structures and function prototypes
cm_pftha.c	Common portable FT/HA functions
smzqbdy1.c	PSF - Q.930/Q.931 stack manager - Body file
smzqptmi.c	PSF - Q.930/Q.931 stack manager - Portable manager interface
smzqexms.c	PSF - Q.930/Q.931 stack manager - External interface file
smzq_err.h	PSF - Q.930/Q.931 stack manager - Error code defines

In addition, PSF - Q.930/Q.931 uses both the common files and the stack manager files described in the *Q.930/Q.931 Portation Guide*.

5.1 Prepare Files for Make

PSF - Q.930 / Q.931 software can only be compiled with Trillium's Q.930 / Q.931 software (release 3.10 and any future releases). The following steps must be taken before the PSF - Q.930 / Q.931 software can be compiled:

1. Q.930 / Q.931 release 3.10 offers backward compatibility in the upper layer, lower layer, and layer manager interfaces. Backward compatibility can be achieved by commenting out the **ISDN_3_8_PLUS**, **INT2** (upper interface), **DAT2** (lower, LAPD interface), and **IN_LMINT3** (layer manager interface) compilation flags in the **#ifdef IN** section of the **envopt.h** file.
2. Backward compatibility is not available when Q.930 / Q.931 software is combined with PSF - Q.930 / Q.931 software. Customers should make sure that the upper and lower layers and the management entity comply with any new interfaces and that no backward compatibility in any interface is required.
3. For the PSF - Q.930 / Q.931 code, the **#ifdef ZQ** section of the **envopt.h** file must be reviewed for the following compile-time options:

Flag	Description
LCZQMILZQ	This flag should be enabled to compile the loosely coupled layer management interface of PSF - Q.930 / Q.931. At run time, the loosely coupled layer manager interface is selected by setting **pst->selector** value to **0**. If this flag is not defined, tight coupling is assumed. At run time, the tightly coupled layer manager interface is selected by setting **pst->selector** value to **1**. For a tightly coupled layer manager interface, the **zq_ptmi.c** file should be compiled with the **SM** option.
CUSTENV	PSF - Q.930 / Q.931 assumes a similar environment (compiler and operating system) on active and standby nodes. Based on this assumption, Trillium supplies non-portable update message packing / unpacking functions. For a different environment, customers can enable this flag and provide portable packing / unpacking in the **zqPtPkStruct** (**zq_bdy3.c**) and **zqPtUnpkStruct** (**zq_bdy5.c**) functions.

4. The **DEBUGP** flag in **envopt.h** file can be enabled so that debug prints from PSF - Q.930 / Q.931 are enabled. If this flag is enabled, the layer manager can send a control request to enable the debug prints at different levels.
5. The **LMINT3** flag is enabled by default in the top section of the **envopt.h** file. This flag must remain enabled for PSF - Q.930 / Q.931.
6. To avoid race conditions at the management interface, PSF - Q.930 / Q.931 should use the same management coupling as Q.930 / Q.931.

7. PSF - Q.930 / Q.931 uses the **cmPFthaYield()** function (**cm_pftha.c** file) to avoid starving other system processes during dynamic control block update. This is not an issue if PSF - Q.930 / Q.931 is ported on an operating system that supports time slicing, because, in that case, the operating system takes care of descheduling the task. To port PSF - Q.930 / Q.931 on a non-preemptive operating system, the **cmPFthaYield** function must be modified to make a descheduling system call.
8. Q.930 / Q.931 and PSF - Q.930 / Q.931 files must be compiled with both **ZQ** and **IN** compile-time flags. For details, refer to the **zq.mak** file.
9. To add the PSIF support in PSF, PSF files must be compiled with the **QW** compile-time flags. For details, refer to the **zq.mak** file.

5.2 PSF - Q.930/Q.931 Compilation

The **zq.mak** file should be used to compile PSF - Q.930 / Q.931. To compile PSF - Q.930 / Q.931 for acceptance tests, the **in.mak** file is used. In this case, the PSF files, as well as the Q.930 / Q.931 files, are compiled.

Abbreviations

The following abbreviations are used in this document:

Abbreviation	Definition
FT / HA	Fault-Tolerant / High-Availability
ISDN	Integrated Services Digital Network
LM	Layer Manager
OOS	Out-Of-Service
PSF	Protocol Specific Function
SAP	Service Access Point
SS	System Services
TAPA	Trillium Advanced Portability Architecture

References

Refer to the following documents for more information:

Fault-Tolerant/High-Availability (FT/HA) Core Functional Specification, Trillium Digital Systems, Inc. (p / n 1091133).

Fault-Tolerant/High-Availability (FT/HA) Core Service Definition, Trillium Digital Systems, Inc. (p / n 1092133).

INT Interface Service Definition, Trillium Digital Systems, Inc. (p / n 1100018).

PSF - Q.930/Q.931 (FT/HA) Functional Specification, Trillium Digital Systems, Inc. (p / n 1091144).

PSF - Q.930/Q.931 (FT/HA) Software Test Sample, Trillium Digital Systems, Inc. (p / n 1094144).

Q.930/Q.931 Functional Specification, Trillium Digital Systems, Inc. (p / n 1091009).

Q.930/Q.931 Portation Guide, Trillium Digital Systems, Inc. (p / n 1093009).

Q.930/Q.931 Service Definition, Trillium Digital Systems, Inc. (p / n 1092009).

Q.930/Q.931 Software Test Sample, Trillium Digital Systems, Inc. (p / n 1094009).

Q.930/Q.931 Training Course, Trillium Digital Systems, Inc. (p / n 1095009).

Q.930 (I.450) - ISDN User-Network Interface Layer 3, ITU.

Q.931 (I.451) - ISDN User-Network Interface Layer 3 Specification for Basic Call Control, ITU.

Q.932 (I.452) - ISDN User-Network Interface Layer 3 Specification - Generic Procedures for the Control of ISDN Supplementary Services, ITU.

System Services Interface Service Definition, Trillium Digital Systems, Inc. (p / n 1111001).