Continuous Computing
Create | Deploy | Converge

Diameter Base Protocol

Service Definition

1092349 1.23a

Diameter Base Protocol

Service Definition

1092349 1.23a

Continuous Computing Corporation *9380 Carroll Park Drive San Diego, CA 92121-2256* ***Phone:*** *+1 (858) 882-8800 Fax: +1 (858) 777-3389* ***Web:*** *http://www.ccpu.com*

Diameter Base Protocol
Service Definition
1092349 1.23a

Contributors: Continuous Computing Development Team, Naveen D’cruz, Kevin MacDowell.

Printed in U.S.A.

Contents

Figures

Tables

Preface

Objective

This document describes in detail the services provided by the Diameter Base Protocol software (p/n 1000349), designed by Continuous Computing Corporation. This product is referred to as Diameter in the rest of the document.

Audience

Continuous Computing assumes that the readers of this document are familiar with telecommunication protocols, specifically IP Multimedia Sub System (IMS).

Document Organization

This document is organized into the following sections:

Section	Contents
1 TAPA Environment	Describes the software environment in which Diameter is designed to operate.
2 System Services Interface	Provides information about the primitives at this interface. Only the system services interface functions unique to Diameter are described in this section.
3 Writing Applications for Trillium's Portable Layer	Issues to be considered when writing applications for TAPA.
4 Upper Layer Interface	Describes the primitives at the AQU Interface.
5 Lower Layer Interface	Describes the primitives at the SCT/HIT Interface.
6 Layer Manager Interface	Describes the layer manager interface primitives and procedures specific to Diameter.
7 Data Structures	Describes the data structures required by Diameter.
Appendix A: Diameter Sample Configuration	Describes the pseudo code for configuring the Diameter layer.
Appendix B: Diameter Performance	Describes the performance analysis of the Diameter Base Protocol software.
Appendix C: Addendum	Describes the addendum changes.

Document Set

The suggested reading order of this document set is:

1. *Diameter Base Protocol Functional Specification*

   Contains the features and highlights that describe the protocol and system characteristics of the software, including the memory characteristics and conformance details.

2. *Diameter Base Protocol Training Course*

   Describes the features and interfaces of the software, including code samples, data flow diagrams, and a list of files.

3. *Diameter Base Protocol 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 software services. The Procedures section describes and shows the flow of primitives and messages across the interfaces.

4. *AQU Interface Service Definition*

   Provides details about the internal upper layer primitives for the **aqu** interface with the Diameter user. The Interface Primitives section describes the software services. The Interface Service Definition describes the interface procedures defined for the service provider software.

5. *Diameter Base Protocol Porting Guide*

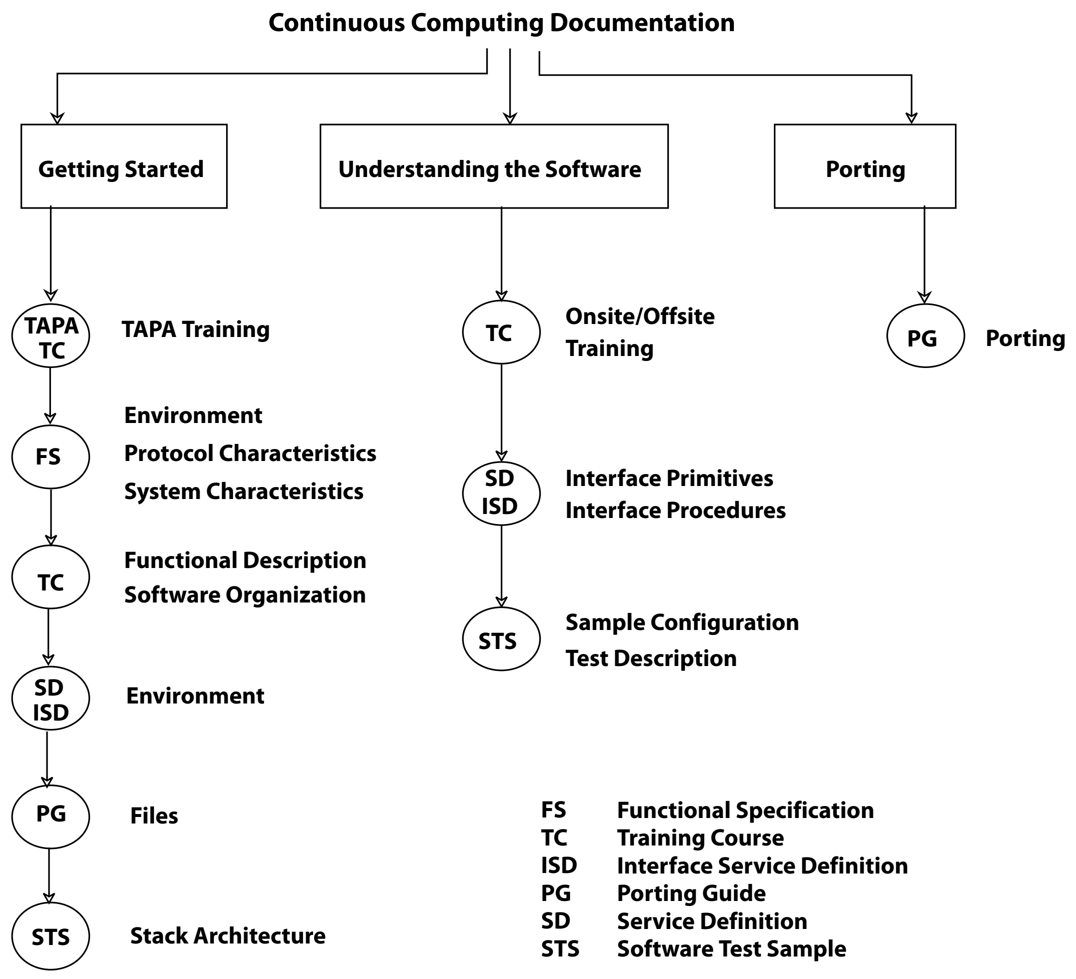
   Describes the files and procedures necessary to port the software to the operating system and into a specific processor family and system architecture. The guide lists the product, common, and sample files associated with the software.

6. *Diameter Base Protocol Software Test Sample*

   Describes the sample files delivered with the product and gives procedures to build a sample test, which partially demonstrates the initialization, configuration, and execution of the product. The document may contain data flow diagrams that show the correct software operation.

Using Continuous Computing® Documentation

The figure below shows the various user approaches to using the software documentation. First time users must read the documents under the **Getting Started** column, where 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 software use, **Understanding the Software** column is suggested. The **Porting** column is for users familiar with Trillium software and related telecommunications protocols and wish to install the software immediately onto their development environments.



FS	**Functional Specification**
TC	**Training Course**
ISD	**Interface Service Definition**
PG	**Porting Guide**
SD	**Service Definition**
STS	**Software Test Sample**

Notations

This table displays the notations used in this document:

Notation	Explanation	Examples
Arial	**Titles**	**1.1 Title**
Book Antiqua	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*	*Diameter Service Definition* This adds *emphasis*.

`Courier New Bold`	**`Code`**
	`Filenames, pathnames`
		`PUBLIC S16`
		`AqUiAquCfgReq (pst, cfg)`
		`Pst       *pst;`
		`AqMngmt   *cfg;`

Release History

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

Version	Date	Author (s)	Description
1.23a	September 03, 2007	Monal Sengar	Update for Toolkit Phase-1 release. Added section for COTS configuration.
1.22a	June 14, 2007	Monal Sengar	Addendum release for updates.
1.21a	April 04, 2007	Monal Sengar	Addendum release for updates.
1.2	December 24, 2006	Monal Sengar	Final release for Diameter software version 1.2.
1.1	January 30, 2006	Boris Naydichev	Initial realease.

1 TAPA Environment

This section describes the software environment in which Diameter is designed to operate.

1.1 Layer Interfaces

Figure 1-1 shows the Diameter interfaces.

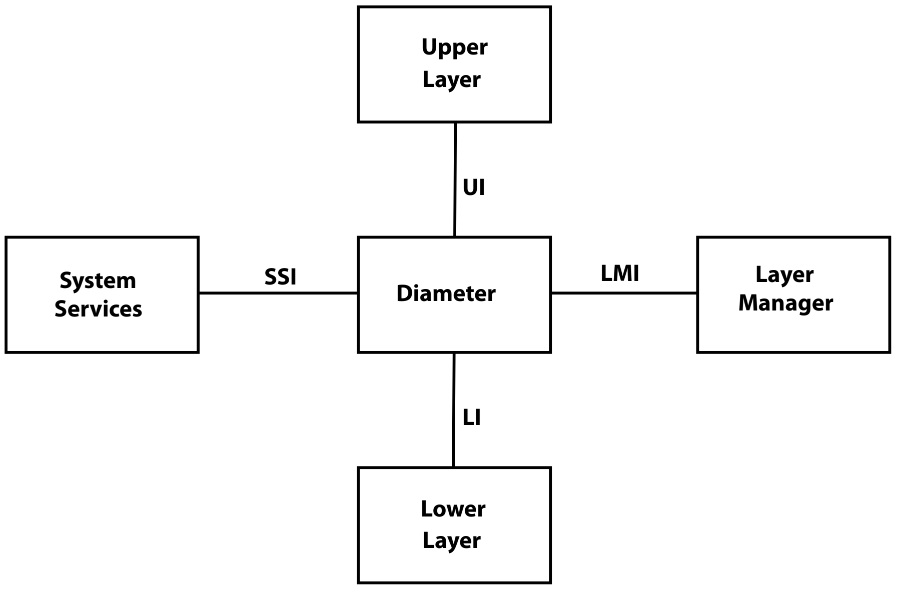


Figure 1-1: Trillium Advanced Portability Architecture

Table 1-1 lists the interfaces and describes their functions.

Table 1-1: Diameter Interfaces

Interface	Description
System Services Interface (SSI)	Provides functions such as buffer management, timer management, date/time management, resource checking, initialization. Refer to the *System Services Interface Service Definition* for details.
Layer Manager Interface (LMI)	Provides the necessary functions to control and monitor the condition of each protocol layer. The interface is described later in this document.
Upper Interface (UI)	The Interface is the service user of the product layer. Refer *AQU Interface Service Definition* for a detailed description of this interface.
Lower Interface (LI)	The HIT/SCT Interface is the service provider used by Diameter for transferring Diameter messages. The TUCL and SCTP Interface is described later in this document. Refer to *SCT/HIT Interface Service Definition* for a detailed description of this interface.

Diameter interacts with the other layers and the layer manager by using the primitives and Service Access Points (SAPs), which are described later in this document**.** Diameter also interacts with system services by using a simple function interface**.**

1.2 Service Access Points

Continuous Computing's Diameter adheres to the Open Systems Interconnection (OSI) reference model. A service user and service provider are present at an interface between two layers in a protocol stack. The service user accesses the services of the service provider at a Service Access Point (SAP). Both the service user and service provider have control blocks that store the state of the SAPs, or managing timers, to manage the interactions at the SAPs. A one-to-one mapping, which defines the SAP, exists between a service user control block and service provider control block.

The Diameter layer provides its services to AQU interface through the upper SAP, and receives its services from the HIT/SCT interface through the lower SAP.

Figure 1-2 shows the relationship between Diameter and the adjacent layers.

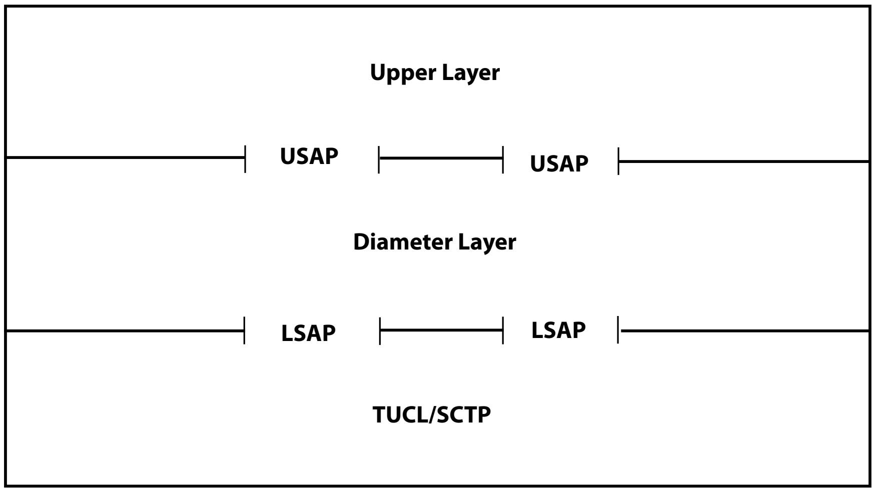


Figure 1-2: Layer definition

The standardized interface of primitives and SAPs allows layers to be defined independently of each other. The peer-to-peer protocol of one layer does not affect any of the upper or lower layer protocols when modified, if the layer interface requirements are met.

There is an implicit SAP between Diameter and the layer manager. There is exactly one SAP at this interface, although it is not explicitly initialized or referenced. At this SAP, the layer manager configures and controls Diameter, retrieves statistics and status information for Diameter, and receives the trace and alarm information from Diameter.

1.2.1 SAP Architecture

Diameter has an upper SAP, per application. Each application is identified by its unique application ID. If there are more than one application running over Diameter then there is one SAP per application.

Diameter has a lower SAP, per network subsystem. Diameter supports the same API at its lower interface for SCTP or TUCL

Figure 1-3 shows the SAP architecture of Diameter.

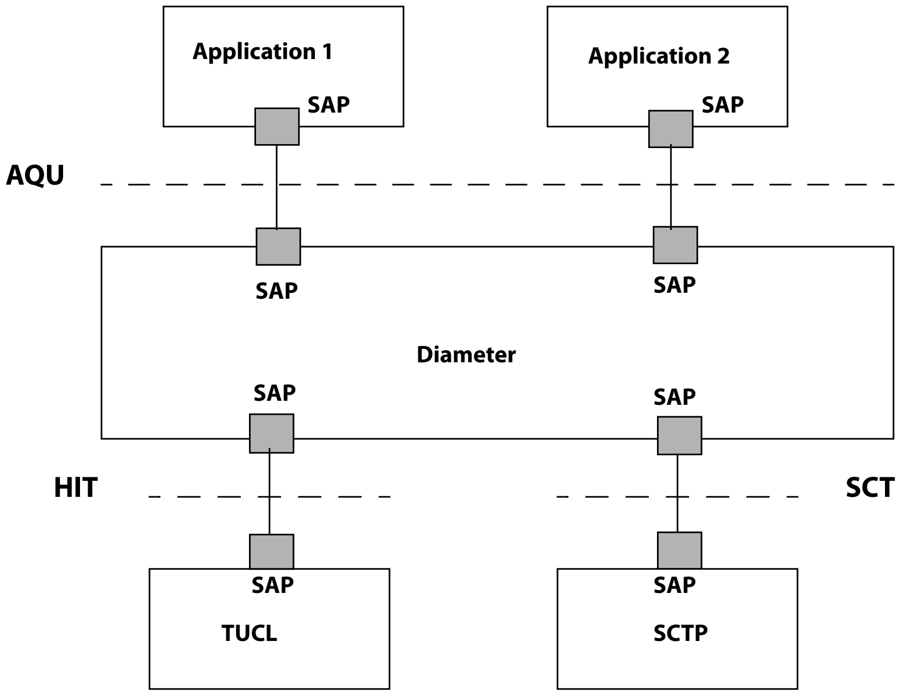


Figure 1-3: SAP architecture of Diameter

1.3 Multi Thread Mode

In normal mode Diameter has two threads, Diameter main task thread and Peer discovery SLP thread. However, for better performance worker threads can be configured for processing encoding/decoding requests in parallel. Diameter main thread, SLP thread and worker threads use the same entity ID 'AQ', however to distinguish between the threads instance IDs can be used.

Maximum number of worker threads is limited to 64. However the recommended value for worker threads is in between 0 to 3.

Figure 1-4 shows the multi threaded architecture of Diameter.

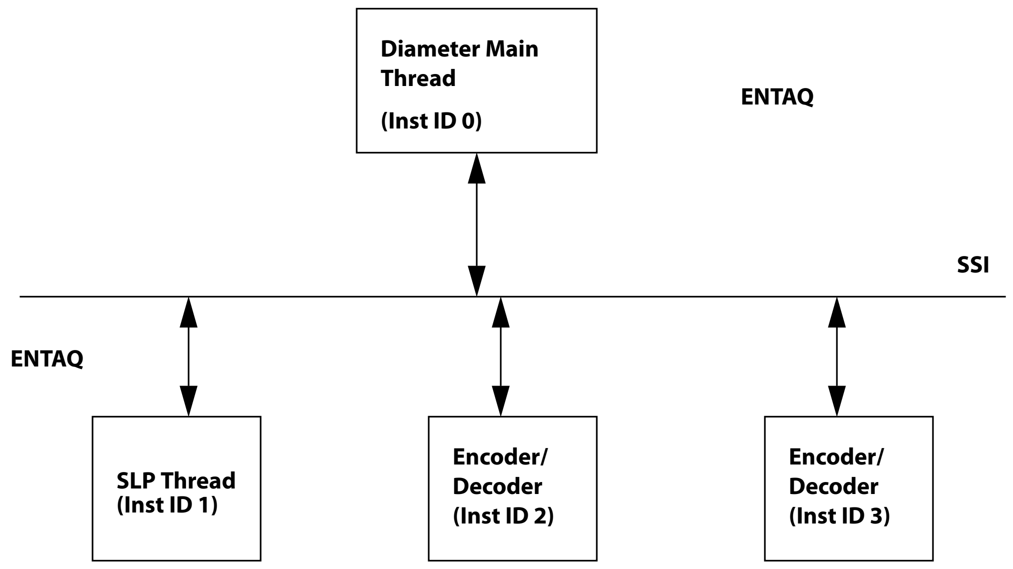


Figure 1-4: Diameter in Multi thread mode

1.4 Primitives

The service user and service provider interact with each other using a set of primitive functions across an SAP.

Interaction between the Diameter software and the upper layer, lower layer, and layer manager takes place using a set of primitive functions. The primitives either initiate or are the result of the interactions between two layers. Primitives completely define the interaction between layers and take the form of:

- Requests (service user to service provider)
- Indications (service provider to service user)
- Responses (service user to service provider)
- Confirms (service provider to service user)

Figure 1-5 shows how primitives function across an SAP.

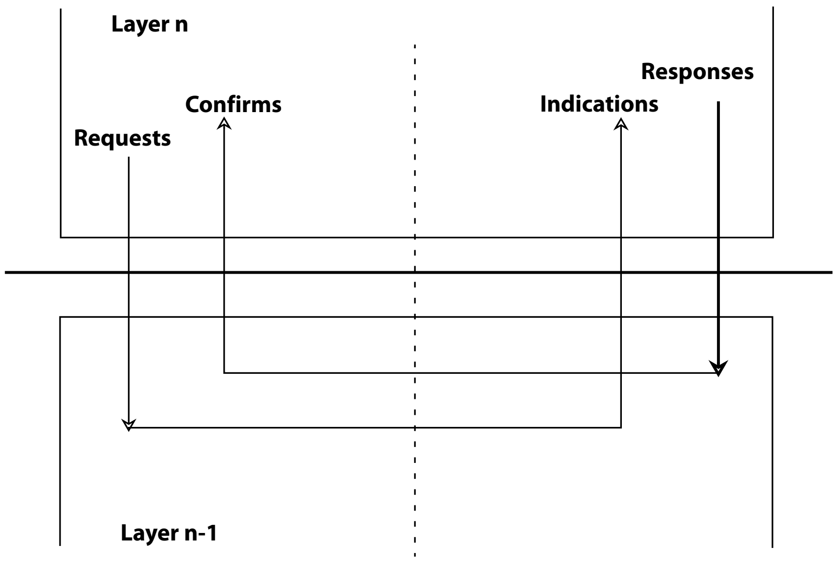


Figure 1-5: Primitive functions

1.5 Primitive Names

Figure 1-6 shows how the interface primitive names are derived.

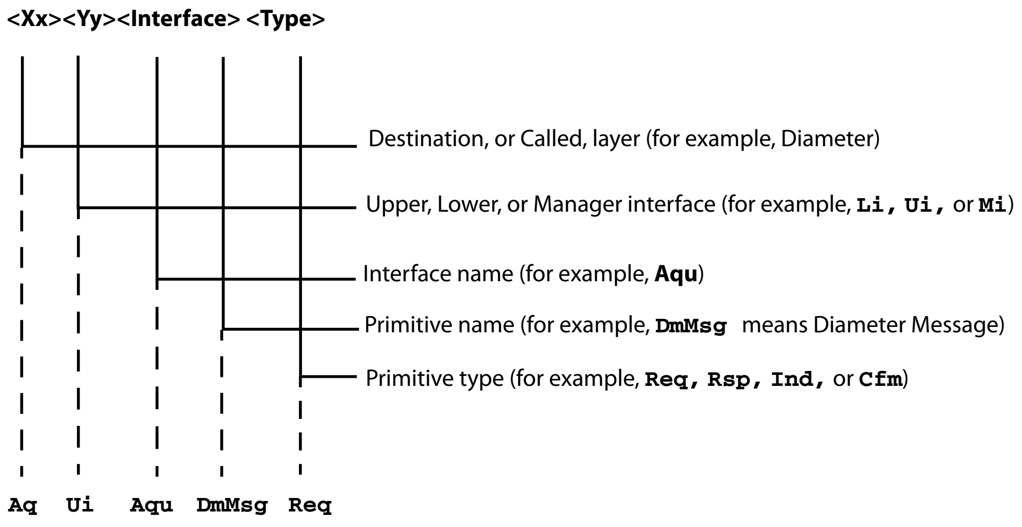

<Xx><Yy><Interface> <Type>
Destination, or Called, layer (for example, Diameter)
Upper, Lower, or Manager interface (for example, Li, Ui, or Mi)
Interface name (for example, Aqu)
Primitive name (for example, DmMsg means Diameter Message)
Primitive type (for example, Req, Rsp, Ind, or Cfm)
Aq
Ui
Aqu
DmMsg
Req

Figure 1-6: Primitive naming conventions

1.5.1 Naming Conventions

These naming conventions apply to the interface primitives:

- Primitive (interface) functions begin with an uppercase letter and the remaining characters are mixed case.
- **`Xx`** represents the layer name. **`XxUiYyy`**, **`XxLiZzz`**, and **`XxMiLxx`** represent the upper, lower, and manager interface abbreviations, respectively. In Figure 1-6, the layer abbreviation **`Aq`** stands for Diameter.
- **`Aq`** is the two-letter prefix for Diameter. The naming conventions for the primitives invoked into the layer are:
  - **`AqUi`** at the upper interface.
  - **`AqMiLaq`** at the management interface.
- The conventions for the primitives invoked from Diameter are:
  - **`XxUi`** at the upper interface, whereby **`Xx`** represents the two-letter prefix of the application layer.
  - **`SmMiLaq`** at the manager interface, whereby **`Sm`** represents the two-letter prefix of a management entity.
- **`Req`**, **`Rsp`**, **`Ind`**, or **`Cfm`** represents the primitive **`type`**. The naming convention for interface primitive follows this order:
  - **`Xx`** is the layer abbreviation.
  - **`Ui`**, **`Li`**, or **`Lmi`** is the upper, lower, or layer manager interface, respectively.
  - **`Lxx`** is the interface abbreviation.

  In Figure 1-6, the primitive type, **`DmMsgReq,`** represents the Diameter Message Request.

A management primitive invoked in the layer manager toward Diameter uses the naming convention **`SmMiLaq<prim_name>`**, whereby **`prim_name`** is the appropriate primitive. For example, to configure Diameter, the layer manager invokes **`SmMiLaqCfgReq`**. This primitive, when terminated in Diameter, uses the naming convention **`AqMiLaq<prim_name>`**, whereby **`prim_name`** is the appropriate primitive. The configuration primitive, **`SmMiLaqCfgReq`**, is terminated in Diameter as **`AqMiLaqCfgReq.`**

Diameter, invoking a primitive toward the layer manager, uses the naming convention **`AqMiLaq<prim_name>`**, whereby **`prim_name`** is the appropriate primitive. For example, Diameter must invoke the primitive **`AqMiLaqCfgCfm`** toward the layer manager to confirm the configuration request. This primitive, when terminated in the layer manager, uses the naming convention **`SmMiLaq<prim_name>`**, whereby **`prim_name`** is the appropriate primitive. The configuration confirm primitive, **`AqMiLaqCfgCfm`**, is terminated in the layer manager as **`SmMiLaqCfgCfm`**.

1.6 Data Types

Table 1-2 lists the primitives and defines the sizes of the primitive data types within the primitives.

Table 1-2: Primitive Data Types and Their Sizes

Mnemonic	# of 8-bit bytes	Sign
`S8`	1	Signed
`U8`	1	Unsigned
`S16`	2	Signed
`U16`	2	Unsigned
`S32`	4	Signed
`U32`	4	Unsigned
`S64`	8	Signed
`U64`	8	Unsigned
`PTR`	as required	Unsigned

Note: 1. The size of the **`PTR`** depends on the specific machine on which the software is ported.

2. **S64** and **U64** is available under compile time flag BIT_64.

1.7 Common Identifiers

Common identifiers (IDs) are used across the interfaces provided by Diameter. Also, some parameters are common across specific interfaces. This section describes some of the common IDs.

1.7.1 Entity ID

In TAPA, each protocol layer is assigned a unique ID, known as the entity ID. System services uses this ID to determine the specific layer from which a primitive was invoked and to which layer the primitive is destined.

1.7.2 Instance ID

In TAPA, the instance ID distinguishes between multiple instances of a single protocol layer. Currently, TAPA layers are designed to have global variables. Multiple instances can be used only when each instance has its own address space, that is, in host-based environments like SUN Solaris or Windows NT. In embedded systems such as VxWorks, in which the address space is usually shared across multiple tasks, multiple TAPA layer instances cannot be used.

1.7.3 Processor ID

In TAPA, the processor ID is an additional ID, which identifies a logical processor on which the layer executes. This can be the logical processor ID of a process in a multiprocessing environment, or it can be used to identify a specific processor on which the layer executes. The format and interpretation of this field depends on the implementation of the message scheduler and the system services interface.

1.8 Common Structures

Parameters have this generic calling sequence:

`<prim_name> (pst, suId/spId, suConnId/spConnId...)`

The parameters described in the subsequent sections frequently occur.

1.8.1 Post

In TAPA, a system consists of multiple TAPA entities or tasks. The post structure:

- Exchanges and communicates primitives between various tasks
- Begins all the UI, MI, and LI primitives
- Contains the information required to identify source and destination TAPA tasks. When the interface is loosely coupled between the source and the destination layers, the source layer provides information required by system services to route the message buffer to the correct destination layer. In the destination layer, the post structure is used only to verify the identity of the source and the specific primitive.
- Contains information allowing a message scheduler to schedule messages efficiently. Each message can be assigned a priority and messages can be broadcast to all the tasks.

```
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) */
#ifdef TDS_ROLL_UPGRADE_SUPPORT
   CmIntfVer intfVer; /* Interface Version Number (U16) */
#else
   U16 spare1;        /* spare 1 (U16) */
#endif
} Pst;
```

Table 1-3 lists the structure parameters and describes their functions.

Table 1-3: Structure Parameters

Parameter	Description
`dstProcId`	Destination Processor ID. Identifies the logical processor of the destination task to which the primitive must be delivered. In a service user SAP, the layer manager configuration provides the value. In a service provider SAP, this value is obtained during the bind procedures. Use this parameter only when the source and destination layers are loosely coupled. Also, refer to Section 1.7.3.
`srcProcId`	Source Processor ID. Identifies the logical processor of the source task, which originated the primitive. This value is obtained when system services initializes the layer. Also, refer to Section 1.7.3.
`dstEnt`	The Entity ID of the destination layer. In a service user SAP, the layer manager configuration provides this value. In a service provider SAP, this value is obtained during the bind procedures. Also, refer to Section 1.7.1.
`dstInst`	The Instance ID of the destination layer. In a service user SAP, the layer manager configuration provides this value. In a service provider SAP, this value is obtained during the bind procedures. Also, refer to Section 1.7.2.
`srcEnt`	The Entity ID of the originating layer. This value is obtained when system services initializes the layer. Also, refer to Section 1.7.1.
`srcInst`	The Instance ID of the originating layer. This value is obtained when system services initializes the layer. Also, refer to Section 1.7.2.
`prior`	Message priority. This value is provided at the SAP configuration. This value can be used to implement a message priority mechanism. Use of this field is implementation-dependent.
`route`	Message route. This field can be used in addition to the aforementioned IDs to route a message from the originating layer to the destination layer. This value is provided at the SAP configuration. Use of this field is implementation-dependent.
`event`	Event type. This parameter identifies the type of generated primitive. The IDs of all the primitives generated at an interface are defined in the corresponding interface header files.
`region`	Region ID. This is the ID of the dynamic memory region from which the primitive message buffers are allocated. The parameter is provided during configuration.

Table 1-3: Structure Parameters

Parameter	Description
`pool`	Pool ID. This parameter identifies the dynamic memory pool from which the primitive message buffers are allocated. This parameter is provided during configuration.
`selector`	Coupling ID. This parameter identifies whether the interface between the source and destination layers is loosely or tightly coupled. When there are multiple destination layers, this parameter also selects the specific destination layer for the tightly coupled operation. This parameter is provided during configuration.
`intfVer`	Interface Version Number. This element is used to support the rolling upgrade feature.
`spare1`	Spare. This parameter is used only for alignment.

1.8.2 SpId and SuId

When the service user calls a primitive, the **`SpId`** identifies the SAP number in the service provider to which the primitive is directed.

When the service provider calls a primitive, **`SuId`** identifies the SAP number in the service user to which the primitive is directed.

`SpId` and **`SuId`** associate a received primitive with a specific SAP in a layer. The service user and service provider define separate IDs, because each ID may have a different set of SAP control blocks for which different IDs may be supplied.

1.8.3 Date and Time

The date and time structure is filled by Diameter when resending the statistics values. It has the format:

```
typedef struct dateTime
{
   U8 month;
   U8 day;
   U8 year;
   U8 hour;
   U8 min;
   U8 sec;
   U8 tenths;
} DateTime;
```

Table 1-4 lists the structure parameters and describes their functions.

Table 1-4: Structure Parameters of `dateTime`

Name	Description and Allowable Values
`month`	Month. The allowable values are: 1 to 12.
`day`	Day. The allowable values are: 1 to 31.
`year`	Year. The allowable values are: 1 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. Trillium products use one byte to store the year value. Thus, the years 1900 to 2155 can be represented without causing an overflow. Customers can add the value 1900 to the value provided by the Trillium product to get the 4-digit year. Refer to the **Note**.
`hour`	Hour. The allowable values are: 0 to 23.
`min`	Minute. The allowable values are: 0 to 59.
`sec`	Second. The allowable values are: 0 to 59.
`tenths`	Tenths of a second. The allowable values are: 0 to 9.

Note: The `DateTime` structure is defined in the file `gen.x`. The layer management can compute the absolute year field by adding 1900 to the value returned by the Trillium product. This C code shows how this can be done:

```
U32         absYear;              /* Absolute year */
DateTime    tTime;
(Void)SGetDateTime(&tTime);
absYear = tTime.year + 1900;
```

1.8.4 Ticks

System timestamp is received through an SSI system call, **`SGetSysTime`**, which returns the number of *Ticks* at that instant.

```
typedef U32 Ticks;                /* system clock ticks */
```

1.8.5 Header

The header structure carries all the common information required to identify the management primitive.

Each management primitive has a single parameter consisting of a header structure, which is followed by a structure and specific to the type of invoked primitive.

```
typedef struct header      /* header */
{
   U16      msgLen;        /* message length - not used */
   U8       msgType;       /* message type - not used */
   U8       version;       /* version - not used */
   U16      seqNmb;        /* sequence number - not used */
   EntityId entId;         /* entity ID - used always */
   ElmntId elmId;          /* element ID - used sometimes */
#ifdef LMINT3
   TranId   transId;       /* sequence number - mandatory */
   Resp     response;      /* struct containing response */
#endif /* LMINT3 */
} Header;
```

Table 1-5 lists the fields and defines their functions.

Table 1-5: Fields of header

Field	Description and Allowable Values
`msgLen` **`msgType`** **`version`** **`seqNm`**	Unused.
`elmId`	Identifies the layer-specific resource to which the management primitive applies. Also, refer to Section 1.8.5.1.
`transId`	Used by the layer manager to correlate the request primitives generated to the Diameter and the confirm responses generated by Diameter to the layer manager. A correlation is required only if the layer manager generates multiple management requests with the same **`elmId`** value; otherwise, this field may be ignored. The allowable values: 1 to (2^{32}-1).

Table 1-5: Fields of header

Field	Description and Allowable Values
`response`	Filled by the layer manager when invoking the primitives toward Diameter. This information is used by Diameter in the response primitive sent to the layer manager. Also, refer to Section 1.8.5.2.

1.8.5.1 ElmntId

This field identifies the specific resource to which the management primitive pertains. Only the **`Elmnt`** field of this structure is mandatory. The Diameter layer does not use the remaining fields, **`ElmntInst1`**, **`ElmntInst2`**, and **`ElmntInst3`**. The structure format is:

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

Table 1-6 lists the fields and defines their functions.

Table 1-6: Fields of elmntId

Field	Description and Allowable Values
`elmnt`	Identifies the layer resource. The use of each resource is explained in the description of the individual management primitives, for example: • **`STAQGEN`**: Used in the management primitives to refer to the entire layer; • **`STAQUSAP`**: Used to manage the upper SAP; • **`STAQLSAP`**: Used to manage the lower SAP; • **`STAQPROT`**: Used for the specific protocol
`elmntInst1` **`elmntInst2`** **`elmntInst3`**	Unused.

1.8.5.2 Resp

The layer manager uses this field to convey information to Diameter of the parameters, which must be filled in the confirmation primitive by the layer. The data structure and its parameters are:

```
typedef struct resp
{
   Selector  selector;       /* 'rev' selector: mandatory */
   Priority  priority;       /* 'rev' priority: mandatory */
   Route     route;          /* 'rev' route: mandatory */
   Memory    mem;            /* 'rev' mem/pool: mandatory */
}Resp;
```

Table 1-7 lists the fields and defines their functions.

Table 1-7: Fields of resp

Field	Description and Allowable Values/Reference
`selector`	The allowable values are: • `LAQ_SEL_LC`: indicates that the layer manager is loosely coupled. • `LAQ_SEL_TC_SM`: indicates that the layer manager is tightly coupled. Also, refer to Section 1.8.1.
`priority`	The allowable values are: 0 to 255. Also, refer to Section 1.8.1.
`route`	The allowable values are: 0 to 255. Also, refer to Section 1.8.1.
`mem`	Identifies the region and pool values from which the message buffer to be sent to the layer manager must be allocated by Diameter. Also, refer to Section 1.8.7.

1.8.6 Common Status

This parameter specifies the status of the request in the confirm messages. The structure format is:

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

Table 1-8 lists the fields and defines their functions:

Table 1-8: Fields of cmStatus

Field	Description and Allowable Values
`status`	Request status. The allowable values are: • `LCM_PRIM_OK` • `LCM_PRIM_NOK`
`reason`	Failure reason. Refer to the specific section for the allowable values.

1.8.7 Memory

This parameter specifies the memory region and pool values. The structure format is:

```
typedef struct mem          /* memory */
{
   Region region;           /* region */
   Pool pool;               /* pool */
   U16 spare;               /* spare for alignment */
}mem;
```

Note: The description and allowable values for the region and pool fields are the same as those in the description of the `Post` structure, in Section 1.8.1. The `spare` field is used for alignment only.

1.8.8 Timer Configuration

This parameter specifies the timer values. The structure format is:

```
typedef struct tmrCfg           /* timer configuration structure */
{
   Bool enb;                    /* enable */
   U16 val;                     /* value */
} TmrCfg;
```

Table 1-9 lists the fields of the timer configuration:

Table 1-9: Fields of tmrCfg

Field	Description and Allowable Values
enb	Enabled. The allowable values are: • **TRUE** • **FALSE**
val	Timer value. Refer to the specific timer for the allowable values.

1.9 Terms

Table 1-10 contains a list of terms, and their definitions, used in this document.

Table 1-10: Terms

Name	Definition
AAA	Authentication, Authorization, and Accounting protocols are deployed to provide dial-up PPP and terminal server access.
Accounting	Act of collecting information on resource for the purpose of capacity planning, auditing, billing.
Authentication	Act of verifying the identity of user.
Authorization	Act of determining whether the user can use the resource or not.
RADIUS	Remote Authentication Dial In User Service an authentication protocol, over which the Diameter was developed.
TACACS	An access control protocol.
AVP	Attribute-Value-Pair includes a header and is used to encapsulate the protocol-specific data.
Agent	An agent is a node that provides relay, proxy, redirect, and translation services.
Diameter Node	Is a host process that implements the Diameter protocol, and acts as a Client, Agent, or Server.
Diameter Peer	Is a direct node to which a given Diameter node has a direct transport connection.
Realm	The string in the NAI that immediately follows '@' character.
Home Realm	Is the administrative domain with which the user maintains the account relationship.
Local Realm	Is the administrative domain which provides services to users.
NAI	Network Access Identifier, is used to extract the users identity and realm name. The identity is used during authentication or authorization of user, while the realm name is used while routing.
Proxy Agent	An agent is called as proxy when it is applying policy decisions in addition to forwarding requests and responses. Proxy agents may also keep the state of the sessions or NAS resources.

Name	Definition
Relay Agent	An agent is called as relay when an agent forwards the requests and responses based on the routing-related AVPs. Since relays do not apply policy decisions they do not examine or alter the non-routing AVPs. Relays do not keep state of sessions or NAS resources.
Redirect Agent	An agent is called as redirect agent when it allows the client and serer communicate directly rather than forwarding requests and responses. The redirect agents do not alter any AVPs inside the message. Redirect agents do not keep state of sessions or NAS resources.
Translation Agent	An agent is called as translation agent when a node performs any translation functionality from Diameter to any AAA protocol like RADIUS and TACACS.
Downstream	This is used to identify the direction of a particular message from home server towards the access device.
Upstream	This is used to identify the direction of a particular message from the access device towards home server.
User	The entity requesting or using some resource, in support of which a Diameter client has generated a request.
Cx	The reference point which used communicate between I-CSCF/S-CSCF and HSS.
Dx	The reference point which is used to communicate between I-CSCF/S-CSCF to find a correct HSS in a multi-HSS environment.
Sh	The reference point which is used to exchange information between SIP AS/OSA SCS and HSS.
Dh	The reference point which is used by AS to find correct HSS in a multi-homed HSS environment.
Gq	The reference point which is used exchange policy decisions related information between P-CSCF and PDF.

1.10 Acronyms

Table 1-11 lists the acronyms used in this document.

Table 1-11: Acronyms

Abbreviation	Description
IMS	IP-Multimedia Subsystem
3GPP	Third Generation Partnership Program
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
SCTP	Stream Control Transmission Protocol
TUCL	TCP/UDP Convergence Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TLS	Transport Layer Security
IPSec	IP Security
NAPTR	Naming Authority Pointer
SLPv2	Service Location Protocol Version 2
NAI	Network Access Identifier
NAS	Network Access Server
AVP	Attribute Value Pair
TAPA	Trillium Advanced Portability Architecture
HIT	TUCL Upper Interface
SCT	SCTP Upper Interface

For a list of commonly used terms please refer to the Engineering Glossary (part number PREN026) at http://www.ccpu.com/search/glossary/

2

Writing Applications for Trillium's Portable Layer

2.1 Guidelines

This section highlights issues to consider when writing application, layer manager, or lower layer interfaces with Trillium's portable layer conforming to TAPA. Customers using multiple layers in a stack must consider these points while writing the lower most layer, application and the stack manager.

2.1.1 Reentrancy

Trillium portable layers are not reentrant (unless specified). There must not be more than one thread executing the layer code simultaneously.

2.1.2 Coupling – Tight/Loose

TAPA provides flexibility for the application interfacing with the portable layer. The application interface may be coupled tightly or loosely. Continuous Computing strongly advises the same coupling be used for all primitives in the same direction on an interface. Do not mix coupling for primitives on an interface in same direction. For example all primitives from a Service user to a provider must have the same coupling. It is acceptable to use different couplings (for all primitives) on different interfaces (upper, lower or layer manager interface).

Tightly coupled interfaces direct function calls to the adjacent layers. The primitive is not packed and unpacked on the sending and receiving side respectively. Tightly coupled interfaces give better performance particularly when the primitives carry structures that take CPU cycles to pack and unpack. A tightly coupled scenario typically requires more stack size due to direct function calls.

Loosely coupled interface result in packing and unpacking of each primitive. This enables two adjacent layers to work as a separate threads which is not possible in tightly coupled interface because Trillium layers are not reentrant.

2.1.3 Buffer Management

Trillium portable layers assume that the receiver of the primitive de-allocates the allocated message buffers. This is applicable in both tight and loose coupling scenarios. For example, if a message is received by a portable layer and it passes the message buffer (type Buffer *) to its application, then it's the application's responsibility to de-allocate the buffer when included in the primitive.

Similarly, if an application sends some data in message buffer format (as defined by the primitive) then the application must not de-allocate the buffer after invoking the primitive.

Continuous Computing advises referring to the sample code for the layer regarding this scenario. A message buffer may not be presented as a parameter in the primitive, it can be presented as a field in a parameter. For these cases the same rule applies.

3

System Services Interface

This section describes the System Services Interface (SSI) primitives. Refer to the *System Services Interface Service Definition* for more information. Diameter does not use all the functions defined in that particular document; only the functions unique to Diameter are described here.

3.1 Diameter-Specific Primitives

This section lists the functions required by Diameter.

Initialization

System services calls the initialization function to initialize a task (layer).

Table 3-1: Initialization Management Functions

Name	Description
`SRegTTsk`	Registers an initialization task.
`aqActvInit`	Initializes a task for Diameter.

Task Scheduling

System services calls the task scheduling function to register, activate, and terminate a task.

Table 3-2: Task Scheduling Management Functions

Name	Description
`SRegActvTsk`	Registers a layer activation task.
`SExitTsk`	Cleans and exits a task.
`SPstTsk`	Posts a message buffer to a destination task.
`SFndProcId`	Locates the processor ID on which a task runs.
`aqActvTsk`	Schedules a layer activation task for Diameter.

Timer Management

System services periodically calls the timer functions to activate the tasks.

Table 3-3: Timer Management Functions

Name	Description
`SRegTmr`	Registers a timer activation task.
`aqActvTmr`	Activates a task for the Diameter timers.

Structure Memory Management

System services calls the structure memory management functions to allocate and deallocate variable-sized buffers used as structures.

Table 3-4: Memory Management Functions

Name	Description
`SGetSMem`	Allocates the structure memory.
`SGetSBuf`	Allocates a buffer from the structure memory.
`SPutSBuf`	Deallocates a buffer and returns it to structure memory.

Message Management

The message management functions initialize, add, and remove data both to and from messages. The structure of a message is system services-specific, and a layer does not recognize it.

Table 3-5: Message Management Functions

Name	Description
`SGetMsg`	Allocates a message from memory.
`SPutMsg`	Deallocates a message and returns it to memory.
`SAddPstMsg`	Adds a byte to the tail of a message.
`SRemPstMsg`	Removes a byte from the tail of a message.
`SRepMsg`	Replaces a specified byte in a message with a new value.
`SExamMsg`	Returns the value of a specified byte in a message.
`SFndLenMsg`	Finds the length of a message.
`SPrntMsg`	Prints the contents of a message.
`SCpyMsgMsg`	Copies a message into a newly allocated message buffer.
`SCatMsg`	Concatenates two message buffers.
`SPkU8`	Adds an unsigned 8-bit value to the head of a message.
`SPkU16`	Adds an unsigned 16-bit value to the head of a message.
`SPkU32`	Adds an unsigned 32-bit value to the head of a message.
`SPkS8`	Adds a signed 8-bit value to the head of a message.
`SPkS16`	Adds a signed 16-bit value to the head of a message.
`SPkS32`	Adds a signed 32-bit value to the head of a message.
`SUnpkU8`	Removes an unsigned 8-bit value from the head of a message.
`SUnpkU16`	Removes an unsigned 16-bit value from the head of a message.
`SUnpkU32`	Removes an unsigned 32-bit value from the head of a message.
`SUnpkS8`	Removes a signed 8-bit value from the head of a message.
`SUnpkS16`	Removes a signed 16-bit value from the head of a message.
`SUnpkS32`	Removes a signed 32-bit value from the head of a message.

Queue Management

The queue management functions initialize, add messages to, and remove messages from queues. The structure of a queue is system services-specific, and a layer does not recognize it. Currently, queues are not used in the Diameter layer.

Table 3-6: Queue Management Functions

Name	Description
`SInitQueue`	Initializes a queue.
`SQueueLast`	Adds a message to the tail of a queue.
`SDequeueFirst`	Removes a message from the head of a queue.
`SFndLenQueue`	Locates the length of a queue.

Miscellaneous

The miscellaneous functions manage date and time structures, handle errors, and check resource availability.

Table 3-7: Miscellaneous Functions

Name	Description
`SLogError`	Handles a system error.
`SChkRes`	Reports available system resources, such as message buffer memory.
`SGetDateTime`	Gets the current date and time.
`SGetSysTime`	Gets the system time in system ticks.
`SPrint`	Prints a preformatted string.

3.2 System Services Primitives

3.2.1 Diameter Initialization Task

Name	`aqActvInit`
Direction	System services to Diameter
Supplied	Yes

Description

The initialization procedure initializes Diameter. The layer manager begins the procedure when it registers the initialization function for Diameter using the **`SRegTTsk`** primitive. System services then calls the initialization function **`aqActvInit`**. The initialization function must be called only once before any other primitives or functions in Diameter are called.

The layer manager initializes a task across a tightly coupled interface in Figure 3-1.

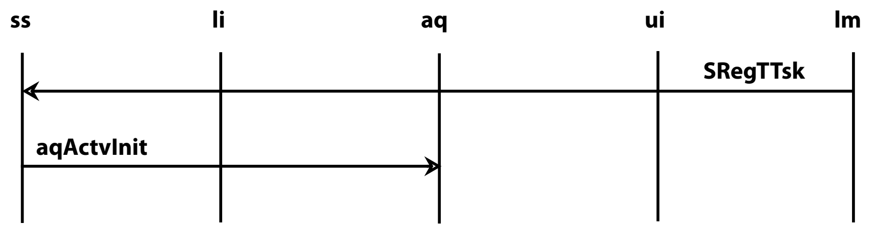


Figure 3-1: Initialization data flow

Synopsis

```
PUBLIC S16 aqActvInit (ent, inst, region, reason)
Ent         ent;
Inst        inst;
Region      region;
Reason      reason;
```

Parameters

Parameter	Description and Allowable Values	Refer
`ent`	Entity ID assigned to Diameter. The allowable value: **`ENTAQ`**.	Section 1.7.1
`inst`	Instance ID assigned to Diameter. The allowable value: 0 to 3 (depends on the maximum number of threads).	Section 1.7.2
`region`	Memory region ID assigned to Diameter. This ID is used to allocate the structure memory.	*Refer to SSI Service Definition.*

`reason`	Reason for initialization. The allowable values are:
	`LAQ_NRM_TERMINATION`
	`LAQ_SHUTDOWN`
	`LAQ_PWR_UP`
	`LAQ_DONT_CARE`
		Refer to SSI Service Definition.

3.2.2 Diameter Activation Task

Name	`aqActvTsk`
Direction	System services to Diameter
Supplied	Yes

Description

The task activation procedure activates Diameter when a loosely coupled interface awaits a message from Diameter. The layer manager begins the procedure when it registers the task activation function by using the **`SRegTTsk`** primitive. Then, system services calls the task activation function, **`aqActvTsk`**, when another layer in the system posts a message, by using the **`SPstTsk`** primitive, to Diameter from a loosely coupled interface. Diameter invokes the **`SExitTsk`** primitive before returning from the **`aqActvTsk`** primitive, so that system services can complete any bookkeeping tasks.

The layer manager posts messages across a loosely coupled LAQ interface in Figure 3-2.

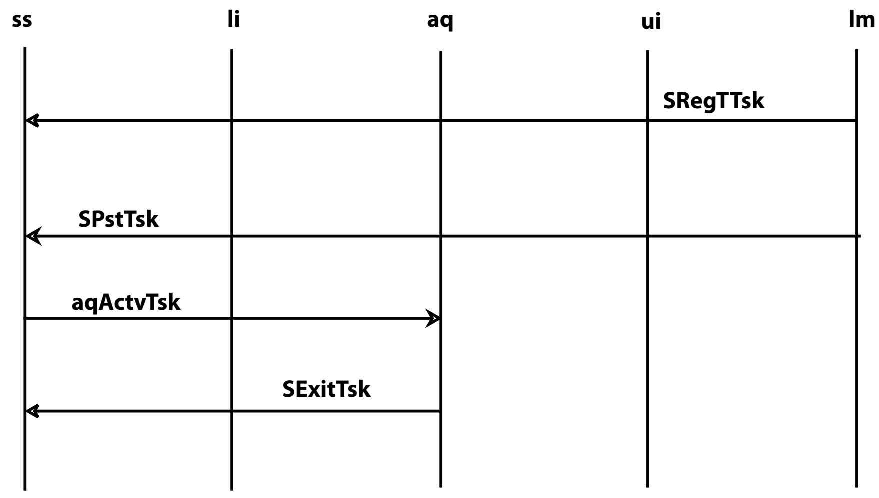


Figure 3-2: Task activation data flow

Synopsis

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

Parameters

Parameter	Description	Refer
pst	Post structure.	Section 1.8.1
mBuf	Task buffer. The message buffer containing the primitive.	*Refer to SSI Service Definition.*

3.2.3 System Services to Diameter

Description

The timer activation procedure periodically activates Diameter to manage its own timers. Diameter begins the procedure when it registers the timer activation function by using the **`SRegTmr`** primitive; **`SRegTmr`** also specifies the time period between successive timer activations. Then, system services periodically calls the timer activation function, **`aqActvTmr`**, which is typically called during the general configuration, and when Diameter obtains the required time resolution from the layer manager.

Diameter initiates and system services calls the timer activation function in Figure 3-3.

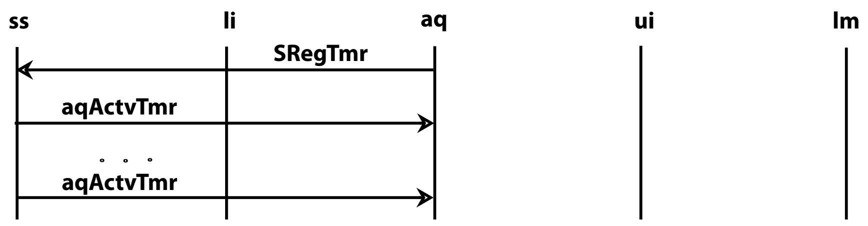


Figure 3-3: Timer activation data flow

Synopsis

```
PUBLIC S16 aqActvTmr()
```

Parameters

None.

4

Upper Layer Interface

Refer to the *AQU Interface Service Definition (p/n 1100073)* for more information about this interface.

5

Lower Layer Interface

Refer to the *SCT (p/n 1100036) and HIT (p/n 1100031) Interface Service Definition documents* for more information about this interface.

6 Layer Manager Interface

This section describes the layer manager interface primitives, data structures, and procedures specific to Diameter. The layer manager interface of Diameter is described in the next section.

6.1 Primitive Listing

This section lists the layer manager interface functions.

Configuration

This function configures the protocol layer resources.

Table 6-1: Configuration Primitives

Name	Description	Refer to
`AqMiLaqCfgReq`	Configuration request.	Section 6.3.1
`AqMiLaqCfgCfm`	Configuration confirm.	Section 6.3.2

Statistics

These functions determine the traffic loads and quality-of-service for the protocol layer:

Table 6-2: Statistics Primitives

Name	Description	Refer to
`AqMiLaqStsReq`	Statistics request.	Section 6.3.3
`AqMiLaqStsCfm`	Statistics confirm.	Section 6.3.4

Solicited Status

These functions indicate the current state of the protocol layer:

Table 6-3: Solicited Status Primitives

Name	Description	Refer to
`AqMiLaqStaReq`	Status request.	Section 6.3.5
`AqMiLaqStaCfm`	Status confirm.	Section 6.3.6

Unsolicited Status

This function indicates a change in the status of the protocol layer:

Table 6-4: Unsolicited Status Primitives

Name	Description	Refer to
`AqMiLaqStaInd`	Status indication.	Section 6.3.7

Control

These functions activate and deactivate the protocol resources:

Table 6-5: Control Primitives

Name	Description	Refer to
`AqMiLaqCntrlReq`	Control request.	Section 6.3.8
`AqMiLaqCntrlCfm`	Control confirm.	Section 6.3.9

Unsolicited Trace

This function traces the incoming and outgoing Diameter messages to the layer manager:

Table 6-6: Unsolicited Trace Primitives

Name	Description	Refer to
`AqMiLaqTrcInd`	Trace indication.	Section 6.3.10

Probe

This function probes for the information related with Peers, Realms and DM and AVP Dictionary

Table 6-7: Probe Primitives

Name	Description	Refer to
`AqMiLaqPrbReq`	Probe request.	Section 6.3.11
`AqMiLaqPrbCfm`	Probe confirm.	Section 6.3.12

6.2 Product-Specific Structures

Each management primitive has the common parameters: **`Pst`** and **`AqMngmt`**. These parameters are described in the next sections.

6.2.1 Pst

The **`Pst`** structure is described in Section 1.8.1, "Post". Table 6-8 lists the individual field values of the **`Pst`** structure at the management interface, which are initiated by the layer manager.

Table 6-8: Parameters of Pst Structure for the primitives initiated by LM to Diameter

Parameter	Allowable Values
`dstProcId`	Diameter processor ID. The allowable values: 0 to 255.
`dstEnt`	Diameter entity ID. The allowable value: **`ENTAQ`**.
`dstInst`	Diameter instance ID. The allowable values: 0 to 255.
`srcProcId`	Layer manager processor ID. The allowable values: 0 to 255.
`srcEnt`	Layer manager entity ID. The allowable value: **`ENTSM`**.
`srcInst`	Layer manager instance ID. The allowable values: 0 to 255.
`prior`	Specific event priority value.
`route`	Route information, if any.
`event`	Specific management request event. The individual event values are specified in each primitive description.
`region`	Memory region information.
`pool`	Memory pool information.
`selector`	Specifies whether the layer manager is loosely or tightly coupled with Diameter. It is used only to route the primitive in the layer manager.
`intfVer`	Specifies the interface version number. The value in this field is **`LAQIFVER`**. This field is present only when the rolling upgrade support is provided by the product.

Table 6-9 lists the individual field values of the **`Pst`** structure at the management interface, which are initiated by Diameter.

Table 6-9: Parameters of Pst structure for the Primitives Initiated by Diameter to LM

Parameter	Allowable Values
`dstProcId`	Layer manager processor ID. The allowable values: 0 to 255.
`dstEnt`	Layer manager entity ID. The allowable value: **`ENTSM`**.
`dstInst`	Layer manager instance ID. The allowable values: 0 to 255.
`srcProcId`	Diameter processor ID. The allowable values: 0 to 255.
`srcEnt`	Diameter entity ID. The allowable value: **`ENTAQ`**.
`srcInst`	Diameter instance ID. The allowable values: 0 to 255.
`prior`	Specific event priority value.
`route`	Route information, if any.
`event`	The specific management response or indication event. The individual event values are specified in the description of each of the primitives.
`region`	Memory region information.
`pool`	Memory pool information.
`selector`	Specifies whether Diameter is loosely or tightly coupled with the layer manager. It is used only to route the primitive in Diameter.
`intfVer`	Specifies the interface version number. The values in this field are the same as that received from the layer manager. This field is present only when the rolling upgrade support is provided by the product.

6.2.2 AqMngmt

The management structure is:

```
typedef struct _aqMngmt
{
   Header      hdr;         /* Header */
   CmStatus    cfm;         /* Primitive's status/conf. */
   union
   {
       AqCfg    cfg;        /* Configuration */
       AqCntrl  cntrl;      /* Control */
       AqSsta   ssta;       /* Solicited status */
       AqUsta   usta;       /* Unsolicited status */
       AqSts    sts;        /* Statistics */
       AqTrc    trc;        /* Trace */
       AqPrb    probe;      /* Probe */
   } u;
} AqMngmt;
```

Every management primitive has a common **`Header`** and **`CmStatus`** (for confirms) section. These data structures and their uses are described here. Other data structures used in multiple primitives at the management interface are also described here. The remaining data structures are specific to each of the management primitives and are described in detail as part of the section describing the individual primitives.

6.2.2.1 Header

Refer to Section 1.8.5 for details.

6.2.2.2 Common Status

Refer to Section 1.8.6 for details

6.3 Primitives and Procedures

These rules apply to each flow diagram in this section:

1. Time flows toward the bottom of the page.
2. The mnemonic above a line represents a function call or Diameter primitive.
3. The mnemonic below a line represents a Diameter message type.
4. A **+** indicates an OR condition (one path or another can be taken).
5. A **o** indicates an AND condition (both paths are taken in parallel).

Table 6-10 lists and defines the abbreviations used above the flow diagrams.

Table 6-10: Abbreviations Used in Flow Diagrams

Abbreviation	Definition
`ss`	System services.
`sb/hi`	SCTP/TUCL.
`aq`	Diameter.
`ab`	Diameter user.
`lm`	Layer manager.

The Diameter-specific procedures are described in this document.

6.3.1 Configuration Request

Name	Configuration request. **`SmMiLaqCfgReq`** is invoked in the layer manager. The **`AqMiLaqCfgReq`** processes this request in Diameter.
Direction	Layer manager to Diameter
Supplied	Yes
Response	Yes

Primitives

`AqMiLaqCfgReq`

`SmMiLaqCfgReq`

Description

The layer manager configures the various elements of Diameter using the management – configuration procedure, which the layer manager initiates. The Diameter configuration request primitive, **`AqMiLaqCfgReq`**, can be called more than once. The **`AqMiLaqCfgReq`** primitives must be called before the bind primitives are called.

Table 6-11 lists the Diameter configuration request primitive types.

Table 6-11: Diameter Configuration Request Primitive Types

Name	Description
General	Passes parameters that apply to the entire Diameter software. It primarily reserves the static memory required by Diameter, Timer Resolution, and so on.
Upper SAP	Passes parameters that apply to the upper SAP of the Diameter software.
Lower SAP	Passes parameters that apply to the lower SAP of the Diameter software.
Protocol	Passes protocol-specific parameters.
Realm	Passes realm configuration parameters.
Peer	Passes peer configuration parameters.
AVP	Passes the AVP configuration which includes the application ID, number of AVPs along with the AVP properties.
Diameter Message	Passes the Diameter Message configuration which includes the application ID, number of Diameter messages and array of Diameter message properties.

To operate properly, the configuration request primitive types must be called in this order:

1. General
2. Protocol
3. Upper SAP
4. Lower SAP
5. Diameter Messages
6. AVP
7. Realm
8. Peer

The general configuration must be done before any other configuration is done.

The protocol configuration must be done once.

Diameter layer has one upper SAP towards each application. That means the number of upper SAP must be same as number of applications running over it. For each application, AVP and DM configuration must be done separately.

For application specific DM and AVP configuration, upper SAP configuration must be done before DM and AVP configuration. If the user wants to configure Diameter base protocol specific DM commands and AVP then the DM and AVP configuration must be done before the upper SAP configuration.

DM and AVP configuration must be done for the complete DM and AVP Dictionary. Single entry configuration is not supported. DM and AVP configuration can be either Diameter Base Protocol specific or application specific.

The lower SAP configuration is sent at least once for each lower SAP:

- If there is only one kind of transport channel, there is only one SAP;
- Else, for multiple transports — for example, SCTP and TUCL — there is one SAP for each transport channel.

Realm configuration must be done before peer configuration. It consists of configuration parameters of static realms.

Peer configuration consists of the information about the manually configured peers.

If any of the configurations is repeated, it is treated as a reconfiguration request. It is assumed that Layer Manager gives only the incremental values during reconfiguration. At the time of reconfiguration, user has to send the values of all the configurable parameters. If the user wants to configure only few parameters and not all then the user must send the existing values for the parameters which are unwanted to reconfigure. Diameter updates the reconfigurable parameters and returns the configuration confirm. Refer to each configuration for more information about the reconfigurable parameter

It is assumed that at the time of reconfiguration, only the incremental values are sent by LM. LM must not send the existing data.

Note: All the parameters are not configurable.

The system services primitives are called during the management–configuration procedure.

Note: The register timer (**`SRegTmr`**) system services primitive is called during the general configuration request procedure to register the Diameter timer activation (**`aqActvTmr`**) function.

The **`aqMngmt.u.cfg`** structure specifies the parameters that the configuration request primitive (**`AqMiLaqCfgReq`**) uses.

Figure 6-1 shows the product-specific configuration data flow.

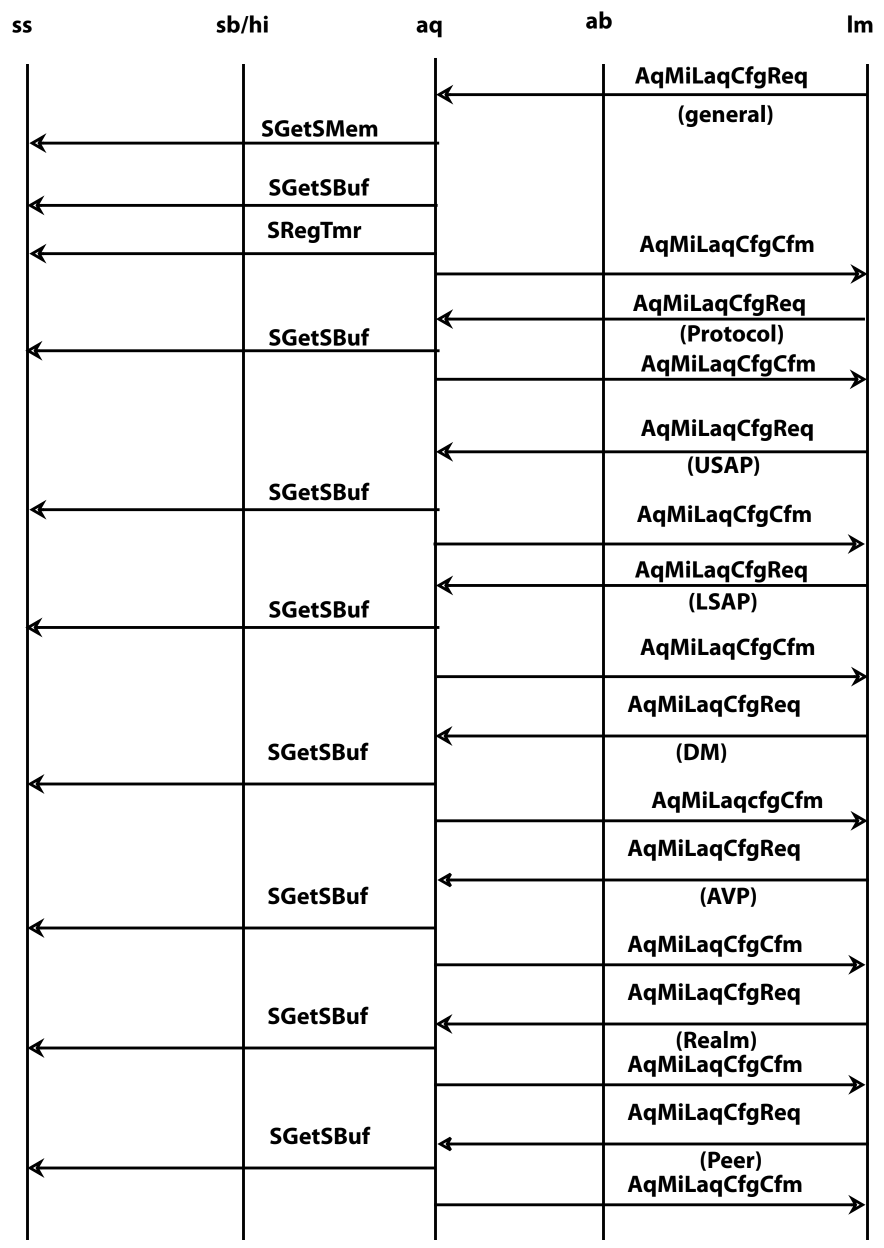


Figure 6-1: Management – Configuration Procedure

Synopsis

```
PUBLIC S16 AqMiLaqCfgReq (pst, cfg)
Pst         *pst;
AqMngmt     *cfg;
```

Parameters

pst

Refer to Section 1.8.1, "Post."

cfg

This parameter represents the management structure described in Section 6.2.2, "AqMngmt". The parameters specific to the configuration request are described in the subsequent sections.

In the **Header** of the configuration request, the values of the relevant fields are set, as shown in Table 6-12.

Table 6-12: Header Values of Configuration Request

Field	Allowable Values
elmId.elmnt	STAQGEN : General STAQUSAP : Upper SAP STAQLSAP : Lower SAP STAQPROT : Protocol STAQREALM : Realm STAQPEER : Peer STAQAVP : AVP STAQDM : Diameter Message

The configuration structure has the format:

```
typedef struct _aqCfg
{
   union
   {
        AqGenCfg    gen;         /* General Configuration */
        AqProtCfg   prot;        /* Protocol Configuration */
        AqDmCfg     dmMsg        /* Diameter Message Dictionary
                                    Configuration */
        AqAvpCfg    avp;         /* AVP Dictionary Configuration */
        AqUSapCfg   uSap;        /* Upper SAP Configuration */
        AqLSapCfg   lSap;        /* Lower SAP Configuration */
        AqRealmCfg  realm;       /* Realm Configuration */
        AqPeerCfg   peer;        /* Peer Configuration */
   } u;
} AqCfg;
```


6.3.1.1 General Configuration

The general configuration is used to configure the common parameters used by the Diameter layer. The data structure and the description of the individual parameters of the general configuration are:

```
typedef struct _aqGenCfg
{
   U16 nmbUSaps;          /* Maximum Number of Upper SAPs */
   U16 nmbLSaps;          /* Maximum Number of Lower SAPs */
   U32 sizePmq;           /* Size of Pending Message Queue */
   U32 firmwareRev;       /* Firmware Revision */
   U32 orgStateId;        /* Origin State ID */
   U8 nmbPeer;            /* Maximum Number of Peers */
   U8 nmbRealm;           /* Maximum Number of Realms */
   LaqStr hostId;         /* Host ID of the node */
   LaqStr localRealm;     /* Host Realm of the Node */
   LaqStr prodName;       /* Product Name */
   U8 nmbWorkerThrds;     /* Number of Worker threads */
   U16 timeRes;           /* Timer resolution */
   Status memUpperThr;    /* Upper threshold of the memory pool */
   Status memLowerThr;    /* Lower threshold of the memory pool */
   Pst lmPst;             /* Post structure for layer manager */
} AqGenCfg;
```

Table 6-14: Parameters of AqGenCfg

Parameter	Description/Reference	Reconfigurable?
`nmbUSaps`	Maximum number of upper SAPs. The allowable values: **`1–65535`**.	No
`nmbLSaps`	Maximum number of lower SAPs. The allowable values: **`1–2`**. **Note:** For Testing purpose, maximum number of lower SAPs allowed is 4. Lower SAP 0 - Real TUCL Lower SAP 1 - Real SCTP Lower SAP 2 - Dummy TUCL Lower SAP 3 - Dummy SCTP Dummy TUCL and Dummy SCTP are used for internal testing. However, in real scenario, maximum allowed values for lower SAP is 2.	No
`sizePmq`	Maximum size of pending message queue. Pending message queue are maintained for each peer to perform failover/failback. When an answer message is received, the corresponding pending request is removed from the peer.	Yes
`firmwareRev`	Firmware revision to be send in CER/CEA. This is used to inform a Diameter peer of the firmware revision of the issuing device. Refer RFC 3588 Section 5.3.4.	Yes
`orgStateId`	Origin-State ID. It is a monotonically increasing value that is advanced whenever a Diameter entity with loss of previous state. Refer RFC 3588 Section 8.16.	No
`nmbPeer`	Maximum number of peer connections that a node can have.	No
`nmbRealm`	Maximum number of realms that a node can support.	No
`hostId`	Host ID of the node.	No
`localRealm`	Realm name of the node.	No
`prodName`	Product Name is the vendor assigned name for the product. Refer RFC 3588 Section 5.3.7.	No

Table 6-14: Parameters of AqGenCfg

Parameter	Description/Reference	Reconfigurable?
`nmbWorkerThrds`	Maximum number of worker threads that can be spawned. Allowed values are: 0 to 63. Refer to Section 1.3, "Multi Thread Mode".	No
`memUpperThr`	Upper threshold of the memory pool. If the free dynamic memory goes below the `memUpperThr`, then an alarm is raised to the layer manager.	Yes
`memLowerThr`	Lower threshold of the memory pool. If the free dynamic memory goes below the `memLowerThr`, then Diameter layer stops accepting the new incoming connections until the memory threshold raises above the `memUpperThr`.	Yes
`timeRes`	Timer resolution, in ticks. The allowable values: `1-65535.` This indicates the number of system ticks elapsed between successive timer monitoring events; for example, if the system tick is one (1) millisecond and a timer event is desired after one (1) second, then the value of the `timeRes` must be 1000. (1 millisecond * 1000 = 1 second.)	No
`lmPst`	Diameter uses this post structure to communicate with the layer manager; for example, at the time of sending alarm and status indications to layer manager. Refer to Section 1.8.1 for pst structure.	Yes

6.3.1.1.1 LaqStr

The structure format and parameter description for LaqStr are:

```
typedef struct _laqStr
{
   U16 length;       /* Number of bytes present in buff array */
   Data buff[LAQ_MAX_STR_SIZE];  /* data */
}LaqStr;
```

Table 6-15: Parameters of LaqStr

Parameters	Description
`length`	This field specifies the size of the buff array.
`buff`	This field specifies the data. Possible values can be 0 to 255.

6.3.1.2 Upper SAP Configuration

Upper SAP configuration is used to configure the SAP related parameters. The data structure and description of the individual parameters of the upper SAP configuration are:

```
typedef struct _aqUSapCfg
{
   SpId          spId;             /* Upper Sap ID */
   Selector      selector;         /* Coupling */
   MemoryId      mem;              /* Memory pool */
   Priority      priority          /* Priority */
   Route         route;            /* Route */
   AqAppId       appId;            /* Application ID */
   AqVendorId    vendorId;         /* Vendor ID */
   Bool          suppVenId;        /* Flag specifies the presence of
                                      supported vendor ID */
   AqVendorId    suppVendorId;     /* Supported Vendor ID */
   AqAppType     appType;          /* Application Type */
} AqUSapCfg;
```

Table 6-16: Parameters of AqUSapCfg

Parameters	Description	Reconfigurable?
`spId`	Upper SAP ID. The allowable values: `0` - `(nmbUSaps - 1).`	No

Table 6-16: Parameters of AqUSapCfg

Parameters	Description	Reconfigurable?
`selector`	Coupling at the upper interface with the Diameter user. The allowable values: • `LAQ_SEL_LC`: Loose coupling. • `LAQ_SEL_TC_AB:` Tightly coupled Diameter user.	Yes
`mem`	Dynamic memory pool for the messages sent in the loosely coupled mode, at the upper interface. Refer to Section 1.8.7 for details.	Yes
`priority`	Priority of messages to be sent through this SAP. The allowable values are `0` – `3`, in which 0 is the highest and 3 the lowest.	Yes
`route`	Route. The allowable values: • `RTESPEC` for the conventional or distributed upper layer. • `RTEPROTO` for the fault-tolerant upper layer.	No
`vendorId`	Vendor ID supported by the upper SAP	No
`suppVendId`	This flag specifies the presence of the of supported vendor ID. Possible values are: • `TRUE`: Supported vendor ID is present. • `FALSE`: Supported vendor ID is not present.	No
`suppVendorId`	This field specifies the supported vendor ID associated with the SAP. Refer RFC 3588 Section 5.3.6.	No
`appId`	This field specifies the application ID supported by the SAP.	No
`appType`	Application type. Possible values can be: • `AQ_AUTH_TYPE` for authentication application. • `AQ_ACCT_TYPE` for accounting application. If the application supports both authentication and accounting then use `AQ_AUTH_TYPE | AQ_ACCT_TYPE.`	No

6.3.1.2.1 AqVendorId

`AqVendorId` is typedef to U32.

```
typedef U32 AqVendorId
```

6.3.1.2.2 AqAppId

`AqAppId` is typedef to U32.

```
typedef U32 AqAppId
```

6.3.1.2.3 AqAppType

`AqAppType` is typedef to U8.

```
typedef U8 AqAppType
```

6.3.1.3 Lower SAP Configuration

Lower SAP configuration is used to configure the parameters of lower SAP. The data structure and description of the individual parameters of the lower SAP configuration are:

```
typedef struct _aqLSapCfg
{
    SuId     suId;           /* Diameter Sap ID */
    SpId     spId;           /* Service Provider SAP ID */
    Selector selector;       /* Coupling */
    MemoryId mem;            /* Memory pool */
    Priority prior;          /* Priority */
    Route    route;          /* Route */
    ProcId   dstProcId;      /* Destination processor ID */
    Ent      dstEntId;       /* Destination entity ID */
    Inst     dstInstId;      /* Destination instance ID */
    U8       transpType;     /* Transport Type SCTP/TUCL*/
    U8       maxBndRetry;    /* Maximum number of bind retries
                                allowed */
    U8       maxSerRetry;    /* Maximum number of server retries
                                allowed */
    TmrCfg   tBndTmr;        /* Bind timer config must be same
                                for SCTP/TUCL */
    TmrCfg   tSerTmr;        /* Server timer config - same for all
                                the three server configs */
    AqServerCfg aqServerCfg; /* Server configuration parameters */
}AqLSapCfg;
```

Table 6-17: Parameters of AqLSapCfg

Parameters	Description/Reference	Reconfigurable?
`suId`	Diameter SAP ID. The allowable values: **`0`** – **`(nmbLSaps - 1)`**, in which the **`nmbLSaps`** is the value configured in the general configuration.	No

Table 6-17: Parameters of AqLSapCfg

Parameters	Description/Reference	Reconfigurable?
`spId`	Service provider SAP ID. The allowable values: **`0`** – **`(max service provider SAPs - 1).`**	No
`selector`	Coupling for the messages sent at the lower interface. The allowable values: • **`LAQ_SEL_LC:`** Loose coupling. • **`LAQ_SEL_TC_SCT`**: Tightly coupled SCT. • **`LAQ_SEL_TC_HIT`**: Tightly coupled HIT.	Yes
`mem`	Dynamic memory pool for messages sent at the lower interface. Refer to Section 1.8.7 for details.	Yes
`priority`	Priority of messages sent on this SAP. The allowable values: **`0`** to **`3`**, in which 0 is the highest and 3 the lowest.	Yes
`route`	Route. The allowable values: • **`RTESPEC`**: for the conventional or distributed lower layer. • **`RTEPROTO`**: for the fault-tolerant lower layer.	No
`dstProcId`	Processor ID of the destination; that is, the service provider. The allowable values: **`0`** – **`255.`**	No
`dstEntId`	Destination entity ID.	No
`dstInstId`	Destination instance ID.	No
`tBndTmr`	Bind timer for binding a lower SAP. This value specifies the timer value for which the Diameter layer waits for bind confirm.	Yes
`tSerTmr`	Server timer for opening the server connection. This value specifies the timer value for which the Diameter layer waits for server connection confirm.	No
`maxSerRetry`	Maximum number of server retries allowed. This value specifies the maximum number of times Diameter layer tries to open the server connection. Diameter layer uses **`XxYyHitServerOpenReq()`** to create the server socket at HIT interface. Similarly, it uses **`XxYySctEndPOpenReq()`** at SCT interface.	No
`maxBndRetry`	Maximum number of bind retries allowed. This value specifies the maximum number of times Diameter layer tries to bind with lower layer.	Yes

Table 6-17: Parameters of AqLSapCfg

Parameters	**Description/Reference**	**Reconfigurable?**
`transpType`	Specifies the Transport Type. Possible values are: • **`LAQ_TRANS_TYPE_TCP_TLS`** for TCP protocol supporting TLS. • **`LAQ_TRANS_TYPE_TCP_IPSEC`** for TCP protocol supporting IPSec. • **`LAQ_TRANS_TYPE_SCTP_IPSEC`** for SCTP protocol supporting IPSec. It can be set to a combination of the above mentioned allowable values. For example, if both TLS and IPSEC are supported by TCP then the transpType is set to **`LAQ_TRANS_TYPE_TCP_TLS | LAQ_TRANS_TYPE_TCP_IPSEC`** **Note**: TLS is not supported along with Trillium SCTP.	No
`aqServerCfg`	This structure contains the Server configuration information for transport type. **Note:** If the node is configured as "**`LAQ_CLIENT`**" then there is no need to do server configuration.	No

6.3.1.3.1 AqServerCfg

This union contains the TCP/IPSec, TCP/TLS, or SCTP/IpSec server configuration based on the protocol and security the node is supporting. The union format and parameter description for **`AqServerCfg`** is:

```
typedef union _aqServerCfg
{
   AqTcpCfg  aqTcpCfg;                   /* TCP Server */
   #ifdef AQ_SCTP
    AqSctpIpSecSerCfg  sctpIpSecSerCfg;  /* SCTP + IpSec Server*/
   #endif
}AqServerCfg;
```

Table 6-18: Parameters of AqServerCfg

Parameters	Description/Reference	Reconfigurable?
`aqTcpCfg`	This structure specifies the TCP Server configuration.	No
`sctpIpSecSerCfg`	This field specifies the SCTP Server configuration.	No

6.3.1.3.2 AqTcpCfg

This structure contains the TCP/TLS and TCP/IPSec server configuration. The structure format and parameter description for **`AqTcpCfg`** is:

```
typedef struct _aqServerCfg
{
   AqTcpSerCfg  tcpIpsecSerCfg;      /* TCP + IPSec Server */
   AqTcpSerCfg  tcpTlsSerCfg;        /* TCP + TLS Server */
}AqTcpCfg;
```

Table 6-19: Parameters of AqTcpCfg

Parameters	Description/Reference	Reconfigurable?
`tcpIpSecSerCfg`	This structure specifies the TCP + IpSec Server configuration.	No
`tcpTlsSerCfg`	This structure specifies the TCP + TLS Server configuration.	No

6.3.1.3.3 AqTcpSerCfg

This structure contains the configured IP address and port for the TCP server. The structure format and parameter description for **`AqTcpSerCfg`** is:

```
typedef struct _aqTcpSerCfg
{
    U32          tcpPort;         /* TCP Port */
    CmNetAddr    tcpIpAddr;       /* IP Address */
    CmTptParam   transpParam;     /* Transport socket and TLS
                                     parameter */
    CmIcmpFilter icmpFilter;      /* ICMP Filter */
} AqTcpSerCfg;
```

Table 6-20: Parameters of AqTcpSerCfg

Parameters	Description/Reference	Reconfigurable?
`tcpPort`	Listening Port of the TCP server.	No
`tcpIpAddr`	IP address of the TCP server. Refer to Section 7.2, "CmNetAddr Structure".	No
`transpParam`	Transport socket and TLS parameter for server configuration. Refer to Section 7.4, "CmTptParam Structure".	No
`icmpFilter`	Specifies the parameters for filtering received ICMP messages. Refer to Section 7.7, "CmIcmpFilter Structure".	No

6.3.1.3.4 AqSctpIpSecSerCfg

This structure contains the IP address and port to be configured for SCTP server supporting IPsec. The structure format and parameter description for **`AqSctpIpSecSerCfg`** is:

```
typedef struct _aqSctpIpSecSerCfg
{
   SctNetAddrLst  sctpIpAddr;  /* IP Address list */
   SctPort        sctpPort;    /* Sctp Port */
} AqSctpIpSecSerCfg;
```

Table 6-21: Parameters of AqSctpIpSecSerCfg

Parameters	Description/Reference	Reconfigurable?
`sctpPort`	Listening port of the STCP server supporting IPSec. Refer to Section 7.3, "SctNetAddrLst Structure".	No
`sctpIpAddr`	List of IP address of the STCP server supporting IPSec.	No

6.3.1.4 Protocol Configuration

Protocol configuration is used to configure the protocol related data. The data structure and description of the individual parameters of the Protocol Configuration is:

```
typedef struct _aqProtCfg
{
   TmrCfg tWatchDogTmr;   /* Watch Dog Timer */
   TmrCfg tBurstBrkTmr;   /* Burst Break Timer */
   TmrCfg tPeerStateTmr;  /* Peer State Timer */
   TmrCfg tReconnTmr;     /* Reconnect Timer */
   Bool   failover;       /* Failover - ON/OFF  */
   U8     secType;        /* Supported Security Type */
   U8     maxConnRetry;   /* Max number of Reconnect try */
   U8     localPolicy;    /* Local policy */
   U8     nmbBurstCnt;    /* Number of pending messages to be
                             processed before posting the message
                             to self */
   U8     nodeType;       /* Specifies the node type - client
                             or server or both */
   CmTptParam transParam; /* Transport Socket and TLS Parameter */
   #ifdef AQ_PEER_DISCOVERY
   union _aqPeerDsc
   {
      AqSlpPeerDsc  aqSlpPeerDsc; /* Peer Discovery Info */
      /* NAPTR */
   } AqPeerDsc;
   U8     peerDscType;    /* Supported Peer Discovery Type */
   TmrCfg tPeerDscTmr;    /* Peer Discovery Timer */
   #endif
   U8     nmbOptAvp;   /* Number of Optional AVP present in CER/CEA
                          which are not defined in RFC 3588*/
   AqAvp  optAvpValue[LAQ_MAX_OPT_AVP];  /* Holds the values for
                                            Optional AVP present in
                                            CER which are not
                                            specified in RFC 3588*/
   AqVendorId  venId;  /* Vendor Id */
   U8       relaySupp; /* Flag to specify the Agent support */
   }AqProtCfg;
```

Table 6-22: Parameters of AqProtCfg

Parameters	Description/Reference	Reconfigurable?
`tWatchDogTmr`	Watch dog timer. This specifies the time interval after which the device watchdog request must be sent. Refer RFC 3539 Section 3.4.1, "Tw timer".	Yes

Table 6-22: Parameters of AqProtCfg

Parameters	Description/Reference	Reconfigurable?
`tBurstBrkTmr`	Burst break timer. This specifies the time interval after which the Failover/Failback has to be resumed.	Yes
`tPeerStateTmr`	Peer State Timer. This specifies the time interval after which the time out event is generated for each state. Refer RFC 3588 Section 5.6.2, "Timeout event".	Yes
`tReconnTmr`	Reconnect Timer. It specifies the time interval for retrying the connection. Refer RFC 3588 Section 2.1, "Tc timer".	Yes
`tPeerDscTmr`	Peer Discovery Timer. It specifies the time interval after which the Peer Discovery is to be started.	Yes
`failover`	Boolean flag which specifies whether the failover is supported or not.	Yes
`peerDscType`	Peer Discovery Type specify the type of service to be used for Peer Discovery. Possible values for the same are: • **`LAQ_PEER_DSC_NONE`** - If the peer discovery is not supported. • **`LAQ_PEER_DSC_SLP`** - SLPv2 is used for peer discovery. • **`LAQ_PEER_DSC_NAPTR`** - NAPTR is used for peer discovery. **Note:** **`NAPTR`** is not supported in this release.	Yes
`secType`	Security Type specifies the type of Security to be used. Possible values are: • **`LAQ_SEC_TLS`** - If TLS is used for security. • **`LAQ_NO_INBAND_SEC`** - If IpSec is used for security.	Yes
`maxConnRetry`	The count specifies the maximum number of times reconnect must be tried.	Yes

Table 6-22: Parameters of AqProtCfg

Parameters	Description/Reference	Reconfigurable?
`localPolicy`	Local policy specifies whether to accept or deny the incoming connections from unknown peers. Possible values are: • **`LAQ_INCOMING_CONN_ACCEPT`** - For accepting the connection request from unknown peers. • **`LAQ_INCOMING_CONN_DENY`** - For rejecting the connection request from unknown peers.	Yes
`nodeType`	This field specifies the type of the Diameter node. Possible values are: • **`LAQ_SERVER`**: Node is configured as server. • **`LAQ_CLIENT`**: Node is configured as client. • **`LAQ_BOTH`**: Node can act as client as well as server.	No
`nmbBurstCnt`	Number of pending messages to be processed before posting an event to self for failover and failback.	Yes
`transParam`	This field specifies the transport socket and TLS parameters.	No
`AqPeerDsc`	This structure contains the information to be configured for peer discovery.	No
`nmbOptAvp`	This field specifies the actual number of optional AVPs present in **`AqOptAvp`**.	No
`optAvpValue`	This array contains the AVP values for those optional AVPs which are not present in CER as per RFC 3588. **Note**: Refer Section 4.5.2.2 of *AQU Interface Service Definition* for AqAvp structure.	No
`venId`	This field contains the vendor ID.	Yes
`relaySupp`	This field specifies whether the node is acting as agent or not.	No

6.3.1.4.1 AqSlpPeerDsc

This structure contains the configuration parameters to be configured for SLP Daemon. The structure format and parameter description for **`AqSlpPeerDsc`** is:

```
typedef struct _aqSlpPeerDsc
{
   LaqStr     peerDscSrvc;        /* Peer Discovery Service Type */
   U8         proto;              /* Protocol */
   U8         sec;                /* Security */
   LaqStr     scope;              /* Scope of Peer Discovery */
   AqAddr     aqPeerDscSelfAddr;  /* Peer Discovery Self Address */
}AqSlpPeerDsc;
```

Table 6-23: Parameters of AqSlpPeerDsc

Parameters	Description/Reference	Reconfigurable?
`peerDscSrvc`	The Service type which has to be discovered dynamically. For example, it can be **service:diameter** for discovering Diameter service.	Yes
`proto`	This field specifies the protocol to be supported by the new discovered peer. Possible values are: • **`LAQ_PROT_TCP`** - TCP is used to for establishing connection with newly discovered peer. • **`LAQ_PROT_SCTP`** - SCTP is used for establishing connection with newly discovered peer. • **`LAQ_PROT_NONE`** - Invalid value.	Yes
`sec`	This field specifies the security to be supported by the new discovered peer. Possible values are: • **`LAQ_SEC_TLS`** - TLS security is used for establishing connection with newly discovered peer. • **`LAQ_NO_INBAND_SEC`** - IPSec is used for establishing connection with newly discovered peer.	Yes
`scope`	This field specifies the scope of the Peer Discovery. Services are grouped together using scopes. Scope can indicate a location, administrative grouping or network grouping. Its the domain within which the service has to be discovered.	Yes

Parameters	Description/Reference	Reconfigurable?
aqPeerDscSelfAddr	The structure contains the information about the self IP address which is used during Peer Discovery.	No

6.3.1.5 Peer Configuration

Peer configuration request is sent to configure all the static peers. The data structure and description of the individual parameters of the Peer Configuration are:

```
typedef struct _aqPeerCfg
{
   U8         nmbPeers;       /* Number of Peers */
   AqPeerInfo *peerInfo;      /* Peer Info */
}AqPeerCfg;
```

Table 6-24: Parameters of AqPeerCfg

Parameters	Description/Reference	Reconfigurable?
nmbPeer	Number of Peers to be configured.	Yes
peerInfo	Pointer to structure containing the Configuration information needed for each Peer entry.	Yes

6.3.1.5.1 AqPeerInfo

This structure contains the configuration parameters of the peer. The structure format and parameter description is as below:

```
typedef struct _aqPeerInfo
{
   LaqStr      hostId ;     /* Host ID */
   LaqStr      realmName;   /* Realm Name */
   U8          security;    /* Security */
   U8          protocol;    /* Protocol */
   AqAddr      aqSelfAddr;  /* Self Address */
   AqAddr      aqPeerAddr;  /* Peer Address */
   SpId        sapId;       /* Lower Sap Id */
}AqPeerInfo;
```

Table 6-25:

Table 6-26: Parameters of AqPeerInfo

Parameters	Description/Reference	Reconfigurable?
`hostId`	Host ID of the configured peer.	No
`realmName`	This field specifies the realm to which the peer belongs.	No
`security`	This field specifies the type of security the Peer is supporting. Possible values are: • **`LAQ_SEC_TLS`** - TLS is supported by the peer. • **`LAQ_NO_INBAND_SEC`** - IpSec is supported by the peer.	No
`protocol`	This field specifies the type of protocol the Peer is supporting. Possible values are: • **`LAQ_PROT_TCP`** - TCP is supported by the peer. • **`LAQ_PROT_SCTP`** - SCTP is supported by the peer.	No
`aqSelfAddr`	This union contains the Self IP Address of the node.	No
`aqPeerAddr`	This union contains the Peer Address of the Destination node.	No
`sapId`	This field specifies the lower SAP ID.	No

NOTE 1: If the self node's security is configured as LAQ_NO_INBAND_SEC (profCfg) and if the security of the peer is configured as LAQ_SEC_TLS, then LAQ_REASON_INV_SEC_CFG error will be returned. If the security of the peer is configured as LAQ_NO_INBAND_SEC (protCfg), then no error will be returned

NOTE 2: If the self node's security is configured as LAQ_SEC_TLS (profCfg) and if the security of the peer is configured as either LAQ_NO_INBAND_SEC or LAQ_SEC_TLS, then no error will be returned.

6.3.1.5.1.1 AqAddr

This union contains the TUCL address and SCTP address to be used by a particular peer. Based on the "protocol", appropriate structure is used. The union format and description of **`AqAddr`** is:

```
typedef  union _aqAddr
{
   AqTuclAddr  tuclAddr;  /* TUCL address */
   AqSctpAddr  sctpAddr;  /* SCTP Address */
}AqAddr;
```

Table 6-27: Parameters of AqAddr

Parameters	Description/Reference	Reconfigurable?
`tuclAddr`	This structure contains the IP address and port for TUCL transport type.	No
`sctpAddr`	This structure contains the list of IP addresses and ports for SCTP transport type.	No

6.3.1.5.1.2 AqTuclAddr

This structure contains the IP Address and Port to be used at the time of connection establishment. The structure format and description of **`AqTuclAddr`** is:

```
typedef  struct _aqTuclAddr
{
   CmNetAddr   IpAddr;  /* IP Address */
   U32         port;    /* Port Number */
}AqTuclAddr;
```

Table 6-28: Parameters of AqTuclAddr

Parameters	Description/Reference	Reconfigurable?
`selfIpAddr`	Self IP address of the peer which is used at the time of connection establishment. Refer to Section 7.2, "CmNetAddr Structure".	No
`port`	Port.	No

6.3.1.5.1.3 AqSctpAddr

This structure contains the list IP Address and Port to be used at the time of connection establishment. The structure format and description of **`AqSctpAddr`** is:

```
typedef  struct _aqSctpAddr
{
   SctNetAddrLst   IpAddr;     /* IP Address */
   SctPort         sctPort;    /* Port Number */
}AqSctpAddr;
```

Table 6-29: Parameters of AqSctpAddr

Parameters	Description/Reference	Reconfigurable?
`selfIpAddr`	Self IP address of the peer which is used at the time of connection establishment. Refer to Section 7.3, "SctNetAddrLst Structure".	No
`port`	Port.	No

6.3.1.6 Realm Configuration

This configuration request is used to send the information needed for realm table. The data structure and description of the individual parameters of the Realm Configuration are:

```
typedef struct _aqRealmCfg
{
   U8           nmbRealm;          /* Number of Realms */
   AqRealmInfo  *realmInfo;        /* Realm Config */
}AqRealmCfg;
```

Table 6-30: Parameters of AqRealmCfg

Parameters	Description/Reference	Reconfigurable?
`nmbRealm`	Number of realms to be configured.	Yes
`realmInfo`	Pointer to structure containing the configuration information needed for each realm entry.	Yes

6.3.1.6.1 AqRealmInfo

This structure contains the configuration parameters for each realm. The structure format and description of the parameters are:

```
typedef struct _aqRealmInfo
{
   LaqStr        realmName;       /* Realm Name */
   U8            nmbApp;          /* Number of application */
   LaqAppHostMap *aqAppHostMap;   /* App ID to Host ID mapping */
   U8            localAction;     /* Local Action */
}AqRealmInfo;
```

Table 6-31: Parameters of AqRealmInfo

Parameters	Description/Reference	Reconfigurable?
realmName	Realm name of the peer.	No
nmbApp	Number of applications the node is supporting.	Yes
aqAppHostMap	Pointer to structure containing the mapping of application ID to host ID.	Yes
localAction	Local action of the realm. Possible values are: • **LAQ_ACTION_LOCAL** - If the Diameter message must be processed locally. • **LAQ_ACTION_PROXY** - If the Diameter message can be proxied to next hop server. Local server may apply its local policies. • **LAQ_ACTION_RELAY** - If the Diameter message can be routed to next hop server without modifying non-routing AVPs. • **LAQ_ACTION_REDIRECT** - If the Diameter message has to be redirected to redirect server. **Note:** In this release, **LAQ_LOCAL_ACTION** and **LAQ_ACTION_RELAY** are supported. However, applications can be implemented over Diameter base protocol to support other local actions.	Yes

6.3.1.6.1.1 AqAppHostMap

The structure format and parameter description for AqAppHostMap is

```
typedef struct _laqAppHostMap
{
   AqAppId   appId;                /* Application ID */
   U8        nmbHost;              /* Number of Host */
   LaqStr    *hostId;              /* Host ID */
}LaqAppHostMap;
```

Table 6-32: Parameters of LaqAppHostMap

Parameters	Description/Reference	Reconfigurable?
`appId`	Application ID supported by the realms.	No
`nmbHostId`	Number of host IDs present in the array.	No
`hostId`	Pointer to structure containing the information of the host ID supporting the application ID.	No

6.3.1.7 AVP Configuration

AVP Configuration is needed for configuring the AVP dictionary specific to application or Diameter base protocol. If LM sends the same AVP more than once, then following actions is taken:

1. If the mandatory flag "M" is set for the previous entry then configuration is rejected and negative confirmation is sent to LM.
2. If the previous AVP entry is optional then the previous AVP entry is over written with the new AVP entry.

The data structure and description of the individual parameters of the AVP Configuration are:

```
typedef struct _aqAvpCfg
{
   AqAppId      appId;          /* Application ID */
   U16          nmbAvps;        /* Number of AVPs */
   LaqAvpEntry  *avpEntry;      /* Array of AVP Entry */
} AqAvpCfg;
```

Table 6-33: Parameters of AqAvpCfg

Parameters	Description/Reference	Reconfigurable?
`appId`	Application ID which specifies the application for which the AVP Dictionary is being configured.	No
`nmbAvps`	Number of AVPs to be configured.	No
`avpEntry`	Pointer to structure containing the AVP information to be configured in the AVP Dictionary.	No

6.3.1.7.1 LaqAvpEntry

The structure format and parameter description for LaqAvpEntry is:

```
typedef struct _laqAvpEntry
{
   U32   avpCode;              /* AVP Code */
   U8    dataType;             /* Data type of AVP */
   U8    flags;                /* MUST flags of AVP */
   U8    mnFlags;              /* MUST NOT flags of AVP */
   LaqGroupedAvp groupedAvp;   /* List of AVPs in Grouped AVP */
} LaqAvpEntry;
```

Table 6-34: Parameters of LaqAvpEntry

Parameters	Description/Reference	Reconfigurable?
`avpCode`	This field specifies the AVP code of the AVP to be configured.	No
`dataType`	This field specifies the data type of the AVP to be configured. Possible values are: • **`AQ_UNSIGNED_32`** - For unsigned 32 datatype. • **`AQ_SIGNED_32`** - For signed 32 datatype. • **`AQ_UNSIGNED_64`** - For unsigned 64 datatype. • **`AQ_SIGNED_64`** - For signed 64 datatype. • **`AQ_ENUMERATED`** - For enumerated datatype. • **`AQ_OCTET_STIRNG`** - For octet string datatype. • **`AQ_DIAMETER_IDENTITY`** - For Diameter identity datatype. • **`AQ_GROUPED`** - For grouped datatype • **`AQ_TIME`** - For time datatype. • **`AQ_ADDRESS`** - For address datatype. • **`AQ_FLOAT_32`** - For float 32 datatype. • **`AQ_FLOAT_64`** - For float 64 datatype. • **`AQ_UTF8STRING`** - For UTF8 String datatype. • **`AQ_DIAMETER_URI`** - For Diameter URI datatype. • **`AQ_IPFILTER_RULE`** - For IPFilter Rule datatype. • **`AQ_QOSFILTER_RULE`** - For QOSFilter Rule datatype. (Refer RFC3588 Section 4.2. and Section 4.3).	No

Parameters	Description/Reference	Reconfigurable?
flags	This field specifies the MUST flags (that is, which must be set in AVP) of the AVP to be configured. Possible values are: • **AQ_AVP_FLAG_P** - If "P" bit is MUST. • **AQ_AVP_FLAG_M** - If "M" bit is MUST. • **AQ_AVP_FLAG_V** - If "V" bit is MUST. Refer RFC3588 Section 4.1.	No
mnFlags	This field specifies the MUST NOT flags (that is, which must not be set in AVP) of the AVP to be configured. Possible values are: • **AQ_AVP_FLAG_P** - If "P" bit is MUST NOT. • **AQ_AVP_FLAG_M** - If "M" bit is MUST NOT. • **AQ_AVP_FLAG_V** - If "V" bit is MUST NOT. Refer RFC3588 Section 4.1.	No
groupedAvp	This field specifies the pointer to list of AVPs present in the grouped AVP.	No

6.3.1.7.2 LaqGroupedAvp

The structure format and parameter description for LaqGroupedAvp is:

```
typedef struct _lAqGroupedAvp
{
   U8          numGrpAvp;     /* Number of AVPs in Grouped Avp */
   AvpMult     avpMult[LAQ_MAX_AVP_MULTIPLICITY];
   LaqAvpEntry *avpEntry;     /* Array of Avps in grouped AVP */
} LaqGroupedAvp;
```

Table 6-35: Parameters of LaqGroupedAvp

Parameters	Description/Reference	Reconfigurable?
numGrpAvp	This field specifies the number of AVPs present in the grouped AVP.	No
avpMult	This structure contains the minimum and maximum occurrence of the AVP entries present within the grouped AVP.	No
avpEntry	This field specifies the AVP entry present in the grouped AVP.	No

6.3.1.7.3 AvpMult

The structure format and parameter description for AvpMult is:

```
typedef struct _avpMult
{
   AqOcc     minOcc;
   AqOcc     maxOcc;
} AvpMult;
```

Table 6-36: Parameters of AvpMult

Parameters	Description/Reference	Reconfigurable?
`minOcc`	This field specifies the minimum number of occurrence allowed for an AVP Entry within the grouped AVP.	No
`maxOcc`	This field specifies the maximum number of occurrence allowed for an AVP Entry within the grouped AVP.	No

6.3.1.8 DM Configuration

DM Configuration is needed for configuring the DM dictionary specific to application or Diameter base protocol. If LM sends the same Diameter message more than once then configuration is rejected and negative confirmation is sent to LM.

The data structure and description of the individual parameters of the DM Configuration are

```
typedef struct _aqDmMsgCfg
{
   AqAppId   appId;              /* Application ID */
   U8        nmbCmd;             /* Number of commands */
   AqDmEntry *dmEntry;           /* DM Entry */
} AqDmMsgCfg;
```

Table 6-37: Parameters of AqDmMsgCfg

Parameters	Description/Reference	Reconfigurable?
`appId`	Application ID which specifies the application for which the DM Dictionary is to be configured.	No
`nmbCmd`	Number of Diameter Messages to be configured.	No
`dmEntry`	Pointer to structure containing the DM information to be configured in the DM Dictionary.	No

6.3.1.8.1 AqDmEntry

The structure format and parameter description for **AqDmEntry** is:

```
typedef struct _aqDmEntry
{
   U32   dmCode;          /* Command Code */
   U8    cmdFlags;        /* Message flags */
   U16   nmbAvpProp;      /* Number of AVPs inside DM */
   AqAvpProperties avpProperties[MAX_NUM_OF_AVP_PROP];
                          /* AVP Properties */
  } AqDmEntry;
```

Table 6-38: Parameters of `AqDmEntry`

Parameters	Description/Reference	Reconfigurable?
`dmCode`	This field specifies the command code of the Diameter Message to be configured.	No
`cmdFlags`	This field specifies the command flags of the Diameter Message to be configured.	No
`nmbAvpProp`	Number of AVPs to present in the Diameter Message.	No
`avpProperties`	This structure contains the information related to the AVP present in the Diameter Message.	No

6.3.1.8.1.1 AqAvpProperties

The structure format and parameter description for AqAvpProperties is:

```
typedef struct _aqAvpProperties
 {
   U32     avpCode;      /* AVP Code */
   U8      properties;   /* Properties of AVP */
   U8      posFlag;      /* Position Flag */
   U8      position;     /* Position of AVP if fixed type */
   AqOcc   minOcc;       /* Minimum num of occurrences  */
   AqOcc   maxOcc;       /* Maximum num of occurrences */
} AqAvpProperties;
```

Table 6-39: Parameters of AqAvpProperties

Parameters	Description/Reference	Reconfigurable?
avpCode	This field specifies the AVP code of the AVP present in the Diameter message.	No
posFlag	This flag specifies if the position of the AVP is valid or not. Possible values can be: • **TRUE**: Position of the AVP is valid. • **FALSE**: Position of the AVP is not valid.	No
position	This field specifies the position of the AVP in the Diameter message.	No
minOcc	This field specifies the minimum number of occurrence allowed for the AVP in the Diameter message.	No
maxOcc	This field specifies the minimum number of occurrence allowed for the AVP in the Diameter message.	No
properties	This field specifies whether the AVP is mandatory, optional or fixed in the Diameter message. Possible values are: • **AQ_AVP_MANDATORY**: For mandatory AVP. • **AQ_AVP_OPTIONAL**: For optional AVP. • **AQ_AVP_FIXED**: For fixed AVP.	No

6.3.1.8.1.2 AqOcc

`AqOcc` is typedef to U16.

```
typedef U16 AqOcc
```

6.3.2 Configuration Confirm

Name	Configuration confirm. The **`AqMiLaqCfgCfm`** is invoked in Diameter. The **`SmMiLaqCfgCfm`** processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes
Response	N/A

Primitives

`AqMiLaqCfgCfm`

`SmMiLaqCfgCfm`

Description

Diameter issues this primitive in response to the **`AqMiLaqCfgReq`** and indicates whether the configuration was successful.

Figure 6-2 shows the data flow for the Diameter configuration.

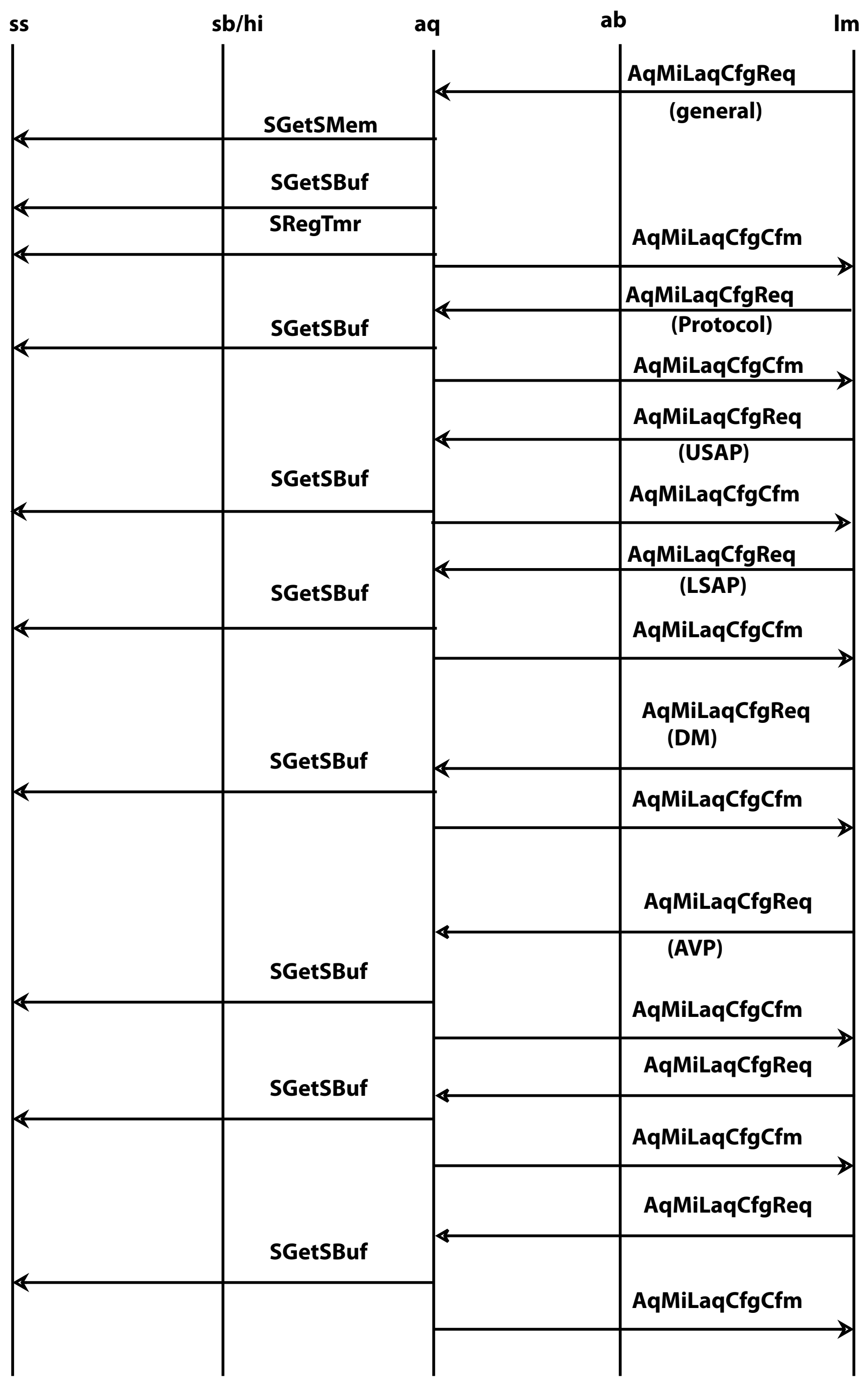


Figure 6-2: Management – Configuration procedure

Synopsis

```
PUBLIC S16 AqMiLaqCfgCfm (pst, cfm)
Pst       *pst;
AqMngmt   *cfm;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`cfm`

This parameter represents a pointer to the management structure described in Section 6.2.2. The configuration confirm contains only the header and the common status.

In the **`Header`** of the configuration confirm, the values of the relevant fields are set, as shown in Table 6-40.

Table 6-40: Header Values of Configuration Confirm

Field	Allowable Values

`elmId.elmnt`	**`STAQGEN : General`**
	`STAQUSAP : Upper SAP`
	`STAQLSAP : Lower SAP`
	`STAQPROT : Protocol`
	`STAQREALM : Realm`
	`STAQPEER : Peer`
	`STAQAVP : AVP`
	`STAQDM : DM`

As described in Section 1.8.6, "Common Status," the status information is returned in the **`CmStatus`** data structure. For the configuration confirm primitive, the values of the **`CmStatus`** are:

`status`

Name	Description
`LCM_PRIM_OK`	Successful request.
`LCM_PRIM_NOK`	Unsuccessful request.

`reason`

The content of this field is set to **`LCM_REASON_NOT_APPL`** when the **`status`** is set to **`LCM_PRIM_OK`**. When the **`status`** field is set to **`LCM_PRIM_NOK`**, the content of the reason field can be set to any of these values, indicating the reason of the failure of the primitive. Table 6-41 lists the invalid values of the configuration.

Table 6-41: Invalid Values of Configuration

Value	Description
`LCM_REASON_MEM_NOAVAIL`	Memory is not available.
`LCM_REASON_INVALID_ELMNT`	Invalid value of the **`elmnt`** field in the header.
`LCM_REASON_REGTMR_FAIL`	Timer registration failed.
`LCM_REASON_GENCFG_NOT_DONE`	General configuration is not done.
`LAQ_REASON_HASHINIT_FAIL`	Hashing failed.
`LCM_REASON_INVALID_PAR_VAL`	Invalid parameter values.
`LAQ_REASON_PROTCFG_NOT_DONE`	Protocol configuration is not done.
`LAQ_REASON_LSAPCFG_NOT_DONE`	Lower SAP configuration is not done.
`LAQ_REASON_USAPCFG_NOT_DONE`	Upper SAP configuration is not done.
`LAQ_REASON_EXCEED_CONF_VAL`	The number exceeds the value in the general configuration.

6.3.3 Statistics Request

Name	Statistics Request. The **`SmMiLaqStsReq`** is in the layer manager, and the **`AqMiLaqStsReq`** processes this request in Diameter.
Direction	Layer manager to Diameter
Supplied	Yes
Diameter Response	Yes

Primitives

`AqMiLaqStsReq`

`SmMiLaqStsReq`

Description

The layer manager gathers statistics information about the Diameter elements by using the management – statistics procedure, initiated by the layer manager. The statistics request primitive, **`AqMiLaqStsReq`**, can be called more than once and at any time after the management – configuration procedure. The statistics values are returned by the **`AqMiLaqStsCfm`** primitive. The statistics counters can be reset using the **`action`** parameter. If the statistics counters are not explicitly reset, they can be reset when the count exceeds the size of the data structure used to store the statistics (currently 2^{32} bytes).

These Statistics request primitive types are further divided into:

- Peer Node statistics - this includes:
  - Total number of messages received and sent.
  - Count of each kind of messages transmitted or received from the peer node.
- General statistics - this includes:
  - Total number of bytes sent/received.
  - Total number of messages sent/received.
  - Number of Erroneous messages.
- Upper SAP statistics - this includes:
  - Total number of requests sent/received by a specific SAP.
  - Total number of responses sent/received by a specific SAP.
- Lower SAP statistics - this includes:
  - Total number of bytes sent/received by a specific SAP.
  - Total number of messages sent/received by a specific SAP.

Figure 6-3 shows the management – Statistics procedure.

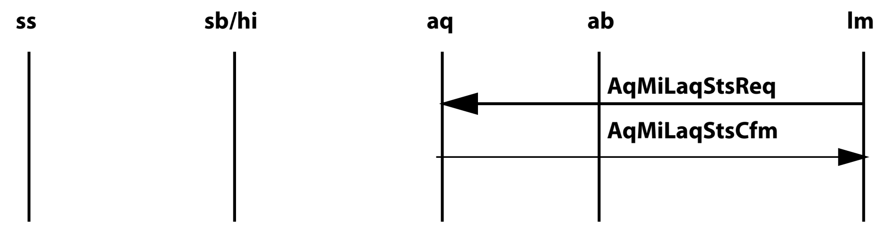


Figure 6-3: Management – Statistics procedure

Synopsis

```
PUBLIC S16 AqMiLaqStsReq (pst, action, sts)
Pst     *pst;
Action  action;
AqMngmt *sts;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`action`

Action.

Value	Description
`LAQ_ZEROSTS`	Resets the counters to zero after reporting.
`LAQ_NOZEROSTS`	Does not reset counters.

`sts`

Pointer to the management structure described in Section 6.2.2, "AqMngmt."

In the **`Header`** of the statistics request, the values of the relevant fields are set as:

Field	Allowable Values

`elmId.elmnt`	`STAQGEN     : General`
	`STAQPEER    : Statistics for specific Peer/peers`
	`STAQALLPEER : Statisics for all the peers`
	`STAQUSAP    : Upper SAP`
	`STAQLSAP    : Lower SAP`

6.3.4 Statistics Confirm

Name	Statistics Confirm. The **`AqMiLaqStsCfm`** is invoked in Diameter and **`SmMiLaqStsCfm`** processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes
Response	NA

Primitives

`AqMiLaqStsCfm`

`SmMiLaqStsCfm`

Description

This primitive is used to convey the statistics as requested by the layer manager in the **`AqMiLaqStsReq`** request.

Figure 6-4 shows how the confirm primitive is processed in the layer manager.

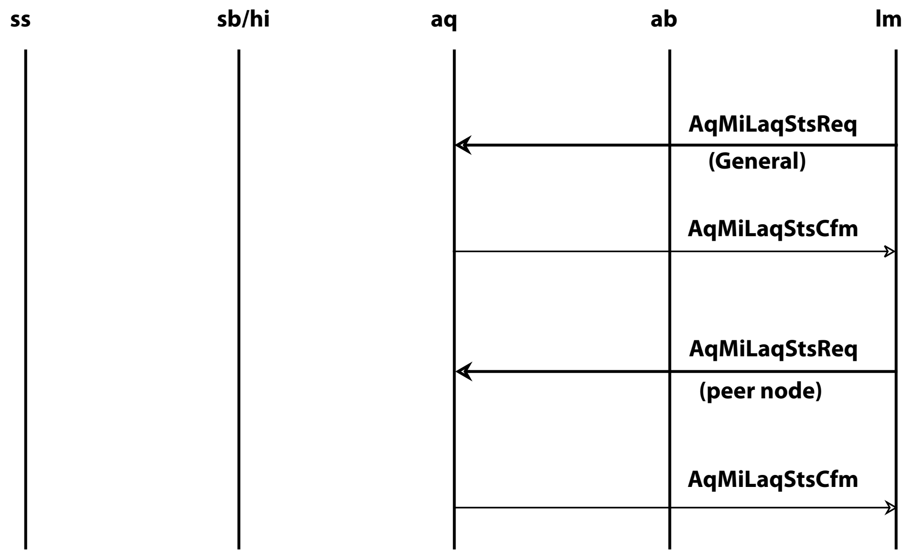


Figure 6-4: Management – Statistics procedure

Synopsis

```
PUBLIC S16 AqMiLaqStsCfm (pst, cfm)
Pst      *pst;
AqMngmt  *cfm
```

Parameters

pst

Refer to Section 1.8.1, "Post" for the description.

cfm

This parameter represents a pointer to the management structure described in Section 6.2.2. The statistics confirm-specific parameters are described in the subsequent sections.

In the **Header** of the statistics request, the values of the relevant fields are set as:

Table 6-42: Header Values of Statistics Confirm

Field	Allowable Values
elmId.elmnt	**STAQGEN : General** **STAQPEER : Statistics for specific Peer/peers** **STAQALLPEER : Statisics for all the peers** **STAQUSAP : Upper SAP** **STAQLSAP : Lower SAP**

As described in Section 1.8.5, the status information is returned in the **CmStatus** data structure. For the configuration confirm primitive, the values of the **CmStatus** are:

status

Name	Description
LCM_PRIM_OK	Successful request.
LCM_PRIM_NOK	Unsuccessful request.

reason

The content of this field is set to **LCM_REASON_NOT_APPL** when the **status** is set to **LCM_PRIM_OK**. When the **status** field is set to **LCM_PRIM_NOK**, the content of the reason field can be set to any of these values, indicating the reason of the failure of the primitive. The valid values of the configuration is:

Value	Description
LCM_REASON_INVALID_ELMNT	Invalid value of the **elmnt** field in the header.

The confirm structure format is:

```
typedef struct _aqSts            /* Statistics structure */
{
   DateTime dt;                  /* Date and time */
   union
   {
       AqPeerNodeSts  node;      /* Peer node statistics */
       AqUsapSts      uSap;      /* Upper Sap statistics */
       AqLsapSts      lSap;      /* Lower Sap statistics */
       AqGenSts       gen;       /* General statistics */
   } s;
} AqSts;
```

Table 6-43: Parameters of AqSts

Parameters	Description
node	Peer node statistics.
uSap	Upper SAP statistics.
lSap	Lower SAP statistics.
gen	General statistics. This is filled only in the confirm.

6.3.4.1 Peer Node Statistics

This contains the information regarding the primitives processed by Diameter for the given peer node. The **`elmnt`** value is **`STAQPEER`**. The **`cfm`** structure is filled only in the confirm. The structure format is:

```
typedef struct _aqPeerNodeSts    /* Peer Node Statistics */
{
    U8         numPeers;          /* Number of Peers */
    AqPeerSts  *peerSts;          /* Pointer to peer Statistics */
}AqPeerNodeSts;
```

The structure format for **AqPeerSts** is:

```
typedef struct _aqPeerSts
{
   LaqStr       hostId;      /* Host ID */
   U8           nmbCmd;      /* Number of Commands */
   AqStsInfo    peerStsInfo [LAQ_MAX_NUM_OF_CMDS];
                             /* Peer Statistics Info */
}AqPeerSts;
```

Table 6-44: Parameters of AqPeerSts

Parameters	Description
`hostId`	Host ID of the peer for which the peer statistics is needed. This is to be filled in the request.
`nmbCmd`	Number of commands for which the statistics is returned. This is to be filled in the confirm.
`peerStsInfo`	This structure contains the statistics information to be filled in the confirm.

6.3.4.1.1 AqStsInfo

The structure format for AqStsInfo is:

```
typedef struct _aqStsInfo
{
    AqAppId appId;          /* Application ID */
    U32     cmdCode;        /* Command code */
    Cntr    reqTxMsg;       /* Number of request transmitted */
    Cntr    reqRxMsg;       /* Number of request recieved */
    Cntr    resTxMsg;       /* Number of responses transmitted */
    Cntr    resRxMsg;       /* Number of responses recieved */
}AqStsInfo;
```

Table 6-45: Parameters of AqPeerStsInfo

Parameters	Description
`appId`	Application ID.
`cmdCode`	Command code.
`reqTxMsg`	This field specifies the number of request messages transmitted.
`reqRxMsg`	This field specifies the number of request messages received.
`resTxMsg`	This field specifies the number of response messages transmitted.
`resRxMsg`	This field specifies the number of response messages received.

6.3.4.2 Upper SAP Statistics

This statistics consists of number of requests (responses) sent (received) at a particular upper SAP. The structure format and parameter description are:

```
typedef struct _aqUsapSts
{
    S16         sapId;      /* SAP ID */
    U8          nmbCmd;     /* Number of Commands */
    AqStsInfo   sapStsInfo  [LAQ_MAX_NUM_OF_CMDS];
                            /* SAP Statistics Info */
    AqAppMsgSts appMsgSts;  /* Application message statistics */
}AqUsapSts;
```

Table 6-46: Parameters of AqUsapSts

Parameters	Description
`sapId`	SAP ID for which the statistics is needed. This is to be filled in the request.
`nmbCmd`	Number of commands for which the statistics is returned. This is to be filled in confirm.
`sapStsInfo`	This structure contains the statistics information to be filled in the confirm. Refer to Section 6.3.4.1.1.
`appMsgSts`	This structure contains the statistics information related to the application messages sent/recieved. The structure is filled in the confirm. Refer to Section 6.3.4.2.1.

6.3.4.2.1 AqAppMsgSts

This statistics consists of the number of request's bytes (response's bytes) sent/received at a particular upper SAP. It also consists the total number of requests (responses) sent/recieved at a particular upper SAP. The structure format and parameter description are:

```
typedef struct _aqAppMsgSts
{
   Cntr reqTxBytes;    /* Number of req bytes transmitted */
   Cntr reqRxBytes;    /* Number of req bytes received */
   Cntr resTxBytes;    /* Number of res bytes transmitted */
   Cntr resRxBytes;    /* Number of res bytes received */
   Cntr sentBytes;     /* Number of bytes sent */
   Cntr rcvdBytes;     /* Number of bytes received */
   Cntr appMsgSent;    /* Number of app Msg transmitted */
   Cntr appMsgRcvd;    /* Number of app Msg received */
}AqAppMsgSts;
```

Table 6-47: Parameters of AqAppMsgSts

Parameters	Description
`reqTxBytes`	This field contains the number of request bytes transmitted.
`reqRxBytes`	This field contains the number of request bytes received.
`resTxBytes`	This field contains the number of response bytes transmitted.
`resRxBytes`	This field contains the number of response bytes received.
`sentBytes`	This field contains the total number of bytes sent.
`rcvdBytes`	This field contains the total number of bytes received.
`appMsgSent`	This field contains the total number of application messages (both request/response) transmitted.
`appMsgRcvd`	This field contains the total number of application messages (both request/response) received.

6.3.4.3 Lower SAP Statistics

This statistics consists of number of messages sent and received at a particular SAP. The structure format and parameter description are:

```
typedef struct _aqLsapSts
{
   S16    sapId;          /* SAP ID */
   Cntr   msgSent;        /* Total number of messages sent */
   Cntr   msgRecv;        /* Total number of messages received */
   Cntr   byteSent;       /* Total number of bytes sent */
   Cntr   byteRecv;       /* Total number of bytes received */
}AqLsapSts;
```

Table 6-48: Parameters of AqLsapSts

Parameters	Description
`sapId`	SAP ID of the lower SAP for which the statistics is needed. This is to be filled in the request. All other fields are filled in confirm.
`msgSent`	This field specifies the number of messages transmitted.
`msgRecv`	This field specifies the number of messages received.
`byteSent`	This field specifies the number of bytes transmitted.
`byteRecv`	This field specifies the number of bytes received.

6.3.4.4 General Statistics

The general Diameter statistics. The **`elmnt`** value is **`STAQGEN`**. The structure format is:

```
typedef struct _aqGenSts
{
   Cntr  nmbMsgTx;       /* Number of messages transmitted */
   Cntr  nmbMsgRx;       /* Number of messages received */
   U32   byteSent;       /* Total number of bytes sent */
   U32   byteRecv;       /* Total number of bytes received */
   Cntr  encErrMsg;      /* Number of encoded errorneous message */
   Cntr  decErrMsg;      /* Number of decoded errorneous message */
} AqGenSts;
```

Table 6-49: Parameters of AqGenSts

Parameters	Description
`nmbMsgTx`	This field specifies the total number of messages transmitted by the node.
`nmbMsgRx`	This field specifies the total number of messages received by the node.
`byteSent`	This field specifies the total number of bytes sent by the node.
`byteRecv`	This field specifies the total number of bytes received by the node.
`endErrMsg`	This field specifies the number of messages with encoding error.
`decErrMsg`	This field specifies the number of messages with decoding error.

6.3.5 Solicited Status Request

Name	Solicited Status Request. **`SmMiLaqStaReq`** is invoked in the layer manager, and **`AqMiLaqStaReq`** processes this request in Diameter.
Direction	Layer manager to Diameter
Supplied	Yes
Response	Yes

Primitives

`AqMiLaqStaReq`

`SmMiLaqStaReq`

Description

The layer manager gathers the solicited status information about the various Diameter elements by using the management – solicited status procedure, which the layer manager initiates. The Diameter status request primitive, **`AqMiLaqStaReq`**, can be called more than once and at any time after the management – configuration procedure.

The Diameter status request primitive types are:

- System ID
- General Status
- SAP Status
- Peer Status

The **`aqMngmt.hdr.elmId`** field specifies the status request primitive type.

The status confirm, **`AqMiLaqStaCfm`**, and other system services primitives are called during the status procedure to return the status value.

The status request and status confirm primitives—**`AqMiLaqStaReq`** and **`AqMiLaqStaCfm`**, respectively—use the **`aqMngmt.t.ssta`** structure to specify the parameters.

Figure 6-5 shows the management – solicited status procedure.

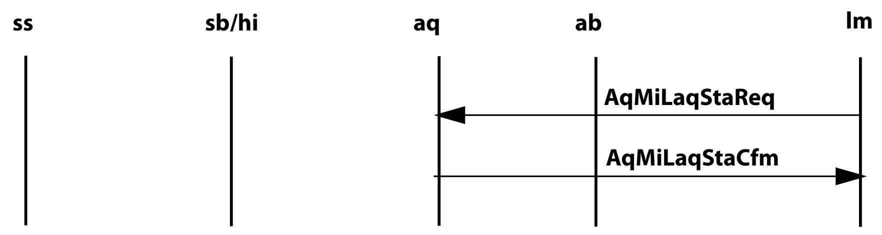


Figure 6-5: Management – Solicited status procedure

Synopsis

```
PUBLIC S16 AqMiLaqStaReq (pst, sta)
Pst     *pst;
AqMngmt *sta;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`sta`

Pointer to the management structure described in Section 6.2.2, "AqMngmt." The status-specific fields are described in the next sections.

In the **`Header`** of the status request, the values of the relevant fields are set as shown

Table 6-50: Header Values of Status Request

Field	Allowable Values

`elmId.elmnt`	`STAQGEN     : General`
	`STAQSID     : System ID`
	`STAQUSAP    : Upper SAP`
	`STAQLSAP    : Lower SAP`
	`STAQPEER    : Get the status of specific peer`
	`STAQPEERALL : Get the status of all the peers`

The **`CmStatus`** field is not used.

6.3.6 Solicited Status Confirm

Name	Solicited Status Confirm. The **`AqMiLaqStaCfm`** is the function invoked in Diameter. The **`SmMiLaqStaCfm`** is the function that processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes
Response	NA

Primitives

`AqMiLaqStaCfm`

`SmMiLaqStaCfm`

Description

Diameter uses this primitive to return the solicited status information to the layer manager. Figure 6-6 shows the data flow for the solicited status operation.

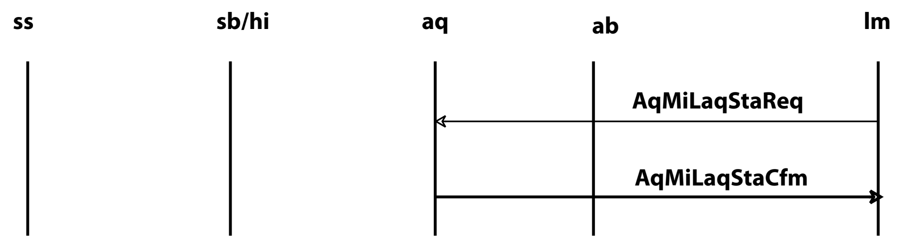


Figure 6-6: Management – Status procedure

Synopsis

```
PUBLIC S16 AqMiLaqStaCfm (pst, cfm)
Pst       *pst;
AqMngmt   *cfm;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`cfm`

This parameter represents a pointer to the management structure described in Section 6.2.2, "AqMngmt." The solicited status confirm-specific parameters are described in the next sections.

In the **`Header`** of the status request, the values of the relevant fields are set as shown:

Field	Allowable Values

`elmId.elmnt`	**`STAQGEN     : General`**
	`STAQSID     : System ID`
	`STAQUSAP    : Upper SAP`
	`STAQLSAP    : Lower SAP`
	`STAQPEER    : Get the status of specific Peer`
	`STAQPEERALL : Get the status of all the peers`

As described in Section 1.8.6, the status information is returned in the **`CmStatus`** data structure. For the configuration confirm primitive, the values of the **`CmStatus`** are:

`status`

Name	Description
`LCM_PRIM_OK`	Successful request.
`LCM_PRIM_NOK`	Unsuccessful request.

`reason`

The content of this field is set to **`LCM_REASON_NOT_APPL`** when the **`status`** is set to **`LCM_PRIM_OK`**. When the **`status`** field is set to **`LCM_PRIM_NOK`**, the content of the reason field can be set to any of these values, indicating the reason of the failure of the primitive. The valid values for the configuration are:

Value	Description
`LCM_REASON_INVALID_ELMNT`	Invalid value of the **`elmnt`** field in the header.

The status has the structure:

```
typedef struct _aqSsta          /* Solicited status */
{
   DateTime dt;                 /* Date and time */
   union
   {
      AqSapSta   sap;           /* SAP Status */
      SystemId   sysId;         /* System ID */
      AqGenSta   gen;           /* General status */
      AqPeerSta  peer;          /* Peer status */
   } s;
} AqSsta;                       /* Solicited status */
```

Table 6-51: Parameters of AqSsta

Parameters	Description
`sap`	SAP status. The `sapId` needs to be filled for the `elmId.elmnt` is `STAQUSAP` or `STAQLSAP`. The other fields are filled and returned by Diameter in the status confirm. Refer to Section 6.3.6.1 for details.
`sysId`	The system ID`.` This is filled and returned by Diameter in the status confirm for the `elmId.elmnt` is `STAQSID`. Refer to Section 6.3.6.2 for details.
`gen`	General status`.` This is filled and returned by Diameter in the status confirm for the `elmId.elmnt` is `STAQGEN.` Refer to Section 6.3.6.3 for details.
`peer`	Peer Status. The **host ID** needs to be filled for the `elmId.elmnt` is `STAQPEER.` The other fields are filled and returned by Diameter in the status confirm. Refer to Section 6.3.6.4 for details.

6.3.6.1 SAP Status

The SAP status gives the bound/unbound SAP status. It has the structure:

```
typedef struct _aqSapSta
{
   SpId sapId;                /* SAP Identifier */
   U8   status;               /* Bound/Unbound/Binding */
} AqSapSta;
```

Table 6-52: Parameters of AqSapSta

Parameters	Description
`sapId`	SAP ID. The allowable values: `0 – (max SAP configured - 1)`. • The maximum SAP configured = **`nmbLSaps`**, if **`elmnt`** is **`STAQLSAP`** **`nmbUSaps`**, if **`elmnt`** is **`STAQUSAP.`**
`status`	The SAP status. The allowable values: • **`LAQ_SAP_CFG`**: SAP configured but not bound. • **`LAQ_SAP_BND:`** SAP configured and bound. • **`LAQ_SAP_BINDING:`** SAP bind initiated and response awaited. This is valid only for lower SAP.

6.3.6.2 System ID Status

This identifies the part number and version number of the Diameter software. It has the structure:

```
typedef struct systemId       /* System ID */
{
   S16 mVer;                  /* Main version */
   S16 mRev;                  /* Main revision */
   S16 bVer;                  /* Branch version */
   S16 bRev;                  /* Branch revision */
   Txt *ptNmb;                /* Part number */
} SystemId;
```

Table 6-53: Parameters of `systemId`

Parameters	Description
`mVer`	Main version of the Diameter software.
`mRev`	Main revision of the Diameter software.
`bVer`	Branch version of the Diameter software.
`bRev`	Branch revision of the Diameter software.
`ptNmb`	Part number of the Diameter software—1000349.

6.3.6.3 General Status

The structure format and description:

```
typedef struct _aqGenSta      /* General status */
{
    U32  memSize;     /* Total static mem reserved by Diameter */
    U32  memAlloc;    /* Total static mem allocated by Diameter */
    Cntr nmbActvPeer; /* Number of active peer connections */
    Cntr nmbFailPeer; /* Number of failed Peer */
} AqGenSta;
```

Table 6-54: Parameters of AqGenSta

Parameters	Description
`memSize`	This field specifies the total memory reserved by Diameter.
`memAlloc`	This field specifies the memory allocated by Diameter.
`nmbActvPeer`	This field specifies the number of Active peer connections.
`nmbFailPeer`	This field specifies the total number of failed connections.

6.3.6.4 Peer Status

The structure format and parameter description:

```
typedef struct _aqPeerSta          /* Peer status */
{
   U8             nmbPeer;         /* Number of peers */
   AqPeerStaInfo  peerStaInfo[LAQ_MAX_NUM_OF_PEERS];
                                   /* List of Peers */
} AqPeerSta;
```

Table 6-55: Parameters of AqPeerSta

Parameters	Description
`nmbPeer`	This field specifies the number of Peers for which the status information is to be retrieved.
`peerStaInfo`	This structure contains the Peer status information. **Host ID** is filled in the status request. All other fields are filled in the status confirm.

6.3.6.4.1 Peer Status Information

The structure format and parameter description:

```
typedef struct _aqPeerStaInfo    /* Peer status */
{
   LaqStr      hostId;          /* Host ID */
   U16         pmqSize;         /* Number of messages in PMQ */
   U8          peerState;       /* State of the peer */
   AqPeerCapb  peerCapb;        /* Peer Capabilities */
} AqPeerStaInfo;
```

Table 6-56: Parameters of AqPeerStaInfo

Parameters	Description
`hostId`	Host ID of the peer.
`peerState`	This field specifies the current state of the peer. Possible values are: • **`AQ_PSMST_CLOSED`** - All the peer entries are initialized to this state. • **`AQ_PSMST_OPEN`** - Once the connection is established and CER is exchanged, peer state is set to **`AQ_PSMST_OPEN`**. • **`AQ_PSMST_WAIT_CONN_ACK`** - After initiating the connection, peer is waiting for connection acknowledgement, peer state is set to **`AQ_PSMST_WAIT_CONN_ACK`**. • **`AQ_PSMST_WAIT_I_CEA`** - After the connection is established and CER is sent and peer is waiting for CEA, peer state is set to **`AQ_PSMST_WAIT_I_CEA`**. • **`AQ_PSMST_WAIT_CONN_ACK_ELECT`** - Node has received incoming connection from the peer along with the CER, peer state is set to **`AQ_PSMST_WAIT_CONN_ACK_ELECT`**. • **`AQ_PSMST_WAIT_RETURNS`** - When the connections are established in both ways (initiating and responding) and CER are exchanged, peer state is set to **`AQ_PSMST_WAIT_RETURNS`**. • **`AQ_PSMST_SUSPECT`** - When device watchdog timer expires and there is no pending DWA in pending message queue, peer state is set to **`AQ_PSMST_SUSPECT`**. • **`AQ_PSMST_REOPEN`** - At the time of failback, if the connection is re-established, the peer state is set to **`AQ_PSMST_REOPEN`**. • **`AQ_PSMST_CLOSING`** - When Diameter receives a control request from LM to delete the peer, the state of the peer is set to **`AQ_PSMST_CLOSING`**. • **`AQ_PSMST_DOWN`** - If device watchdog timer expires in **`AQ_PSMST_SUSPECT`** state or if the Diameter receives termination indication from the lower layer, the peer state is set to **`AQ_PSMST_DOWN`**.
`peerCapb`	This structure contains the capability information of the Peer, such as vendor ID, supported application ID, and so on.

6.3.6.4.2 AqPeerCapb

The structure format and description is:

```
typedef struct _aqPeerCapb     /* Peer Capability */
{
   AqVendorId vendorId;         /* Vendor ID */
   U8         nmbSuppVenId;     /* Number of supp Vendor ID */
   U8         nmbInbandSecId;   /* Number of inband security ID */
   U8         nmbVenSpecAppId;  /* Number of ven specific app ID */
   U8         nmbAppId;         /* Number of app ID */
   AqVendorId suppVendId[LAQ_MAX_SUPP_VEN_ID];
                                /* Array of supp vendor ID */
   AqSec      inBandSec[LAQ_MAX_INBAND_SEC];
                                /* Array of inband security ID */
   AqVendorId venSpecAppId[LAQ_MAX_VEN_SPEC_APPID];
                                /* Array of vendor specific app ID */
   AqAppId    suppAppId[LAQ_MAX_APP_ID]
                                /* Array of supported app ID */
} AqPeerCapb;
```

Table 6-57: Parameters of AqPeerCapb

Parameters	Description
vendorId	This field specifies the vendor ID supported by the peer.
nmbSuppVenId	This field specifies the actual number of supported vendor ID associated with the peer.
nmbInbandSecId	This field specifies the actual number of inband security ID supported by the peer.
nmbVenSpecAppId	This field specifies the actual number of vendor specific application ID supported by the peer.
nmbAppId	This field specifies the number of application ID supported by the peer.
suppVenId	This field specifies the array of supported vendor ID exchanged during the CER.
inBandSec	This field specifies the array of inband security ID exchanged during the CER.
venSpecAppId	This field contains the supported vendor ID exchanged during CER.
suppAppId	This field specifies the array of application ID exchanged during CER.

6.3.7 Unsolicited Status Indication

Name	Unsolicited Status Indication. The **`AqMiLaqStaInd`** is invoked in Diameter, and **`SmMiLaqStaInd`** processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes

Primitives

`AqMiLaqStaInd`

`SmMiLaqStaInd`

Description

The management – unsolicited status procedure provides unsolicited status information about the Diameter elements to the layer manager. Diameter initiates this procedure. The Diameter status indication primitive, **`AqMiLaqStaInd`**, can be called more than once and, if the unsolicited status is enabled, at any time after the configuration procedure. The Diameter status indication primitive is not called if the unsolicited status is disabled. The unsolicited status can be enabled or disabled with the management – control procedure.

The **`aqMngmt.t.usta`** structure specifies the parameters used by the status indication primitive, **`AqMiLaqStaInd`**.

Figure 6-7 shows the management – unsolicited status procedure.

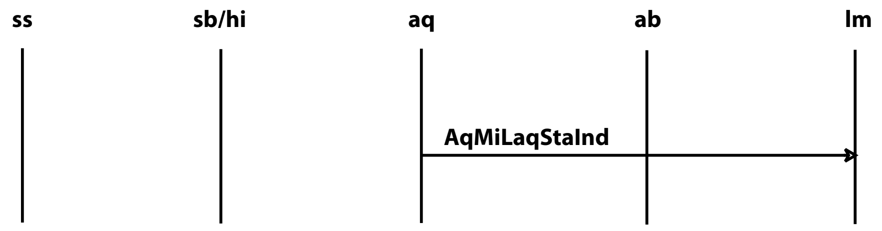


Figure 6-7: Management – Unsolicited Status Procedure

Synopsis

```
PUBLIC S16 AqMiLaqStaInd (pst, sta)
Pst       *pst;
AqMngmt   *usta;
```

Parameters

pst

Refer to Section 1.8.1, "Post."

usta

Unsolicited Status Indication. It is the pointer to the management structures as described in Section 6.2.2, "AqMngmt." Specific values for the header are not used in the status indication. Also the Cmstatus structure is not used. The unsolicited status structure has format:

```
typedef struct _aqUsta
{
   DateTime  dt;       /* Date and Time */
   cmAlarm   alarm;    /* Alarm */
   AqUstaDgn dgn;      /* alarm Diagnostic */
 }AqUsta;
```

Table 6-58: Parameter of AqUsta

Parameter	Description
`dt`	Date and Time. Refer to Section 1.8.3.
`alarm`	Alarm category and event. This structure provides information about the date and time, category, event that cause for an alarm generation.
`dgn`	Alarm diagnostic. This enables the layer manager to know the reason for the alarm.

alarm

The structure format and parameter description is:

```
typedef struct cmAlarm
{
   DateTime   dt;           /* Date and Time */
   U16        category;     /* Category */
   U16        event;        /* Event */
   U16        cause;        /* Cause for  event */
   Cntr       maxRetryCnt;  /* Maximum Retry count */
}CmAlarm;
```

Table 6-59: Parameters of cmAlarm

Parameter	Description
`dt`	Date and Time. Refer to Section 1.8.3, "Date and Time."
`category`	The alarms generated by Diameter are divided into various categories. These categories depend on the nature of the alarm generated.
`event`	This parameter specifies the event that caused the generation of a status indication to the layer manager, from the Diameter software. Event codes are not unique and must be interpreted in conjunction with the category of the generated alarm.
`cause`	This parameter specifies the cause for an alarm.
`maxRetryCnt`	This field specifies the number of retry count; for example, at the time of reconnection, alarm is sent to Layer manager once the maxRetryCnt is exceeded. The same retry count is filled here to inform LM, the number of times the retry is being done.

`category`

Table 6-60: Allowable Values of category:

Allowable Values	Description
`LCM_CATEGORY_RESOURCE`	An alarm in this category is generated if an error occurs during allocation, deallocation of memory from system resources.
`LCM_CATEGORY_INTERFACE`	An alarm in this category is generated if an error occurs during handling of interface events.
`LCM_CATEGORY_INTERNAL`	An alarm in this category is generated if an internal software error occurs.
`LCM_CATEGORY_PROTOCOL`	An alarm in this category is generated if a protocol error occurs and recovery is not defined.

event

For each alarm category, the allowable values are:

Allowable Values	Description
LCM_EVENT_UI_INV_EVT	This event is generated if an illegal event is received and if an interface parameter is in error at the upper interface.
LCM_EVENT_LI_INV_EVT	This event is generated if an illegal event is received and if an interface parameter is in error at the lower interface.
LCM_EVENT_BND_FAIL	This event is generated if the bind procedure with the lower layer fails. The bind procedure can fail either due to the absence of a bind confirm or if there is a negative bind confirm from the lower layer.
LCM_EVENT_BND_OK	This event is generated if the bind procedure with the lower layer is successful.
LCM_EVENT_DMEM_ALLOC_FAIL	This event is generated if the dynamic memory pool goes below the configured low threshold.
LAQ_EVENT_CONN_OK	This event is generated if the Diameter base protocol connects successfully with the peer.
LAQ_EVENT_CONN_FAIL	This event is generated if the Diameter base protocol fails to connect to the peer.
LAQ_EVENT_MAX_CONN_RETRY	This event is generated if the Diameter base protocol reaches the maximum number of retry while establishing connections.
LAQ_EVENT_FAILOVER_INIT	This event is generated if the Diameter base protocol invokes failover.
LAQ_EVENT_PEER_DOWN	This event is generated if any of the peer goes down.
LAQ_EVENT_PEER_UP	This event is generated if any of the peer comes up.
LAQ_EVENT_PEER_DOWN_PERM	This event is generated when the peer is down with disconnect reason as **DO_NO_WANT_TO_TALK_TO_YOU**.
LAQ_EVENT_NO_ALT_PEER_FOUND	This event is generated when failover module fails to retrieve the alternate peer.

Allowable Values	Description
`LAQ_EVENT_PEER_NOT_INSERTED`	This event is generated if any of the peer is not inserted into the database at the time of static configuration.
`LAQ_EVENT_REALM_NOT_ISNERTED`	This event is generated if any of the realm is not inserted into the database at the time of static configuration.
`LAQ_EVENT_PEER_DELETED`	This event is generated if any of the peer is deleted when control request is received.
`LAQ_EVENT_OPEN_SERV_FAIL`	This event is generated if the server open fails.
`LAQ_EVENT_LI_BND_CFM`	This event is generated if the bind confirm is received from lower layer.
`LAQ_EVENT_LI_HIT_CON_CFM`	This event is generated if the connection confirmed is received from HIT interface.
`LAQ_EVENT_LI_HIT_CON_IND`	This event is generated if the connection indication is received from HIT interface.
`LAQ_EVENT_SCT_ENDP_OPEN_CFM`	This event is generated if the end point open confirm is received from SCT interface.
`LAQ_EVENT_SCT_ENDP_CLOSE_CFM`	This event is generated if the end point close confirm is received from SCT interface.
`LAQ_EVENT_LI_SCT_ENDP_ASSOC_IND`	This event is generated if the association indication is received from SCT interface.
`LAQ_EVENT_LI_HIT_DAT_IND`	This event is generated if the data indication is received from HIT interface.
`LAQ_EVENT_LI_SCT_DAT_IND`	This event is generated if the data indication is received from SCT interface.
`LAQ_EVENT_LI_HIT_DISC_IND`	This event is generated if the disconnect indication is received from HIT interface.
`LAQ_EVENT_LI_SCT_TERM_IND`	This event is generated if the termination indication is received from SCTP interface.
`LAQ_EVENT_LI_SCT_TERM_CFM`	This event is generated if the termination confirm is received from SCTP interface.
`LAQ_EVENT_LI_SCT_SET_PRI_CFM`	This event is generated to set the priority confirm from SCT interface.
`LAQ_EVENT_LI_SCT_HBEAT_CFM`	This event is generated if the heart beat confirm is received from SCT interface.

Allowable Values	Description
LAQ_EVENT_LI_SCT_STA_CFM	This event is generated if the status confirm is received from SCT interface.
LAQ_EVENT_LI_SCT_STA_IND	This event is generated if the status indication is received from SCT interface.
LAQ_EVENT_LI_HIT_FLC_IND	This event is generated if the flow control indication is received from HIT interface.
LAQ_EVENT_LI_SCT_FLC_IND	This event is generated if the flow control indication is received from SCT interface.
LAQ_EVENT_UI_SAP_UBND_SUCC	This event is generated when the USAP is unbound successfully.
LAQ_EVENT_LI_SCT_ASSOC_CFM	This event is generated if the association confirm is received from the SCT interface.
LAQ_EVENT_LI_SCT_PEER_INSERTED	This event is generated when peer is inserted in Peer table.
LAQ_EVENT_RECONN_TIMEOUT	This event is generated when reconnect timer is expired.
LAQ_EVENT_OPEN_SERVER_UP	This event is generated when server open is successful.

cause

Allowable Values	Description
LCM_CAUSE_INV_SPID	SAP ID is out of range.
LCM_CAUSE_INV_STATE	Invalid SAP ID.
LCM_CAUSE_INV_PAR_VALUE	Invalid parameter value.
LCM_CAUSE_UNKNOWN	Unknown cause.
LAQ_CAUSE_N_RETRY	Maximum retry count is reached.
LAQ_CAUSE_UNKNOWN_OPT_AVP	Unknown optional AVP is received.
LAQ_CAUSE_NO_PEER_INOPEN_ST	No peer is in Open state.
LAQ_CAUSE_INV_SER_STATE	Invalid server state.
LAQ_CAUSE_NO_COMM_APP	No common application.
LAQ_CAUSE_NO_COMM_SEC	No common security.
LAQ_CAUSE_CER_EXCHGD	CER is exchanged.
LAQ_CAUSE_DPR_SENT	DPR is sent.
LAQ_CAUSE_DPR_RECEIVED	DPR is received.

Allowable Values	Description
`LAQ_CAUSE_DW_TIMEOUT`	Device watchdog time-out has happened.
`LAQ_CAUSE_TERM_IND`	Termination indication is received.
`LAQ_CAUSE_TCP_IPSEC_SRV_OPEN_FAIL`	TCP - IPSec server open fails.
`LAQ_CAUSE_TCP_TLS_SRV_OPEN_FAIL`	TCP - TLS server open fails.
`LAQ_CAUSE_SCTP_IPSEC_SRV_OPEN_FAIL`	SCTP-IPSec server open fails.
`LAQ_CAUSE_DEC`	Decoding fails.
`LAQ_CAUSE_TCP_IPSEC_SRV_UP`	TCP-IPSec server is up.
`LAQ_CAUSE_TCP_TLS_SRV_UP`	TCP-TLS server is up.
`LAQ_CAUSE_SCTP_IPSEC_SRV_UP`	SCTP-IPSec server is up.
`LAQ_CAUSE_TCP_IPSEC_SRV_DISCONNECTED`	TCP-IPSec server is disconnected.
`LAQ_CAUSE_TCP_TLS_SRV_DISCONNECTED`	TCP-TLS server is disconnected.
`LAQ_CAUSE_SCTP_IPSEC_SRV_DISCONNECTED`	SCTP-IPSec server is disconnected.
`LAQ_CAUSE_TCP_IPSEC_SRV_OPEN_SUCC`	TCP IpSec server is open successfully.
`LAQ_CAUSE_TCP_TLS_SRV_OPEN_SUCC`	TCP TLS server is open successfully.
`LAQ_CAUSE_SCTP_IPSEC_SRV_OPEN_SUC`	SCTP IPsec server is open successfully.
`LAQ_CAUSE_SAP_BND`	SAP is in bound state.
`LAQ_CAUSE_SAP_UBND`	SAP is in unbound state.
`LAQ_CAUSE_TLS_FAIL`	TLS establishment fails.
`LAQ_CAUSE_DWR_EXCHGD`	DWR are exchanged.

dgn

The format of this structure is

```
typedef struct _aqUstaDgn
{
    U8              type;         /* type */
    union
    {
        AqAppId     appId;        /* Application ID */
        typedef     struct _aqPeerDgn
        {
            U32     peerConnId;   /* Peer Connection ID */
            LaqStr  hostId;       /* Host ID */
            U8      peerState;    /* Peer state */
            U8      peerEvnt;     /* Peer event */
        }AqPeerDgn;
    LaqStr          realmName;    /* Realm name */
    U32             avpCode;      /* AVP Code */
    Memory          mem;          /* Memory pool and region */
    SpId            sapId;        /* SAP ID */
    }u;
}AqUstaDgn;
```

Table 6-61: Parameters of AqUstaDgn

Parameter	Description
`type`	Type of diagnostic information. The allowable values are: • `LAQ_USTA_DGNVAL_NONE` • `LAQ_USTA_DGNVAL_MEM` • `LAQ_USTA_DGNVAL_SAPID` • `LAQ_USTA_DGNVAL_APPID` • `LAQ_USTA_DGNVAL_PEER` • `LAQ_USTA_DGNVAL_REALM` • `LAQ_USTA_DGNVAL_AVPCODE`
`appId`	Application ID.
`avpCode`	AVP Code of the AVP for which the alarm is raised at the time of encoding/ decoding.
`hostId`	Host ID of the peer for which the alarm is raised.
`peerConnId`	Connection ID of the peer for which the alarm is raised.
`mem`	Memory Pool and region.
`sapId`	SAP ID. The allowable values: 0 - (max SAP-1) The maximum SAP = `nmbLSaps`, if elmnt is `STAQLSAP` or `nmbUSaps`, if elmnt is `STAQUSAP`.

6.3.8 Control Request

Name	`AqMiLaqCntrlReq`
Direction	Diameter to layer manager
Supplied	Yes

Primitives

`AqMiLaqCntrlReq`

`SmMiLaqCntrlReq`

Description

The Layer Manager uses the control procedure to control the Diameter elements. The Layer Manager initiates this procedure. The Diameter control request primitive, **`AqMiLaqCntrlReq`** can be called more than once and at any time after the configuration procedure.

These Diameter control request primitives can be called:

- Trace Control
- Debug Control
- SAP Control
- Realm Control
- Peer Control

The aqMngmt.u.cntrl.action field specifies the control request primitive type.

The aqMngmt.u.cntrl.subaction field specifies the element to be controlled.

The aqMngmt,u.cntrl structure specifies the parameters used by the control request primitive **`AqMiLaqCntrlReq`**.

Figure 6-8 shows the management - control request procedure.

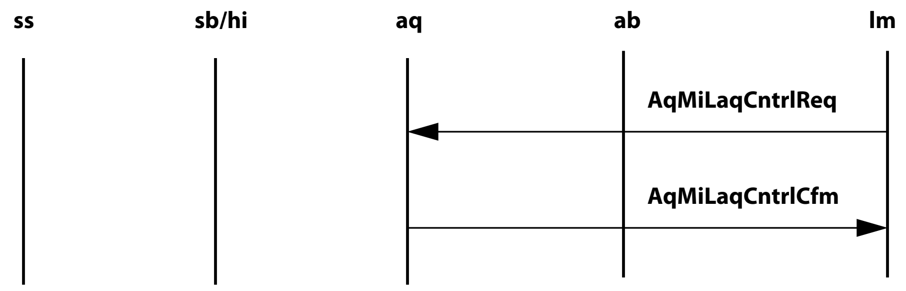


Figure 6-8: Management – Control Procedure

Synopsis

```
PUBLIC S16 AqMiLaqCntrlReq (pst, cntrl)
Pst       *pst;
AqMngmt   *cntrl;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`cntrl`

This is a pointer to the management structure described in Section 6.2.2, "AqMngmt". The control request specific parameters are described in the next sections.

In the Header of the control request, the values of the relevant fields are set as:

Table 6-62: Header Values of Control Request

Field	Allowable Values

`elmId.elmnt`	`STAQGEN : General`
	`STAQLSAP : Lower SAP`
	`STAQUSAP : Upper SAP`
	`STAQREALM : Realm`
	`STAQPEER : Peer`

The control structure has the format:

```
typedef struct _aqCntrl
{
   DateTime dt;                /* Date and Time */
   Action   action;            /* Action */
   Action   subAction;         /* Sub action */
   union
   {
      AqSapCntrl    sap;       /* SAP Control */
      AqTrcCntrl    trc;       /* Lower SAP trace control */
      #ifdef DEBUGP
      AqDbgCntrl    dbg;       /* Debug Control */
      #endif
      AqRealmCntrl  realm;     /* Realm Control */
      AqPeerCntrl   peer;      /* Peer Control */
   } u;
} AqCntrl;
```

Table 6-63: Parameters of AqCntrl

Parameter	Description
`action`	This field specifies the action that the Diameter layer must take.
`subAction`	This field specifies the protocol element on which the Diameter layer takes the specified action.
`sap`	The layer manager can selectively bind/unbind the different SAPs.
`trc`	The layer manager can selectively enable/disable various levels of message tracing.
`dbg`	The layer manager can selectively enable/disable various levels of debug printing.
`realm`	The layer manager can delete the realms from the Realm table.
`peer`	The layer manager can delete the peers from the Peer table.

`dt`

Refer to Section 1.8.3, "Date and Time".

`action`

This field specifies the action the Diameter layer must take.

Element	Action	SubAction	Description
STAQGEN	**AENA**	**SAUSTA**	Enable alarms.
STAQGEN	**ADISIMM**	**SAUSTA**	Disable alarms.
STAQGEN	**AENA**	**SADBG**	Enable Debug printing.
STAQGEN	**ADISIMM**	**SADBG**	Disable Debug printing.
STAQGEN	**LAQ_START**	--	Start the server and initiate the connection with other peers.
STAQGEN	**ASHUTDOWN**	--	Shutdown the Diameter layer.
STAQLSAP	**AENA**	**SATRC**	Enable trace generation.
STAQLSAP	**ADISIMM**	**SATRC**	Disable trace generation.
STAQLSAP	**ABND_ENA**	--	Bind and enable Lower SAP.
STAQLSAP	**ADEL**	--	Delete the lower SAP.
STAQUSAP	**ADEL**	--	Delete the upper SAP.
STAQLSAP	**AUBND_DIS**	--	Unbind and disable lower SAP.

Element	Action	SubAction	Description
STAQUSAP	**AUBND_DIS**	--	Unbind and disable upper SAP.
STAQPEER	**ADEL**	**LAQ_SPEC**	Delete the Peer Entries as specified.
STAQREALM	**ADEL**	**LAQ_SPEC**	Delete the Realm Entry as specified.
STAQPEER	**ADEL**	**LAQ_SAALL**	Delete all the Peer Entries present in Peer table.
STAQREALM	**ADEL**	**LAQ_SAALL**	Delete all the Realm entries present in Realm table.

6.3.8.1 SAP Control

The SAP control has the format:

```
typedef struct _aqSapCntrl     /* SAP Control */
{
   SpId  sapId;                /* SAP Identifier */
} AqSapCntrl;
```

Table 6-64: Parameter of AqSapCntrl

Parameter	Description
sapId	The SAP ID. The allowable values: **0 – (max SAP configured - 1).** The maximum SAP configured: • **nmbLSaps**, if **elmnt** is **STAQLSAP.** • **nmbUSaps**, if **elmnt** is **STAQUSAP.**

6.3.8.2 Peer Trace Control

The structure and parameter description:

```
typedef struct _aqTrcCntrl       /* Trace control */
{
   SpId spId;                     /* Service Provider ID */
   U32  trcMask;                  /* Trace Mask */
   S32  trcLen;                   /* Trace Length */
} AqTrcCntrl;
```

Table 6-65: Parameters of AqTrcCntrl

Parameter	Description
`spId`	This field specifies the Service Provider SAP ID.
`trcMask`	The direction of the messages to be traced. The allowable values: • `LAQ_TRC_PNODE_IN` • `LAQ_TRC_PNODE_OUT` It can be set to a combination of the aforementioned allowable values. For example, if tracing in both directions of the layer are to be enabled/disabled, the `trcMask` is set to `LAQ_TRC_PNODE_IN` | `LAQ_TRC_PNODE_OUT`.
`trcLen`	The length of the data buffer to be traced. The allowable values: • `0         /* no data */` • `-1        /* full data */` • `1 – 32676`

6.3.8.3 Debug Control

Debug printing control. The structure and parameter descriptions are:

```
typedef struct _aqDbgCntrl    /* Debug control */
{
   U32  dbgMask;              /* Debug mask */
   #ifdef LAQ_FILE_LOG
   Bool fileLogEnb;           /* To enable/disable file logging */
   Txt  filePath[LAQ_MAX_FILE_PATH];
                              /* Path to store log files */
   U32  nmbDbgLines;          /* Number of lines per Debug file */
   #endif
} AqDbgCntrl;
```

Table 6-66: Parameter of AqDbgCntrl

Parameter	Description
dbgMask	Debug mask. It indicates the debug printing control. For example, some of the possible values are: • **DBGMASK_UI** - For debug prints at upper interface. • **DBGMASK_LI** - For debug prints at lower interface. • **DBGMASK_MI** - For debug prints at layer manager. • **LAQ_DBGMASK_MP** - For debug prints of message processor functions. • **LAQ_DBGMASK_DENC** - For debug prints of encoder/decoder functions. • **LAQ_DBGMASK_PSM** - For debug prints of peer state machine functions. • **LAQ_DBGMASK_CDB** - For debug prints of core database functions. • **LAQ_DBGMASK_TMR** - For debug prints of timer functions. • **LAQ_DBGMASK_MODULES** - For debug prints at module level interface functions. • **LAQ_DBGMASK_SUPP_FUNC** - For debug prints of supporting functions. • **LAQ_DBGMASK_SMEM** - For debug prints of static memory allocation. • It can be set to a combination of the above mentioned allowable values. For example, if the debugging prints at the upper and lower interfaces of the layer are to be enabled/disabled, the **dbgMask** is set to **DBGMASK_UI** | **DBGMASK_LI**.
fileLogEnb	This field can be used to enable or disable file logging. **Note:** File logging is supported on Linux and Solaris platforms only.

Table 6-66: Parameter of AqDbgCntrl

Parameter	Description
`filePath`	This field specifies the path to store the log files.
`nmbDbgLines`	This field specifies the number of lines per debug files.

6.3.8.4 Peer Control

Peer Control allows the LM to delete the Peers from the Peer table. In such cases, pending message queue of the associated peers are cleaned up.

The structure and description are:

```
typedef struct _aqPeerCntrl
{
    U8      nmbPeers;                      /* Number of peers */
    LaqStr  hostId[LAQ_MAX_NUM_OF_PEERS];  /* Array of host ID*/
}AqPeerCntrl;
```

Table 6-67: Parameter of AqPeerCntrl

Parameter	Description
`nmbPeers`	Number of peers to be deleted from the peer table.
`hostId`	Array of host IDs which needs to be deleted from the peer table.

6.3.8.5 Realm Control

Realm table control allows the LM to delete the Realm from the Realm table.

The structure and parameter description are:

```
typedef struct _aqRealmCntrl
{
   U8     nmbRealms;                        /* Number of Realms */
   LaqStr realmName[LAQ_MAX_NUM_OF_REALMS]; /* Array of Realms */
}AqRealmCntrl;
```

Table 6-68: Parameters of AqRealmCntrl

Parameter	Description
`nmbRealms`	Number of realms to be deleted.
`realmName`	Array of realm name which needs to be deleted from the realm table.

6.3.9 Control Confirm

Name	Control Confirm. The **`AqMiLaqCntrlCfm`** is invoked in Diameter, and **`SmMiLaqCntrlCfm`** processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes

Primitives

`AqMiLaqCntrlCfm`

`SmMiLaqCntrlCfm`

Description:

Diameter uses this primitive to confirm the **`AqMiLaqCntrlReq`** primitive. Figure 6-9 shows the data flow of the control operation.

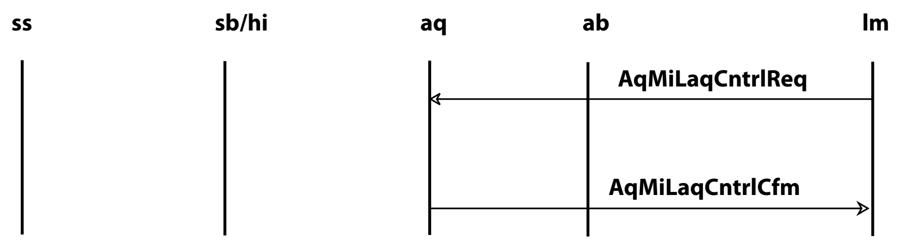


Figure 6-9: Management – Control procedure

Synopsis

```
PUBLIC S16 AqMiLaqCntrlCfm (pst, cfm)
Pst        *pst;
AqMngmt    *cfm;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`cfm`

This parameter represents a pointer to the management structure described in Section 6.2.2. The control confirm contains only the header and common status.

In the **`Header`** of the configuration confirm, the values of the relevant fields are set as:

Field	Allowable Values

`elmId.elmnt`	**`STAQGEN : General`**
	`STAQLSAP : Lower SAP`
	`STAQUSAP : Upper SAP`
	`STAQREALM : Realm`
	`STAQPEER : Peer`

As described in Section 1.8.6, the status information is returned in the **`CmStatus`** data structure. For the configuration confirm primitive, the values of the **`CmStatus`** are:

`status`

Name	Description
`LCM_PRIM_OK`	Successful request.
`LCM_PRIM_NOK`	Unsuccessful request.

`reason`

The content of this field is set to **`LCM_REASON_NOT_APPL`** when the **`status`** is set to **`LCM_PRIM_OK`**. When the **`status`** field is set to **`LCM_PRIM_NOK`**, the content of the reason field can be set to any of these, indicating the reason of the failure of the primitive. The valid values valid for the configuration are:

Value	Description
`LCM_REASON_INVALID_ACTION`	Invalid action in the control request.
`LCM_REASON_INVALID_SUBACTION`	Invalid subaction in the control request.
`LCM_REASON_INVLID_PAR_VAL`	Invalid parameter value in the control request.
`LCM_REASON_INVALID_ELMNT`	Invalid value of **`elmnt`** field in the header.

6.3.10 Trace Indication

Name	Trace Indication. The **`AqMiLaqTrcInd`** is invoked in Diameter, and the **`SmMiLaqTrcInd`** processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes

Primitives

`AqMiLaqTrcInd`

`SmMiLaqTrcInd`

Description

The management – tracing procedure presents the Diameter trace information to the layer manager. Diameter initiates this procedure. The Diameter trace indication primitive, **`AqMiLaqTrcInd`**, can be called more than once. If tracing is enabled, the **`AqMiLaqTrcInd`** can be called at anytime upon receiving or transmitting a Diameter Message, after the management – configuration procedure. The **`AqMiLaqTrcInd`** is not called if tracing is disabled. Tracing can be enabled or disabled with the management – control procedure.

The **`AqMngmt.u.trc`** structure specifies the parameters used by the trace indication primitive, **`AqMiLaqTrcInd`**.

Figure 6-10 shows the management – trace indication procedure.

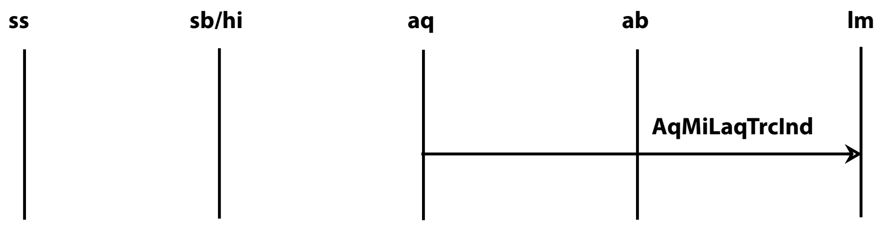


Figure 6-10: Management – Trace indication procedure

Synopsis

```
PUBLIC S16 AqMiLaqTrcInd (pst, trc, mBuf)
Pst     *pst;
AqMngmt *trc;
Buffer  *mBuf;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`trc`

It is a pointer to the management structure as described in Section 6.2.2, "AqMngmt." Specific values for the header are not used in the status indication. Also, the **`CmStatus`** structure is not used. The trace specific structure has the format:

```
typedef struct _aqTrc
{
   DateTime  dt;               /* Date and time */
   U16       event;            /* Event */
   U8        protocol;         /* Protocol */
   SpId      spId;             /* Provider SAP Identifier */
   SuId      suId;             /* User SAP identifier */
   AqAddr    srcAddr;          /* Source Address */
   AqAddr    dstAddr;          /* Destination Address */
   U32       appTransId;       /* Application transaction ID */
} AqTrc;
```

Table 6-69: Parameters of AqTrc

Parameters	**Description**
`dt`	Date and time. Refer to Section 1.8.3 for details.
`event`	Event causing the trace.
`protocol`	This field specifies the protocol to identify the address structure.
`suId`	This field specifies the Service User SAP ID.
`spId`	This field specifies the Service Provider SAP ID.
`srdAddr`	This field specifies the IP address of the originating node.
`destAddr`	This field specifies the source port.
`srcPort`	This field specifies the IP address of the destination node.
`dstPort`	This field specifies the destination port.
`appTransId`	This field specifies the transaction ID of the message.

6.3.11 Probe Request

Name	Probe Request. **`SmMiLaqPrbReq`** is invoked in the layer manager, and **`AqMiLaqPrbReq`** processes this request in Diameter.
Direction	Layer manager to Diameter
Supplied	Yes
Response	Yes

Primitives

`AqMiLaqPrbReq`

`SmMiLaqPrbReq`

Description

The layer manager probes the database information about the various Diameter elements by using the management – probe procedure, which the layer manager initiates. The Diameter probe request primitive, **`AqMiLaqPrbReq`**, can be called more than once and at any time after the management – configuration procedure.

Diameter probe request primitive can be used at the time of debugging. It dumps the complete peer table, realm table, AVP dictionary or DM dictionary based on the header field.

The Diameter probe request primitive types are:

- Peer
- Realm
- AVP
- DM

The **`aqMngmt.hdr.elmId`** field specifies the status request primitive type.

In the **`Header`** of the status request, the values of the relevant fields are set as shown:

Field	Allowable Values

`elmId.elmnt`	**`STAQPEER : Peer Entry`**
	`STAQALLPEER : All Peer Entry`
	`STAQREALM : Realm Entry`
	`STAQALLREALM : All Realm Entry`
	`STAQAVP : AVP Dictionary`
	`STAQDM : DM Dictionary`

The probe confirm, **`AqMiLaqPrbCfm`**, and other system services primitives are called during the probe procedure to return the status value.

The status request and status confirm primitives—**`AqMiLaqPrbReq`** and **`AqMiLaqPrbCfm`**, respectively—use the **`aqMngmt.t.prb`** structure to specify the parameters.

Figure 6-11 shows the management – probe procedure.

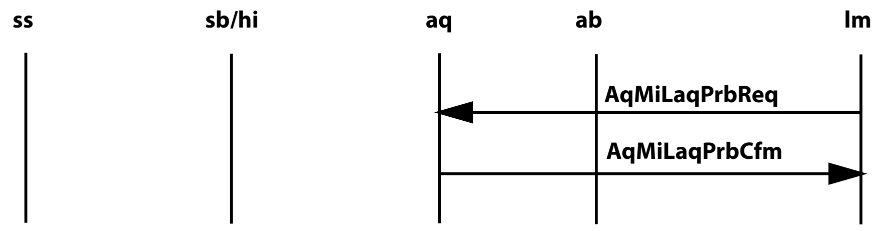


Figure 6-11: Management – Probe procedure

Synopsis

```
PUBLIC S16 AqMiLaqPrbReq (pst, sta)
Pst     *pst;
AqMngmt *pro;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`pro`

Pointer to the management structure described in Section 6.2.2, "AqMngmt." The status-specific fields are described in the next sections.

The probe structure is:

```
typedef struct _aqPrb    /* Probe Info */
{
   DateTime dt;         /* Date and time */
   union
   {
      union
      {
         AqPeerPrbReq  peerPrbReq;
         AqRealmPrbReq realmPrbReq;
      }AqPrbReq;        /* Specifies information to be probed for*/
      union
      {
         AqPeerPrbCfm peer;       /* Peer related information */
         AqRealmPrbCfm realm;     /* Realm related information */
         AqAvpPrbCfm avp;         /* Avp related information */
         AqDmPrbCfm dm;           /* DM related information */
      }AqPrbCfm;
   }p;

} AqPrb;
```

Table 6-70: Parameters of AqPrb

Parameters	Description
`dt`	This structure specifies the data and time.
`AqPrbReq`	This union contains the information to be probe. This is filled by the stack manager at the time of request.
`peerPrbReq`	This structure contains the array of peer host ID for which the information is needed.
`realmPrbReq`	This structure contains the array of realm names for which the information is needed.
`appId`	This field specifies the application ID for which the AVP and DM Dictionary is to be probed.
`AqPrbCfm`	This union contains the probed information. This is filled by the Diameter layer at the time of confirmation.
`peer`	This structure contains the probed information associated with the peers sent in Peer Probe request.
`realm`	This structure contains the probed information associated with the realms sent in realm probe request.
`avp`	This structure contains the probed information related to AVP Dictionary associated with the **`appId`** sent in Probe request.
`dm`	This structure contains the probed information related to DM Dictionary associated with the **`appId`** sent in Probe request.

6.3.11.1 AqPeerPrbReq

The structure and parameter description for **AqPeerPrbReq** are:

```
typedef  struct _aqPeerPrbReq
{
   U8     nmbPeer;                          /* Number of Peers */
   LaqStr hostId[LAQ_MAX_NUM_OF_PEERS];     /* Array of Host ID */
}AqPeerPrbReq;
```

Table 6-71: Parameters of AqPeerPrbReq

Parameters	Description
nmbPeers	Number of peers for which the Peer information is to be retrieved.
hostId	Array of host IDs of the peers for which the Peer information is to be retrieved.

6.3.11.2 AqRealmPrbReq

The structure and parameter description for AqRealmInfo are:

```
typedef  struct _aqRealmPrbReq
{
   U8     nmbRealm;                          /* Number of Realms */
   LaqStr realmName[LAQ_MAX_NUM_OF_REALMS]; /* Array of realms */
}AqRealmPrbReq;
```

Table 6-72: Parameters of AqRealmPrbReq

Parameters	Description
nmbRealm	Number of realm for which the realm information is to be retrieved.
realmName	Array of realm name for which the realm information is to be retrieved.

6.3.11.3 AqPeerPrbCfm

The structure format and parameter description for **AqPeerPrbCfm** are:

```
typedef struct _aqPeerPrbCfm
{
   U8            nmbPeers;      /* Number of Peers */
   LaqPeerEntry  *peerEntry;    /* Peer Entry */
}AqPeerPrbCfm;
```

Table 6-73: Parameters of AqPeerPrbCfm

Parameters	Description
`nmbPeers`	Number of peers present in the Peer table.
`peerEntry`	Pointer to probed peer entry.

6.3.11.3.1 LaqPeerEntry

This structure contains the peer information to be returned in Probe confirm. The structure format and parameter description for **`LaqPeerEntry`** are:

```
typedef struct _laqPeerEntry
{
    LaqStr   hostId;          /* Host ID */
    S16      cntxId;          /* Context ID */
    LaqStr   realmName;       /* Realm Name */
    TknU32   iConnId;         /* I-Conn ID */
    TknU32   rConnId;         /* R-Conn ID */
    UConnId  iSuConnId;       /* I-SuConn ID */
    UConnId  rSuConnId;       /* R-suConn ID */
    U8       peerState;       /* Peer state */
    U8       peerType;        /* Peer type */
    U16      pmqSize;         /* PMQ Size */
}LaqPeerEntry;
```

Table 6-74: Parameters of LaqPeerEntry

Parameters	Description
`hostId`	This field specifies the host ID of the peer.
`cntxId`	This field specifies the context ID associated with the peer which is being used for TLS configuration.
`realmName`	This field specifies the realm name to which the peer belongs.
`iConnId`	This field specifies the initiating connection ID.
`rConnId`	This field specifies the responding connection ID.
`iSuConnId`	This field specifies the initiating connection ID for incoming connections.
`rSuConnId`	This field specifies the responding connection ID for incoming connections.
`peerState`	This field specifies the state of the peer.
`pmqSize`	This field specifies the size of pending message queue. This size specifies only the application specific pending requests.

Parameters	Description
peerType	This field specifies the type of the peer. Possible values are: • **LAQ_PEER_STATIC** - For Static peers. • **LAQ_PEER_DYNAMIC** - For Dynamic peers. • **LAQ_PEER_UNKNOWN** - For unknown peers.

6.3.11.4 AqRealmPrbCfm

The structure format and parameter description for **AqRealmPrbCfm** are:

```
typedef struct _aqRealmPrbCfm
{
   U8             nmbRealms;    /* Number of realms */
   AqRealmInfo    *realmInfo;   /* Pointer to probed realm info */
}AqRealmProCfm;
```

Table 6-75: Parameters of AqRealmPrbCfm

Parameters	Description
nmbRealms	Number of Realms present in the Realm table.
realmInfo	Pointer to probed realm info. Refer to Section 6.3.1.6.1, "AqRealmInfo".

6.3.11.5 AqAvpPrbCfm

AVP Probe returns the array of AVP entries present in AVP Dictionary. For group AVPs, memory is allocated for the grouped AVP and the AVP Entry present within group AVP. Layer Manager has to free the memory allocated for group AVPs. The structure format and parameter description for **AqAvpPrbCfm** are:

```
typedef struct _aqAvpPrbCfm
{
   U8           nmbAvp;            /* Number of AVP */
   LaqAvpEntry *avpEntry;          /* AVP Entry */
}AqAvpPrbCfm;
```

Table 6-76: Parameters of AqAvpPrbCfm

Parameters	Description
nmbAvp	Number of AVPs present in the AVP Dictionary.
avpEntry	Pointer to AVP entry. Refer to Section 6.3.1.7.1, "LaqAvpEntry".

6.3.11.6 AqDmPrbCfm

The structure format and parameter description for **AqDmPrbCfm** are:

```
typedef struct _aqDmPrbCfm
{
   U8          nmbDmCmd;      /* Number of cmds */
   AqDmEntry   *dmEntry;      /* List of cmds */
}AqDmPrbCfm;
```

Table 6-77: Parameters of AqDmProCfm

Parameters	Description
nmbCmd	Number of commands present in DM Dictionary.
dmEntry	Pointer to Diameter message entries Refer to Section 6.3.1.8.1, "AqDmEntry".

6.3.12 Probe Confirm

Name	Probe Confirm. The **`AqMiLaqPrbCfm`** is the function invoked in Diameter. The **`SmMiLaqPrbCfm`** is the function that processes this confirm primitive in the layer manager.
Direction	Diameter to layer manager
Supplied	Yes
Response	NA

Primitives

`AqMiLaqProCfm`

`SmMiLaqProCfm`

Description

Diameter uses this primitive to return the solicited status information to the layer manager. Figure 6-12 shows the data flow for the management-probe procedure.

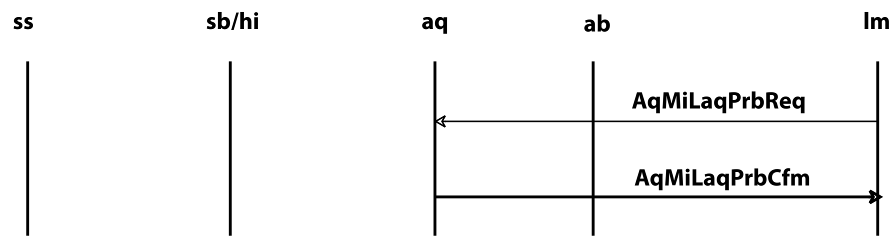


Figure 6-12: Management – Probe procedure

Synopsis

```
PUBLIC S16 AqMiLaqPrbCfm (pst, cfm)
Pst       *pst;
AqMngmt   *cfm;
```

Parameters

`pst`

Refer to Section 1.8.1, "Post."

`cfm`

This parameter represents a pointer to the management structure described in Section 6.2.2, "AqMngmt." The solicited status confirm-specific parameters are described in the next sections.

In the **`Header`** of the status request, the values of the relevant fields are set as shown:

Field	Allowable Values

`elmId.elmnt`	**`STAQPEER : Peer Entry`**
	`STAQALLPEER : All Peer Entry`
	`STAQREALM : Realm Entry`
	`STAQALLREALM : All Realm Entry`
	`STAQAVP : AVP Dictionary`
	`STAQDM : DM Dictionary`

7

Data Structures

This section contains the list and definition of Data structures referred in the previous sections.

7.1 TmrCfg Structure

The format and parameter description of TmrCfg are:

```
typedef struct tmrCfg
{
   Bool enb;   /* Enable/Disable */
   U16  val;   /* Value */
}TmrCfg;
```

Table 7-1: Parameters of TmrCfg

Parameter	Description
`enb`	This field specifies to enable or disable the timer.
`val`	This field specifies the timer value.

7.2 CmNetAddr Structure

The format and parameter description of CmNetAddr are:

```
typedef struct cmNetAddr
{
   U8 type;
   union
   {
      CmIpv4NetAddr ipv4NetAddr; /* IPV4 network address */
      CmIpv6NetAddr ipv6NetAddr; /* IPV6 network address */
   }u;
}CmNetAddr;
```

Table 7-2: Parameters of CmNetAddr

Parameter	Description
`type`	This field specifies the type of network addresses to be used. Allowable values are: • **`CM_NETADDR_NOTPRSNT`**: Network address parameters not present. • **`CM_NETADDR_IPV4`**: An IP version 4 address. • **`CM_NETADDR_IPV6`**: An IP version 6 address.
`ipv4NetAddr`	This parameter is used to represent the IP version 4 network address.
`ipv6NetAddr`	This parameter is used to represent the IP version 6 network address.

Note: Refer to *HIT Interface Service Definition* Section 3.5.1 "Network Address".

7.3 SctNetAddrLst Structure

The format and parameter description of **`SrcNetAddrLst`** is:

```
typedef struct sctNetAddrLst
{
   U8        nmb;                   /* Number of network addr */
   CmNetAddr nAddr[MAX_TPT_ADDRS];  /* Network Address list */
}SctNetAddrLst;
```

Table 7-3: Parameters of SctNetAddrLst

Parameter	Description
`nmb`	This field specifies the number of network addresses in list.
`nAddr`	This field specifies the list of network addresses.

Note: Refer to *SCT Interface Service Definition* Section 3.5.2 "Network Address List".

7.4 CmTptParam Structure

The format and parameter description of **`cmTptParam`** are:

```
typedef struct cmTptParam
{
   U8 type;                        /* Type of transport */
   union
   {
      CmSockParam sockParam;      /* Socket Parameters */
      #ifdef CM_ALL
         AalConParam aalParam;   /* AAL Connection Parameters */
      #endif
      #ifdef CM_TLS
         TlsTptParam tlsParam;   /* TLS Connection Parameters */
      #endif
   }u;
}CmTptParam;
```

Table 7-4: Parameters of CmTptParam

Parameter	Description
`type`	Identifies the specific type of transport parameters required for the connection. Allowable values: • **`CM_TPTPARAM_NOTPRSNT`**: Specifies that this parameter does not have valid information and must be ignored. • **`CM_TPTPARAM_SOCK`**: Specifies that the parameters are for socket interface operations.
`sockParam`	Specifies the values for socket interface operation.
`tlsParam`	Specifies the values for TLS operation.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3 "Transport Parameters".

7.5 CmSockParam Structure

The format and parameter description of **`cmSockParam`** are:

```
typedef struct cmSockParam
{
   U8          listenQSize;    /* Listen Queue Size */
   U8          numOpts;        /* Number of Socket options */
   CmSockOpts  sockOpts[CM_MAX_SOCK_OPTS]; /* Socket options */
}CmSockParam;
```

Table 7-5: Parameters of CmSockParam

Parameter	Description
`listenQSize`	Specifies the number of backlog connections to support for the TCP server. Allowable values: 0 to 5.
`numOpts`	Indicates the number of socket options specified in the sockOpts array.
`sockOpts`	Specifies the individual socket options for the transport connection. The value of **`CM_MAX_SOCK_OPTS`** is 10.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1 "Transport Socket Parameters".

7.6 CmSockOpts Structure

The format and parameter description of **`cmSockOpts`** are:

```
typedef struct cmSockOpts
{
   U32   level;         /* level of the socket options */
   U32   option;        /* option name */
   union                /* option parameters */
   {
      #ifdef IPV6_SUPPORTED
         CmNetMCastInf6 mCastInfo6; /* IPV6 multicast
                                       information*/
         CmIpv6NetAddr  infAddr6;   /* IPV6 multicast outgoing
                                       intf */
      #endif
      CmNetCastInf mCastInfo;    /* IPV4 multicast information */
      CmNetAddr    mCastAddr;    /* multicast address */
      CmNetAddr    lclAddr;      /* local outgoing interface */
      U32          value;        /* option value */
   }optVal;
}CmSockOpts;
```

Table 7-6: Parameters of CmSockOpts

Parameter	Description
`level`	Specifies at the level to which the socket option in the specified transport stack applies. Allowable values: • **`CM_SOCKOPT_LEVEL_IP`**: Applies to the IP protocol level in the transport stack. • **`CM_SOCKOPT_LEVEL_TCP`**: Applies to the TCP protocol level in the transport stack. • **`CM_SOCKOPT_LEVEL_SOCKET`**: Applies to the socket code in the transport stack.

Table 7-6: Parameters of CmSockOpts

Parameter	Description
`option`	Specifies the socket option name. Allowable values • **`CM_SOCKOPT_OPT_BLOCK`**: Makes a socket non blocking. • **`CM_SOCKOPT_OPT_REUSEADDR`**: Allows the local addresses to be reused. • **`CM_SOCKOPT_OPT_RX_BUF_SIZE`**: Changes the size of socket receive buffer. • **`CM_SOCKOPT_OPT_TX_BUF_SIZE`**: Changes the size of socket transmit buffer. • **`CM_SOCKOPT_OPT_CP_NODELAY`**: Disables the TCP Nagle algorithm. • **`CM_SOCKOPT_OPT_ADD_MCAST_MBR`**: Joins a multicast group. • **`CM_SOCKOPT_OPT_DRP_MCAST_MBR`**: Leaves a multicast group. • **`CM_SOCKOPT_IP_HDRINCL`**: Setting this option results in TUCL constructing the IP Header for the data sent on this socket.
`mCastInfo6`	Refer to *HIT Interface Service Definition* Section 3.5.3.1.1.2.
`infAddr6`	Refer to *HIT Interface Service Definition* Section 3.5.3.1.3.
`mCastInfo`	Refer to *HIT Interface Service Definition* Section 3.5.3.1.1.1.
`mCastAddr`	Refer to *HIT Interface Service Definition* Section 3.5.3.1.1.2.
`lclAddr`	Refer to *HIT Interface Service Definition* Section 3.5.3.1.1.
`value`	Option value.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1.1 "Transport Socket Options".

7.7 CmIcmpFilter Structure

The format and parameter description of **`CmIcmpFilter`** are:

```
typedef struct cmIcmpFilter
{
   U8  type;                            /* ICMP version */
   union
   {
      CmIcmpv4Filter icmpv4Filter;    /* ICMPv4 Filter params */
      CmIcmpv6Filter icmpv6Filter;    /* ICMPv6 Filter params */
   }u;
}CmIcmpFiler;
```

Table 7-7: Parameters of CmIcmpFilter

Parameter	Description
`type`	Specifies the version of ICMP. Possible values are: • **`CM_ICMP_NO_FILTER`** • **`CM_ICMPVER4_FILTER`** • **`CM_ICMPVER6_FILTER`**
`icmpv4Filter`	This structure specifies the ICMP V4 filter parameters.
`icmpv6Filter`	This structure specifies the ICMP V6 filter parameters.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1.1 "Transport Socket Options".

7.8 CmIcmpv4Filter Structure

The format and parameter description of **`CmIcmpv4Filter`** are:

```
typedef struct cmIcmpv4Filter
{
   U8            icmpMsgFlag;  /* Flag for ICMP messages */
   U8            allMsg;       /* All messages to be send */
   U8            protocol;     /* Filtering protocol */
   U8            num;          /* Number of specified types */
   CmIcmpError   icmpError[CM_MAX_ICMP_ERROR];
                               /* ICMP error types and codes */
}CmIcmpv4Filter;
```

Table 7-8: Parameters of CmIcmpv4Filter

Parameter	Description
`icmpMsgFlag`	This flag specifies whether the application wants to listen to ICMP messages. Possible values are: • **`TRUE`**: Application wants to listen to ICMP message. • **`FALSE`**: Application does not want to listen to ICMP messages.
`allMsg`	Indicates whether the application wants to listen to all ICMP messages or filtering needs to be done. • **`TRUE`**: Send all ICMP messages without filtering. • **`FALSE`**: Do not send all ICMP messages.
`protocol`	Specifies protocol for which the filtering needs to be done. Possible values are: • **`CM_INET_PROTO_STCP`**: Protocol type is SCTP. • **`CM_INET_PROTO_RAW`**: Protocol type is RAW.
`num`	Specifies the number of valid entries in cmIcmpError array.
`CmIcmpError`	Represent the error type and error codes which an application can specify for filtering. The value for **`CM_MAX_ICMP_ERROR`** is 5.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1.3 "ICMP Filter Parameters".

7.9 CmIcmpv6Filter Structure

The format and parameter description of **`CmIcmpv6Filter`** are:

```
typedef struct cmIcmpv6Filter
{
   U8           icmpMsgFlag;  /* Flag for ICMP messages */
   U8           allMsg;       /* All messages to be send */
   U8           protocol;     /* Filtering protocol */
   U8           num;          /* Number of specified types */
   CmIcmpError  icmpError[CM_MAX_ICMP_ERROR];
                              /* ICMP error types and codes */
}CmIcmpv6Filter;
```

Table 7-9: Parameters of CmIcmpv4Filter

Parameter	Description
`icmpMsgFlag`	This flag specifies whether the application wants to listen to ICMP messages. Possible values are: • **`TRUE`**: Application wants to listen to ICMP message. • **`FALSE`**: Application does not want to listen to ICMP messages.
`allMsg`	Indicates whether the application wants to listen to all ICMP messages or filtering needs to be done. • **`TRUE`**: Send all ICMP messages without filtering. • **`FALSE`**: Do not send all ICMP messages.
`protocol`	Specifies protocol for which the filtering needs to be done. Possible values are: • **`CM_INET_PROTO_STCP`**: Protocol type is SCTP. • **`CM_INET_PROTO_RAW`**: Protocol type is RAW.
`num`	Specifies the number of valid entries in cmIcmpError array.
`CmIcmpError`	Represent the error type and error codes which an application can specify for filtering. The value for **`CM_MAX_ICMP_ERROR`** is 5.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1.3 "ICMP Filter Parameters".

7.10 CmIcmpError Structure

The format and parameter description of **`CmIcmpError`** are:

```
typedef struct cmIcmpError
{
   U8   errType;       /* Type of the ICMP error message */
   U32  errCodeMask;   /* ICMP Code mask */
}CmIcmpError;
```

Table 7-10: Parameters of CmIcmpError

Parameter	Description
`errType`	Represents all the error type for an ICMP message.
`errCodeMask`	This is a bit mask for all the error codes corresponding to a particular error type.

Note: Refer to *HIT Interface Service Definition* Section 3.5.3.1.4 "ICMP Error Types and Codes".

8

COTS Configuration

This section describes the various possible combination of COTS configuration.

8.1 Diameter User and Diameter LC, Diameter and TUCL LC

In this scenario, Diameter user and Diameter as well as Diameter and its lower layer TUCL both are loosely coupled.

8.1.1 Changes in Makefile

The changes in makefile are as follows:

- Enable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Enable **`LCHIUIHIT`** in **`HIOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.1.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in ‘pst’ structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LC`** (that is, 0).

8.1.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_LC`** (that is, 0). This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#endif
    uSapCfg->selector = LAQ_SEL_TC_AB;
----------------------------------------------------------
```

8.1.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`** (that is, 0). This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.1.5 TUCL USAP Configuration

As TUCL and Diameter are loosely coupled, the selector value in USAP configuraiton of TUCL must be assinged to **`HI_UISEL_LC`** (that is, 0). This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
----------------------------------------------------------
```

8.1.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-------------------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
---------------------------------------------------------------
```

8.2 User and Diameter TC, Diameter and TUCL TC

In this scenario, Diameter user and Diameter as well as Diameter and its lower layer TUCL both are tightly coupled.

8.2.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Disable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Disable **`LCHIUIHIT`** in **`HIOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.2.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are tightly coupled, the selector value in ‘pst’ structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_TC_AB`**.

8.2.3 Diameter USAP Configuration

As the Diameter and Diameter user are tightly coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_TC_AB`**. This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.2.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are tightly coupled, the selector value in LSAP configuration of Diameter must be assinged to **`LAQ_SEL_TC_HIT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.2.5 TUCL USAP Configuration

As TUCL and Diameter are tightly coupled, the selector value in USAP configuration of TUCL must be assinged to **`HI_UISEL_TC`**. This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
----------------------------------------------------------
```

8.2.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
---------------------------------------------------------
```

8.3 Diameter User and DIameter LC, DIameter and TUCL TC

In this scenario, Diameter user and Diameter are loosely coupled where Diameter and its lower layer TUCL are tigtly coupled

8.3.1 Changes in Makefile

The changes in makefile are as follows:

- Enable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Disable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Disable **`LCHIUIHIT`** in **`HIOPT`**S
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.3.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LC`** (that is, 0).

8.3.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_LC`** (that is, 0). This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.3.4 Diameter LSAP Configuration

As the Diameter and its lower layer TUCL are tightly coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_TC_HIT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.3.5 TUCL USAP Configuration

As the TUCL and Diameter are tightly coupled, the selector value in USAP configuraiton of TUCL must be assinged to **`HI_UISEL_TC`**. This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
-----------------------------------------------------------
```

8.3.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
--------------------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
---------------------------------------------------------------
```

8.4 Diameter User and Diameter TC, Diameter and TUCL LC

In this scenario, Diameter user and Diameter are tightly coupled where as Diameter and its lower layer TUCL are loosely coupled.

8.4.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Enable **`LCHIUIHIT`** in **`HIOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.4.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are tightly coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_TC_AB`** (that is, 0).

8.4.3 Diameter USAP Configuration

As the Diameter and Diameter user are tightly coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_TC_AB`** (that is, 0). This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
---------------------------------------------------------
```

8.4.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
---------------------------------------------------------
```

8.4.5 TUCL USAP Configuration

As TUCL and Diameter are loosely coupled, the selector value in USAP configuraiton of TUCL must be assinged to **`HI_UISEL_LC`**. This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
---------------------------------------------------------
```

8.4.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
----------------------------------------------------------
```

8.5 Diameter User and Diameter LWLC, Diameter and TUCL LC

In this scenario, Diameter user and Diameter are light weight loosely coupled where as Diameter and its lower layer TUCL are loosely coupled.

8.5.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Enable **`LCHIUIHIT`** in **`HIOPTS`**
- Enable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.5.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are light weight loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LWLC_AB`** (that is, 2).

8.5.3 Diameter USAP Configuration

As the Diameter and Diameter user are light weight loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_LWLC_AB`** (that is, 2). This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.5.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.5.5 TUCL USAP Configuration

As TUCL and Diameter are loosely coupled, the selector value in USAP configuraiton of TUCL must be assinged to **`HI_UISEL_LC`**. This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
----------------------------------------------------------
```

8.5.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-------------------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
--------------------------------------------------------------
```

8.6 Diameter User and Diameter LWLC, Diameter and TUCL TC

In this scenario, Diameter user and Diameter are light weight loosely coupled where as Diameter and its lower layer TUCL are tightly coupled.

8.6.1 Changes in Make File

The changes in make file are as follows:

- Disable **LCABLIAQU** and **LCAQUIAQU** in **PRDACCOPTS**
- Disable **LCAQLIHIT** in **PRDACCOPTS**
- Disable **LCHIUIHIT** in **HIOPTS**
- Enable **LWLCABLIAQU** and **LWLCAQUIAQU** in **PRDACCOPTS**

8.6.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are light weight loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **LAQ_SEL_LWLC_AB** (that is, 2).

8.6.3 Diameter USAP Configuration

As the Diameter and Diameter user are light weight loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **LAQ_SEL_LWLC_AB** (that is, 2). This can be changed in the function **aqUSAPCfg** in file **aq_acc_acct.c**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.6.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are tightly coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_TC_HIT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.6.5 TUCL USAP Configuration

As TUCL and Diameter are tightly coupled, the selector value in USAP configuraiton of TUCL must be assinged to **`HI_UISEL_TC`**. This can be changed in the function **`hiSapConfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCHIUIHIT
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_LC;
#else
    cfg.t.cfg.s.hiSap.uiSel = HI_UISEL_TC;
#endif
----------------------------------------------------------
```

8.6.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
----------------------------------------------------------
```

8.7 Diameter User and Diameter LC, Diameter and SCTP LC

In this scenario, Diameter user and Diameter as well as Diameter and its lower layer SCTP are loosely coupled. SCTP and TUCL are also loosely coupled.

8.7.1 Changes in Makefile

The changes in makefile are as follows:

- Enable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLISCT`** in **`PRDACCOPTS`**
- Enable **`LCSCT`** , **`LCSBUISCT`** and **`LCSBLIHIT`** in **`SBOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**

8.7.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in ‘pst’ structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LC`** (that is, 0).

8.7.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_LC`** (taht is, 0). This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
-----------------------------------------------------------
```

8.7.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
----------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
----------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
----------------------------------------------------------
```

8.7.5 SCTP USAP Configuration

As SCTP and Diameter are loosely coupled, the selector value in USAP configuration of SCTP must be assigned to 0.

8.7.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;
-------------------------------------------------------------
```

8.8 Diameter User and Diameter TC, Diameter and SCTP TC

In this scenario, Diameter user and Diameter as well as Diameter and its lower layer SCTP are tightly coupled. SCTP and TUCL are loosely coupled.

8.8.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **LCABLIAQU** and **LCAQUIAQU** in **PRDACCOPTS**
- Disable **LCAQLISCT** in **PRDACCOPTS**
- Disable **LCSCT** and **LCSBUISCT** in **SBOPTS**
- Enable **LCSBLIHIT** in **SBOPTS**
- Disable **LWLCABLIAQU** and **LWLCAQUIAQU** in **PRDACCOPTS**
- Enable **AQ_SCTP** and **AQ** in **PRDACCOPTS**

8.8.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are tightly coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **LAQ_SEL_TC_AB**.

8.8.3 Diameter USAP Configuration

As the Diameter and Diameter user are tightly coupled, the selector value in USAP configuration of Diameter must be assinged to **LAQ_SEL_TC_AB**. This can be changed in the function **aqUSAPCfg** in file **aq_acc_acct.c**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.8.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are tightly coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_TC_SCT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
-----------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
-----------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
-----------------------------------------------------------
```

8.8.5 SCTP USAP Configuration

As SCTP and Diameter are tightly coupled, the selector value in USAP configuration of SCTP must be assigned to 10.

8.8.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;
-----------------------------------------------------------
```

8.9 Diameter User and Diameter LC, Diameter and SCTP TC

In this scenario, Diameter user and Diameter are loosely coupled where as Diameter and its lower layer SCTP are tightly coupled. SCTP and TUCL are loosely coupled.

8.9.1 Changes in Makefile

The changes in makefile are as follows:

- Enable **LCABLIAQU** and **LCAQUIAQU** in **PRDACCOPTS**
- Disable **LCAQLISCT** in **PRDACCOPTS**
- Disable **LCSCT** and **LCSBUISCT** in **SBOPTS**
- Enable **LCSBLIHIT** in **SBOPTS**
- Disable **LWLCABLIAQU** and **LWLCAQUIAQU** in **PRDACCOPTS**
- Enable **AQ_SCTP** and **AQ** in **PRDACCOPTS**

8.9.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **LAQ_SEL_LC** (that is, 0).

8.9.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **LAQ_SEL_LC** (that is, 0). This can be changed in the function **aqUSAPCfg** in file **aq_acc_acct.c**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.9.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are tightly coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_TC_SCT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
-----------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
-----------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
-----------------------------------------------------------
```

8.9.5 SCTP USAP Configuration

As SCTP and Diameter are tightly coupled, the selector value in USAP configuration of SCTP must be assigned to 10.

8.9.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
-----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;

-----------------------------------------------------------
```

8.10 Diameter User and Diameter TC, Diameter and SCTP LC

In this scenario, Diameter user and Diameter are tightly coupled where as Diameter and SCTP are loosely coupled. SCTP and TUCL are loosely coupled.

8.10.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLISCT`** in **`PRDACCOPTS`**
- Enable **`LCSCT`** and **`LCSBUISCT`** in **`SBOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`AQ_SCTP`** in **`PRDACCOPTS`**
- Enable **`LCSBLIHIT`** in **`SBOPTS`**

8.10.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are tightly coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_TC_AB`**.

8.10.3 Diameter USAP Configuration

As the Diameter and Diameter user are tightly coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_TC_AB`**. This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.10.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
---------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
---------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
---------------------------------------------------------
```

8.10.5 SCTP USAP Configuration

As SCTP and Diameter are loosely coupled, the selector value in USAP configuration of SCTP must be assigned to 0.

8.10.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------

peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;

---------------------------------------------------------
```

8.11 Diameter User and Diameter LWLC, Diameter and SCTP LC

In this scenario, Diameter user and Diameter are light weight loosely coupled where as Diameter and its lower layer SCTP are loosely coupled. SCTP and TUCL are loosely coupled.

8.11.1 Changes in Makefile

The changes in makefile are as follows:

- Disable **LCABLIAQU** and **LCAQUIAQU** in **PRDACCOPTS**
- Enable **LCAQLISCT** in **PRDACCOPTS**
- Enable **LCSCT** and **LCSBUISCT** in **SBOPTS**
- Enable **LWLCABLIAQU** and **LWLCAQUIAQU** in **PRDACCOPTS**
- Enable **AQ_SCTP** in **PRDACCOPTS**
- Enable **LCSBLIHIT** in **SBOPTS**

8.11.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are light weight loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **LAQ_SEL_LWLC_AB**.

8.11.3 Diameter USAP Configuration

As the Diameter and Diameter user are light weight loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **LAQ_SEL_LWLC_AB**. This can be changed in the function **aqUSAPCfg** in file **aq_acc_acct.c**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.11.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
---------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
---------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
---------------------------------------------------------
```

8.11.5 SCTP USAP Configuration

As SCTP and Diameter are loosely coupled, the selector value in USAP configuration of SCTP must be assigned to 0.

8.11.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;
---------------------------------------------------------
```

8.12 Diameter User and Diameter LWLC, Diameter and SCTP TC

In this scenario, Diameter user and Diameter are light weight loosely coupled where as Diameter and its lower layer SCTP are tightly coupled. SCTP and TUCL are loosely coupled.

8.12.1 Changes in Make File

The changes in make file are as mentioned below:

- Disable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Disable **`LCAQLISCT`** in **`PRDACCOPTS`**
- Disable **`LCSCT`** and **`LCSBUISCT`** in **`SBOPTS`**
- Enable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`AQ_SCTP`** and **`AQ`** in **`PRDACCOPTS`**
- Enable **`LCSBLIHIT`** in **`SBOPTS`**

8.12.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are light weight loosely coupled, the selector value in ‘pst’ structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LWLC_AB`**.

8.12.3 Diameter USAP Configuration

As the Diameter and Diameter user are light weight loosely coupled, the selector value in USAP configuration of Diameter must be assinged to **`LAQ_SEL_LWLC_AB`**. This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
----------------------------------------------------------
```

8.12.4 Diameter LSAP Configuration

As Diameter and its lower layer SCTP are tightly coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_TC_SCT`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLISCT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_SCT;
#endif
----------------------------------------------------------
```

For SCTP transport, transport type must be configured as:

```
----------------------------------------------------------
#ifdef AQ_SCTP
    lSapCfg->transType = LAQ_TRANS_TYPE_SCTP_IPSEC;
#endif
----------------------------------------------------------
```

8.12.5 SCTP USAP Configuration

As SCTP and Diameter are tightly coupled, the selector value in USAP configuration of SCTP must be assigned to 10.

8.12.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_SCTP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_SCTP;
----------------------------------------------------------
```

8.13 Diameter User and Diameter LC, Diameter and TUCL LC with TLS support

In this scenario, both Diameter user and Diameter as well as Diameter and its lower layer TUCL are loosely coupled along with TLS support.

8.13.1 Changes in Makefile

The changes in makefile are as follows:

- Enable **AQ_TLS**, **HI_CMPTBL_FLAG**, **HI_TLS**, **HI_TCP_TLS**, **HI_TLS_EXT_CALLBACK**, **CM_TLS, HITV2**, **HI_REL_1_4**, **OPENSSL_NO_KBR5** in **HITOPTS**
- Enable **LCABLIAQU** and **LCAQUIAQU** in **PRDACCOPTS**
- Enable **LCAQLIHIT** in **PRDACCOPTS**
- Enable **LCHIUIHIT** in **HIOPTS**
- Disable **LWLCABLIAQU** and **LWLCAQUIAQU** in **PRDACCOPTS**.

8.13.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in 'pst' structure of LSAP configuration of Diameter user must be assigned to **LAQ_SEL_LC**.

8.13.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assigned to **LAQ_SEL_LC**. This can be changed in the function **aqUSAPCfg** in file **aq_acc_acct.c**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
---------------------------------------------------------
```

8.13.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

Also to enable TLS, transport type must be configured as:

```
----------------------------------------------------------
#ifndef CM_TLS
     lSapCfg->transType = LAQ_TRANS_TYPE_TCP_TLS;
#else
     lSapCfg->transTpye = LAQ_TRANS_TYPE_TCP_IPSEC;
#endif
----------------------------------------------------------
```

8.13.5 TUCL USAP Configuration

As TUCL and Diameter are loosely coupled, the selector value in USAP configuration of TUCL must be assigned to 0.

8.13.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
----------------------------------------------------------
```

Also to support TLS, security must be configured as:

```
----------------------------------------------------------
#ifdef CM_TLS
     peerCfg->peerInfo[0].security = LAQ_NO_INBAND_SEC;
#else
     peerCfg->peerInfo[0].security = LAQ_SEC_TLS;
----------------------------------------------------------
```

8.14 Diameter User and Diameter LC, Diameter and TUCL LC with IPV6 support

In this scenario, both Diameter user and Diameter as well as Diameter and its lower layer TUCL are loosely coupled along with IPV6 support.

8.14.1 Changes in Make File

The changes in make file are as mentioned below:

- Enable **`IPV6_SUPPORTED`** in **`PRDACCOPTS`**
- Enable **`LCABLIAQU`** and **`LCAQUIAQU`** in **`PRDACCOPTS`**
- Enable **`LCAQLIHIT`** in **`PRDACCOPTS`**
- Enable **`LCHIUIHIT`** in **`HIOPTS`**
- Disable **`LWLCABLIAQU`** and **`LWLCAQUIAQU`** in **`PRDACCOPTS`**.

8.14.2 Diameter User LSAP Configuration

As the Diameter user and Diameter are loosely coupled, the selector value in ‘pst’ structure of LSAP configuration of Diameter user must be assigned to **`LAQ_SEL_LC_AB`**.

8.14.3 Diameter USAP Configuration

As the Diameter and Diameter user are loosely coupled, the selector value in USAP configuration of Diameter must be assigned to **`LAQ_SEL_LC`**. This can be changed in the function **`aqUSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
---------------------------------------------------------
#ifdef LCAQUIAQU
      uSapCfg->selector = LAQ_SEL_LC;
#elif LWLCAQUIAQU
     uSapCfg->selector = LAQ_SEL_LWLC_AB;
#else
    uSapCfg->selector = LAQ_SEL_TC_AB;
#endif
---------------------------------------------------------
```

8.14.4 Diameter LSAP Configuration

As Diameter and its lower layer TUCL are loosely coupled, the selector value in LSAP configuraiton of Diameter must be assinged to **`LAQ_SEL_LC`**. This can be changed in the function **`aqLSAPCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
#ifdef LCAQLIHIT
    lSapCfg->selector = LAQ_SEL_LC;
#else
    lSapCfg->selector = LAQ_SEL_TC_HIT;
#endif
----------------------------------------------------------
```

8.14.5 TUCL USAP Configuration

As TUCL and Diameter are loosely coupled, the selector value in USAP configuration of TUCL must be assigned to 0.

8.14.6 Peer Configuration

Assign protocol field of peer in peer configuration to **`LAQ_PROT_TCP`**. This can be changed in the function **`aqPeerCfg`** in file **`aq_acc_acct.c`**. The code snippet for the same is as follows:

```
----------------------------------------------------------
peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
----------------------------------------------------------
```

For IPV6 support, configure peer address as global IPV6 address.

```
----------------------------------------------------------
/*********************************************
        Configuring Peer Address
*********************************************/

peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.IpAddr.
        type = CM_NETADDR_IPV6;

peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.
IpAddr.u.ipv6NetAddr = IPV6_PEER_ADDR;



/**********************************************
```

```
         Configuring Self Address
************************************************/

peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.
       type = CM_NETADDR_IPV6;
peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.u.
       ipv6NetAddr = IPV6_SELF_ADDR;

----------------------------------------------------------
```

Appendix A

Diameter Sample Configuration

This section describes the pseudo code for configuring Diameter Layer. It includes:

- General Configuration - Section 6.3.1.1, "General Configuration".
- Upper Sap Configuration - Section 6.3.1.2, "Upper SAP Configuration".
- Lower Sap Configuration - Section 6.3.1.3, "Lower SAP Configuration".
- Protocol Configuration - Section 6.3.1.4, "Protocol Configuration".
- Peer Configuration - Section 6.3.1.5, "Peer Configuration".
- Realm Configuration - Section 6.3.1.6, "Realm Configuration".
- AVP Configuration - Section 6.3.1.7, "AVP Configuration".
- DM Configuration - Section 6.3.1.8, "DM Configuration".

A.1 General Configuration

```
/****************************************************************
                     Local Variable Declaration

****************************************************************/
    AqMngmt aqMngmt;
    Pst pst;
    U16   dstProcId;
    U16   srcProcId;
    AqGenCfg    *genCfg;

/****************************************************************
                      Variable initialization

****************************************************************/
    cmMemset((U8 *) &aqMngmt,0, sizeof(AqMngmt));
    cmMemset((U8 *) &pst, 0, sizeof(Pst));
    dstProcId   =   1;
    srcProcId   =   1;
    genCfg   = &(aqMngmt.u.cfg.u.gen);

/****************************************************************
                    Sample General configuration

****************************************************************/
    genCfg->nmbUSaps         = 2;
    genCfg->nmbLSaps         = 4;
    genCfg->sizePmq          = 100;
    genCfg->firmwareRev      = 0;
    genCfg->orgStateId       = 0;
    genCfg->nmbPeer          = 5;
    genCfg->nmbRealm         = 3;
    cmMemset((U8 *) &genCfg->hostId, 0, sizeof(LaqStr));

/****************************************************************
                        Configuring Host ID

****************************************************************/
#ifdef CLIENT
    #ifdef AQACC_CMD_LINE
         genCfg->hostId.length      = hostId.hostLen;
         aqStrcpy((S8 *)genCfg->hostId.buff, (S8*)hostId.hostName);
    #else
         genCfg->hostId.length      = 16;
         aqStrcpy((S8 *)genCfg->hostId.buff,"client.example.com");
    #endif
#else
    #ifdef AQACC_CMD_LINE
         genCfg->hostId.length      = hostId.hostLen;
         aqStrcpy((S8 *)genCfg->hostId.buff,(S8*)hostId.hostName);
```

```
     #else
         genCfg->hostId.length     = 13;
         aqStrcpy((S8 *)genCfg->hostId.buff,"server.example.com");
     #endif
#endif

/****************************************************************
                       Configuring Host Realm

****************************************************************/
     cmMemset((U8 *) &genCfg->localRealm, 0, sizeof(LaqStr));
#ifdef AQACC_CMD_LINE
     genCfg->localRealm.length     = cmStrlen((U8*)realm);
     aqStrcpy((S8 *)genCfg->localRealm.buff, realm);
#else
     genCfg->localRealm.length     = 8;
     aqStrcpy((S8 *)genCfg->localRealm.buff, "example.com");
#endif
     cmMemset((U8 *) &genCfg->prodName, 0, sizeof(LaqStr));
     genCfg->prodName.length       = 13;
     aqStrcpy((S8 *)genCfg->prodName.buff, "ccpu-diameter");
     genCfg->nmbWorkerThrds        = 0;
     genCfg->timeRes               = 10;
     genCfg->memUpperThr           = 7;
     genCfg->memLowerThr           = 1;

/****************************************************************
              Configuring Layer Manger Post structure

****************************************************************/
     genCfg->lmPst.dstProcId       = dstProcId;
     genCfg->lmPst.srcProcId       = srcProcId;
     genCfg->lmPst.dstEnt          = ENTSM;
     genCfg->lmPst.srcEnt          = ENTAQ;
     genCfg->lmPst.dstInst         = TSTINST;
     genCfg->lmPst.srcInst         = TSTINST;
     genCfg->lmPst.prior           = PRIOR0;
     genCfg->lmPst.route           = RTESPEC;
     genCfg->lmPst.event           = 0;
     genCfg->lmPst.region          = TSTREG;
     genCfg->lmPst.pool            = TSTPOOL;

/****************************************************************
           Configuring coupling mode between layer and LM

****************************************************************/
#ifdef LCSMAQMILAQ
     genCfg->lmPst.selector        = LAQ_SEL_LC;
#else
     genCfg->lmPst.selector        = LAQ_SEL_TC_SM;
#endif
```

A.2 Lower Sap Configuration

```
/****************************************************************
                     Local Variable Declaration

****************************************************************/
     AqMngmt      aqMngmt;
     Pst          pst;
     U16          dstProcId;
     U16          srcProcId;
     AqLSapCfg     *lSapCfg;

/****************************************************************
                    Local Variable Initialization

****************************************************************/
    cmMemset((U8*)&aqMngmt,0, sizeof(AqMngmt));
    cmMemset((U8*)&pst,0, sizeof(Pst));

    srcProcId = SFndProcId();
    dstProcId = SFndProcId();
    lSapCfg   = &(aqMngmt.u.cfg.u.lSap);

/****************************************************************
                         Configuring Lower SAP

****************************************************************/
    lSapCfg->suId                   = 0;
    lSapCfg->prior                  = PRIOR0;
    lSapCfg->route                  = RTESPEC;
    lSapCfg->dstEntId               = ENTHI;
    lSapCfg->dstInstId              = TSTINST;
#ifdef CM_TLS
    lSapCfg->transpType             = LAQ_TRANS_TYPE_TCP_TLS;
#else
    lSapCfg->transpType             = LAQ_TRANS_TYPE_TCP_IPSEC;
#endif
    lSapCfg->maxBndRetry            = 10;
    lSapCfg->maxSerRetry            = 10;
    lSapCfg->tBndTmr.enb            = 1;
    lSapCfg->tBndTmr.val            = 10;
    lSapCfg->tSerTmr.enb            = 1;
    lSapCfg->tSerTmr.val            = 10;
```

```
/****************************************************************
            Server Configuration based on the transpType

****************************************************************/
    if (lSapCfg->transpType == LAQ_TRANS_TYPE_TCP_IPSEC)
    {
         /*TCP + IPSEC*/
#ifdef CLIENT
     lSapCfg->aqServerCfg.aqTcpCfg.tcpIpsecSerCfg.tcpPort =
                                                    CLIENT_PORT;
#else
     lSapCfg->aqServerCfg.aqTcpCfg.tcpIpsecSerCfg.tcpPort =
                                                    SERVER_PORT;
#endif
     lSapCfg->aqServerCfg.aqTcpCfg.tcpIpsecSerCfg.tcpIpAddr.type =
CM_NETADDR_IPV4;
     lSapCfg->aqServerCfg.aqTcpCfg.tcpIpsecSerCfg.transpParam.u.
              sockParam.listenQSize = 5;
     lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.transpParam.u.
              tlsParam.numOpts     = 0;
#endif /* end of CM_TLS */
    cmMemset((U8*)&lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.
transpParam.u.sockParam, 0, sizeof(CmSockParam));
     }   /* if (lSapCfg->transpType == LAQ_TRANS_TYPE_TCP_IPSEC) */
     else if (lSapCfg->transpType == LAQ_TRANS_TYPE_TCP_TLS)
     {
         /*TCP + TLS*/
         #ifdef CLIENT
         lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpPort =
                                                    SERVER_PORT;
         #else
         lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpPort =
                                                    CLIENT_PORT;
         #endif
         lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.transpParam.
         type = CM_TPTPARAM_TLS;

/*****************************************************************
                     Configuring TLS Parameters

*****************************************************************/
#ifdef CM_TLS
      lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.transpParam.u.
      tlsParam.listenQSize = 5;
      lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.transpParam.u.
      tlsParam.numOpts = 0;
      lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.transpParam.u.
      tlsParam.ctxId = AQ_CTX_ID;
#endif
```

```
     lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpIpAddr.type =
     CM_NETADDR_IPV4;
 #ifdef AQACC_CMD_LINE
     lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpIpAddr.
     u.ipv4NetAddr = hostId.ipAddr;
#else
#ifdef CLIENT
    lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpIpAddr.
    u.ipv4NetAddr = CLIENT_IP_ADDR;
#else
    lSapCfg->aqServerCfg.aqTcpCfg.tcpTlsSerCfg.tcpIpAddr.
    u.ipv4NetAddr = SERVER_IP_ADDR;
#endif
#endif /* AQACC_CMD_LINE */
```

A.3 Upper Sap Configuration

```
/*****************************************************************
                    Local Variable Declaration

*****************************************************************/
     AqMngmt aqMngmt;
     Pst     pst;
     U16     dstProcId;
     U16     srcProcId;

     AqUSapCfg    *uSapCfg;

/*****************************************************************
                  Local Variable Initialization

*****************************************************************/
     cmMemset((U8*)&aqMngmt,0, sizeof(AqMngmt));
     cmMemset((U8*)&pst,0, sizeof(Pst));

     srcProcId = SFndProcId();
     dstProcId = SFndProcId();

     uSapCfg   = &(aqMngmt.u.cfg.u.uSap);

/*****************************************************************
                       Configuring Lower SAP

*****************************************************************/

     uSapCfg->spId                  =  1;
#ifdef LCABLIAQU
     /* Loose Coupling is enabled */
     uSapCfg->selector              =  LAQ_SEL_LC;
#else
     /* Tight coupling mode  */
     uSapCfg->selector              =  LAQ_SEL_TC_AB;
#endif
     uSapCfg->mem.region            =  TSTREG;
     uSapCfg->mem.pool              =  TSTPOOL;
     uSapCfg->priority              =  PRIOR0;
     uSapCfg->route                 =  RTESPEC;
     uSapCfg->vendorId              =  DFLT_VENDOR_ID;
     uSapCfg->appId                 =  DFLT_ACCT_APPID;
     uSapCfg->suppVenId             =  0;
     uSapCfg->suppVendorId          =  0;
     uSapCfg->appType               =  AQ_ACCT_TYPE;
```

A.4 Protocol Configuration

```
/****************************************************************
                    Local Variable Declaration

****************************************************************/
     AqMngmt     aqMngmt;
    Pst          pst;
    U16          dstProcId;
    U16          srcProcId;

    AqProtCfg      *protCfg;
    CmTptParam     *tptParam;
    CmSockParam    *sockParam;

/****************************************************************
                      Variable initialization

****************************************************************/
    cmMemset((U8*)&aqMngmt,0, sizeof(AqMngmt));
    cmMemset((U8*)&pst,0, sizeof(Pst));
    srcProcId = SFndProcId();
    dstProcId = SFndProcId();
    protCfg   = &(aqMngmt.u.cfg.u.prot);

/****************************************************************
                    Configuring Protocol Timers

****************************************************************/
      (protCfg->tWatchDogTmr).enb     =   TRUE;
      (protCfg->tWatchDogTmr).val     =   30;

      (protCfg->tBurstBrkTmr).enb     =   TRUE;
      (protCfg->tBurstBrkTmr).val     =   30;

      (protCfg->tPeerStateTmr).enb    =   TRUE;
      (protCfg->tPeerStateTmr).val    =   30;

      (protCfg->tReconnTmr).enb       =   TRUE;
      (protCfg->tReconnTmr).val       =   30;

#ifdef AQ_PEER_DISCOVERY
      (protCfg->tPeerDscTmr).enb      =   TRUE;
      (protCfg->tPeerDscTmr).val      =   30;
      protCfg->peerDscType            =   LAQ_PEER_DSC_SLP;
#endif /* end of AQ_PEER_DISCOVERY */
```

```
/****************************************************************
                        Sample Configuration

****************************************************************/
        protCfg->failover            =   0;
#ifdef CM_TLS
        protCfg->secType             =   LAQ_SEC_TLS;
#else
        protCfg->secType             =   LAQ_NI_INBAND_SEC;
#endif
        protCfg->maxConnRetry        =   3;
        protCfg->localPolicy         =   LAQ_INCOMING_CONN_ACCEPT;
        protCfg->nmbBurstCnt         =   50;

/****************************************************************
                 Transport Parameters Configuration

****************************************************************/
      tptParam = &(protCfg->transParam);
#ifdef CM_TLS
      tptParam->type  =   CM_TPTPARAM_TLS;
#else
      tptParam->type  =   0;
#endif
       if (tptParam->type ==  CM_TPTPARAM_SOCK)
       {
           sockParam = &tptParam->u.sockParam;
           if (sockParam != NULLP)
           {
                  sockParam->listenQSize  =  10;
                  sockParam->numOpts      =  0
           } /* End of NULL check   */
      }
      else if (tptParam->type  ==  CM_TPTPARAM_TLS)
      {
      #ifdef CM_TLS
             tptParam->u.tlsParam.ctxId         =  AQ_CTX_ID;
             tptParam->u.tlsParam.listenQSize   =  10;
             tptParam->u.tlsParam.numOpts       =  0;
      #endif
 }

/****************************************************************
             Configure Optional AVP values for CER/CEA.

****************************************************************/
      protCfg->nmbOptAvps             = 0;
      protCfg->optAvpValue[0].avpCode = 0;
      protCfg->optAvpValue[0].avpType = 0;
```

```
****************************************************************
         Configure the node type (server or client or both)

****************************************************************/
#ifdef CLIENT
      protCfg->nodeType = LAQ_CLIENT;
#else
      protCfg->nodeType = LAQ_SERVER;
#endif
```

A.5 Realm Configuration

```
/*****************************************************************
                        Variable Declaration

*****************************************************************/
   AqMngmt aqMngmt;
   Pst pst;
   U16  dstProcId;
   U16  srcProcId;
   AqRealmCfg    *realmCfg;
   S16           ret;

/*****************************************************************
                   Local Variable Initialization

*****************************************************************/
   ret = ROK;
   cmMemset((U8*)&aqMngmt, 0, sizeof(AqMngmt));
   cmMemset((U8*)&pst, 0, sizeof(Pst));
   srcProcId = SFndProcId();
   dstProcId = SFndProcId();
   realmCfg  = &(aqMngmt.u.cfg.u.realm);

/*****************************************************************
                 Default configuration for Realm 1

*****************************************************************/
   /* Number of realm to be configured */
    realmCfg->nmbRealm = 1;

   /* Allocate memory for RealmInfo */
      SGetSBuf(TSTREG, TSTPOOL, (Data**)&realmCfg->realmInfo,
sizeof(AqRealmInfo));
     cmMemset((U8 *) &realmCfg->realmInfo[0].realmName, 0,
sizeof(LaqStr));

#ifdef AQACC_CMD_LINE
     realmCfg->realmInfo[0].realmName.length = cmStrlen((U8*)realm);
     aqStrcpy((S8 *)realmCfg->realmInfo[0].realmName.buff, realm);
#else
     realmCfg->realmInfo[0].realmName.length = 8;
     aqStrcpy((S8 *)realmCfg->realmInfo[0].realmName.buff,
"example.com");
#endif
     realmCfg->realmInfo[0].nmbApp       = 1;
     realmCfg->realmInfo[0].localAction  = LAQ_ACTION_LOCAL;
```

```
/****************************************************************
                Configuring App ID to Host ID Mapping

****************************************************************/
       /* Allocate memroy for aqAppHostMap */
       SGetSBuf(TSTREG, TSTPOOL,
        (Data**)&realmCfg->realmInfo[0].aqAppHostMap,
       sizeof(LaqAppHostMap));
       realmCfg->realmInfo[0].aqAppHostMap[0].appId=
                                               DFLT_ACCT_APPID;
       realmCfg->realmInfo[0].aqAppHostMap[0].nmbHost= 1;

       /* Allocate memory for hostId */
       SGetSBuf(TSTREG, TSTPOOL,
       (Data**)&realmCfg->realmInfo[0].aqAppHostMap[0].hostId,
       sizeof(LaqStr));
       cmMemset((U8 *) &realmCfg->realmInfo[0].aqAppHostMap[0].
       hostId[0], 0, sizeof(LaqStr));
 #ifdef CLIENT
       #ifdef AQACC_CMD_LINE
       realmCfg->realmInfo[0].aqAppHostMap[0].hostId[0].length=
       peerId.hostLen;
       aqStrcpy((S8 *)realmCfg->realmInfo[0].aqAppHostMap[0].
       hostId[0].buff, (S8*)peerId.hostName);
       #else
       realmCfg->realmInfo[0].aqAppHostMap[0].hostId[0].length= 13;
       aqStrcpy((S8 *)realmCfg->realmInfo[0].aqAppHostMap[0].
       hostId[0].buff, "server.example.com");
       #endif
#else
       #ifdef AQACC_CMD_LINE
       realmCfg->realmInfo[0].aqAppHostMap[0].hostId[0].length=
       peerId.hostLen;
       aqStrcpy((S8 *)realmCfg->realmInfo[0].aqAppHostMap[0].
       hostId[0].buff,(S8*)peerId.hostName);
       #else
       realmCfg->realmInfo[0].aqAppHostMap[0].hostId[0].length= 16;
       aqStrcpy((S8 *)realmCfg->realmInfo[0].aqAppHostMap[0].
       hostId[0].buff, "client.example.com");
       #endif
#endif
```

A.6 Peer Configuration

```
/*****************************************************************
                        Variable Declaration

*****************************************************************/
   AqMngmt aqMngmt;
   Pst   pst;
   U16   dstProcId;
   U16   srcProcId;

   AqPeerCfg    *peerCfg;
   S16           ret;

/*****************************************************************
                      Variable Initialization

*****************************************************************/

   ret = ROK;
   cmMemset((U8*)&aqMngmt,0, sizeof(AqMngmt));
   cmMemset((U8*)&pst,0, sizeof(Pst));

   srcProcId = SFndProcId();
   dstProcId = SFndProcId();
   peerCfg   = &(aqMngmt.u.cfg.u.peer);

/*****************************************************************
                  Sample configuration for Peer 1

*****************************************************************/

    peerCfg->nmbPeers   =  1;
    /* Allocate memory for PeerInfo */
    SGetSBuf(TSTREG, TSTPOOL, (Data**)&peerCfg->peerInfo,
             sizeof(AqPeerInfo));
    cmMemset((U8 *) &peerCfg->peerInfo[0].hostId, 0,
             sizeof(LaqStr));
#ifdef CLIENT
     #ifdef AQACC_CMD_LINE
      peerCfg->peerInfo[0].hostId.length = peerId.hostLen;
      aqStrcpy((S8 *)peerCfg->peerInfo[0].hostId.buff,
      (S8*)peerId.hostName);
    #else
      peerCfg->peerInfo[0].hostId.length = 13;
      aqStrcpy((S8 *)peerCfg->peerInfo[0].hostId.buff,
      "server.example.com");
    #endif
```

```
#else
    #ifdef AQACC_CMD_LINE
        peerCfg->peerInfo[0].hostId.length = peerId.hostLen;
        aqStrcpy((S8 *)peerCfg->peerInfo[0].hostId.buff,
        (S8*)peerId.hostName);
    #else
        peerCfg->peerInfo[0].hostId.length = 16;
        aqStrcpy((S8 *)peerCfg->peerInfo[0].hostId.buff,
                   "client.example.com");
    #endif
#endif
#ifndef CM_TLS
    peerCfg->peerInfo[0].security = LAQ_NO_INBAND_SEC;
#else
    peerCfg->peerInfo[0].security = LAQ_SEC_TLS;
#endif
    peerCfg->peerInfo[0].protocol = LAQ_PROT_TCP;
    peerCfg->peerInfo[0].realmName.length=11;
    aqStrcpy((S8*)peerCfg->peerInfo[0].realmName, "example.com");

/****************************************************************
                        Configuring Peer Address

****************************************************************/
    if(peerCfg->peerInfo[0].protocol == LAQ_PROT_TCP)
    {
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.IpAddr.type =
                                            CM_NETADDR_IPV4;
        #ifdef CLIENT
        #ifdef AQACC_CMD_LINE
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.
        IpAddr.u.ipv4NetAddr = peerId.ipAddr;
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.port = SERVER_PORT;
        #else
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.IpAddr.
                  u.ipv4NetAddr = SERVER_IP_ADDR;
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.port = SERVER_PORT;
        #endif
        #else
        #ifdef AQACC_CMD_LINE
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.
        IpAddr.u.ipv4NetAddr = peerId.ipAddr;
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.port = CLIENT_PORT;
        #else
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.
                  IpAddr.u.ipv4NetAddr = CLIENT_IP_ADDR;
        peerCfg->peerInfo[0].aqPeerAddr.tuclAddr.port = CLIENT_PORT;
        #endif
        #endif
```

```
/******************************************************************
                      Configuring Self Address

******************************************************************/
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.type =
                                                 CM_NETADDR_IPV4;
   #ifdef CLIENT
   #ifdef AQACC_CMD_LINE
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.u.
                                     ipv4NetAddr = hostId.ipAddr;
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.port = 5678;
   #else
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.u.
                                    ipv4NetAddr = CLIENT_IP_ADDR;
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.port = 5678;
   #endif
   #else
   #ifdef AQACC_CMD_LINE
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.
                                  u.ipv4NetAddr = hostId.ipAddr;
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.port = 4567;
   #else
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.IpAddr.
                                  u.ipv4NetAddr = SERVER_IP_ADDR;
        peerCfg->peerInfo[0].aqSelfAddr.tuclAddr.port = 4567;
   #endif
   #endif
```

A.7 DM Configuration

```
/******************************************************************
                        Variable Declaration

******************************************************************/
     AqMngmt aqMngmt;
     Pst pst;
     U16  dstProcId;
     U16  srcProcId;
     AqDmMsgCfg    *dmMsgCfg;
     S16            ret;

/******************************************************************
                      Variable Initialization

******************************************************************/
     ret = ROK;
     cmMemset((U8*)&aqMngmt,0, sizeof(AqMngmt));
     cmMemset((U8*)&pst,0, sizeof(Pst));

     srcProcId = SFndProcId();
     dstProcId = SFndProcId();
     dmMsgCfg   = &(aqMngmt.u.cfg.u.appDmMsg);

     dmMsgCfg->appId     =  DFLT_ACCT_APPID;
     dmMsgCfg->nmbCmd     = 1;
     /* Allocate the memory for DmEntry */
     SGetSBuf(TSTREG, TSTPOOL, (Data**)&dmMsgCfg->dmEntry,
sizeof(AqDmEntry));

/******************************************************************
                   Populating DM Message header

******************************************************************/
     dmMsgCfg->dmEntry[0].dmCode = AQ_ACR_CMD_CODE;
     dmMsgCfg->dmEntry[0].cmdFlags = AQ_CMD_FLAG_R;
     dmMsgCfg->dmEntry[0].nmbAvpProp = 18;

/******************************************************************
            Populating Session ID Avp within ACR message

*******************************************************************/
    dmMsgCfg->dmEntry[0].avpProperties[0].avpCode =
AQ_SESSION_ID_AVPCODE;
    dmMsgCfg->dmEntry[0].avpProperties[0].properties = AQ_AVP_FIXED;
    dmMsgCfg->dmEntry[0].avpProperties[0]. posFlag = TRUE;
    dmMsgCfg->dmEntry[0].avpProperties[0]. position = 1;
    dmMsgCfg->dmEntry[0].avpProperties[0]. minOcc = 1;
    dmMsgCfg->dmEntry[0].avpProperties[0]. maxOcc = 1;
```

```
/*****************************************************************
           Populating Origin Host Avp within ACR message

******************************************************************/
    dmMsgCfg->dmEntry[0].avpProperties[1].avpCode =
AQ_ORIGIN_HOST_AVPCODE;
    dmMsgCfg->dmEntry[0].avpProperties[1].properties =
AQ_AVP_MANDATORY;
    dmMsgCfg->dmEntry[0].avpProperties[1]. posFlag = FALSE;
    dmMsgCfg->dmEntry[0].avpProperties[1]. position = 2;
    dmMsgCfg->dmEntry[0].avpProperties[1]. minOcc = 1;
    dmMsgCfg->dmEntry[0].avpProperties[1]. maxOcc = 1;

/*****************************************************************
          Populating Origin Realm Avp within ACR message

******************************************************************/
    dmMsgCfg->dmEntry[0].avpProperties[2].avpCode =
AQ_ORIGIN_REALM_AVPCODE;
    dmMsgCfg->dmEntry[0].avpProperties[2].properties =
AQ_AVP_MANDATORY;
    dmMsgCfg->dmEntry[0].avpProperties[2]. posFlag = FALSE;
    dmMsgCfg->dmEntry[0].avpProperties[2]. position = 3;
    dmMsgCfg->dmEntry[0].avpProperties[2]. minOcc = 1;
    dmMsgCfg->dmEntry[0].avpProperties[2]. maxOcc = 1;

/*****************************************************************
        Populating Destination Realm Avp within ACR message

******************************************************************/
    dmMsgCfg->dmEntry[0].avpProperties[3].avpCode =
AQ_DEST_REALM_AVPCODE;
    dmMsgCfg->dmEntry[0].avpProperties[3].properties =
AQ_AVP_MANDATORY;
    dmMsgCfg->dmEntry[0].avpProperties[3]. posFlag = FALSE;
    dmMsgCfg->dmEntry[0].avpProperties[3]. position = 4;
    dmMsgCfg->dmEntry[0].avpProperties[3]. minOcc = 1;
    dmMsgCfg->dmEntry[0].avpProperties[3]. maxOcc = 1;

/*****************************************************************
        Populating Accnt Record Type Avp within ACR message

******************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[4].avpCode =
AQ_ACCT_RECORD_TYPE_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[4].properties =
AQ_AVP_MANDATORY;
   dmMsgCfg->dmEntry[0].avpProperties[4]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[4]. position = 5;
   dmMsgCfg->dmEntry[0].avpProperties[4]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[4]. maxOcc = 1;
```

```
/*****************************************************************
         Populating Accnt Record Num Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[5].avpCode =
AQ_ACCT_RECORD_NMB_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[5].properties =
AQ_AVP_MANDATORY;
   dmMsgCfg->dmEntry[0].avpProperties[5]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[5]. position = 6;
   dmMsgCfg->dmEntry[0].avpProperties[5]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[5]. maxOcc = 1;

/*****************************************************************
           Populating Accnt App ID Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[6].avpCode =
AQ_ACCT_APPID_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[6].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[6]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[6]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[6]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[6]. maxOcc = 1;

/*****************************************************************
        Populating Vendor Spec App ID Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[7].avpCode =
AQ_VEN_SPEC_APPID_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[7].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[7]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[7]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[7]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[7]. maxOcc = 1;

/*****************************************************************
             Populating User Name Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[8].avpCode =
AQ_USER_NAME_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[8].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[8]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[8]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[8]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[8]. maxOcc = 1;
```

```
/*****************************************************************
       Populating Accnt Subsession ID Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[9].avpCode =
AQ_ACCT_SUBSESSION_ID_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[9].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[9]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[9]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[9]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[9]. maxOcc = 1; /

/*****************************************************************
         Populating Accnt Session ID Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[10].avpCode =
AQ_ACCT_SESSION_ID_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[10].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[10]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[10]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[10]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[10]. maxOcc = 1;

/*****************************************************************
      Populating Accnt Multi Session ID Avp within ACR message

*****************************************************************/

   dmMsgCfg->dmEntry[0].avpProperties[11].avpCode =
AQ_ACCT_MULTI_SESSIONID_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[11].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[11]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[11]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[11]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[11]. maxOcc = 1;

   dmMsgCfg->dmEntry[0].avpProperties[12].avpCode =
AQ_ACCT_INTERIM_INTERVAL_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[12].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[12]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[12]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[12]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[12]. maxOcc = 1;
```

```
/******************************************************************
       Populating Accnt Realmtime Req Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[13].avpCode =
AQ_ACCT_REALTIME_REQ_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[13].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[13]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[13]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[13]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[13]. maxOcc = 1;

/******************************************************************
         Populating Origin StateId Avp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[14].avpCode =
AQ_ORIGIN_STATEID_AVPCODE;
    dmMsgCfg->dmEntry[0].avpProperties[14].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[14]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[14]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[14]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[14]. maxOcc = 1;
/******************************************************************
           Populating Event TimeStamp within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[15].avpCode =
AQ_EVENT_TIMESTAMP_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[15].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[15]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[15]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[15]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[15]. maxOcc = 1;
/******************************************************************
              Populating ProxyInfo within ACR message

*****************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[16].avpCode =
AQ_PROXY_INFO_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[16].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[16]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[16]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[16]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[16]. maxOcc = 1;
```

```
/******************************************************************
            Populating Route Record within ACR message

******************************************************************/
   dmMsgCfg->dmEntry[0].avpProperties[17].avpCode =
AQ_ROUTE_RECORD_AVPCODE;
   dmMsgCfg->dmEntry[0].avpProperties[17].properties =
AQ_AVP_OPTIONAL;
   dmMsgCfg->dmEntry[0].avpProperties[17]. posFlag = FALSE;
   dmMsgCfg->dmEntry[0].avpProperties[17]. position = 0;
   dmMsgCfg->dmEntry[0].avpProperties[17]. minOcc = 1;
   dmMsgCfg->dmEntry[0].avpProperties[17]. maxOcc = 1;
```

A.8 AVP Configuration

```
/*****************************************************************
                        Variable Declaration

*****************************************************************/
   AqMngmt aqMngmt;
   Pst pst;
   U16  dstProcId;
   U16  srcProcId;
   AqAvpCfg    *avpCfg;
   LaqAvpEntry* tempAqAvpEntry;
   LaqAvpEntry* tempAqAvpEntry1;
   S16   ret;
/*****************************************************************
                       Variable Initialization

*****************************************************************/
   ret = ROK;
   cmMemset((U8*)&aqMngmt, 0, sizeof(AqMngmt));
   cmMemset((U8*)&pst,0, sizeof(Pst));
   srcProcId = SFndProcId();
   dstProcId = SFndProcId();
   avpCfg   = &(aqMngmt.u.cfg.u.appAvp);

/*****************************************************************
                     Sample configuration of  AVP

*****************************************************************/
     avpCfg->appId          =  DFLT_ACCT_APPID;
    avpCfg->nmbAvps         =  18;

     /* Allocate memory for AVPEntry */
     SGetSBuf(TSTREG, TSTPOOL,(Data **)&(avpCfg->avpEntry)
,(sizeof(LaqAvpEntry) *(18)));



/*****************************************************************
                      Populating Session ID AVP

*****************************************************************/
     avpCfg->avpEntry[0].avpCode  = AQ_SESSION_ID_AVPCODE;
     avpCfg->avpEntry[0].dataType = AQ_UTF8STRING;
     avpCfg->avpEntry[0].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;
```

```
/*****************************************************************
                    Populating Origin Host AVP

*****************************************************************/
     avpCfg->avpEntry[1].avpCode  = AQ_ORIGIN_HOST_AVPCODE;
     avpCfg->avpEntry[1].dataType = AQ_DIAMETER_IDENTITY;
     avpCfg->avpEntry[1].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/*****************************************************************
                    Populating Origin Realm AVP

*****************************************************************/
   avpCfg->avpEntry[2].avpCode    = AQ_ORIGIN_REALM_AVPCODE;
   avpCfg->avpEntry[2].dataType   = AQ_DIAMETER_IDENTITY;
   avpCfg->avpEntry[2].flags      = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/*****************************************************************
                    Populating Dest Realm AVP

*****************************************************************/
     avpCfg->avpEntry[3].avpCode  = AQ_DEST_REALM_AVPCODE;
     avpCfg->avpEntry[3].dataType = AQ_DIAMETER_IDENTITY;
     avpCfg->avpEntry[3].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/*****************************************************************
                  Populating Acct Record Type AVP

*****************************************************************/
    avpCfg->avpEntry[4].avpCode  = AQ_ACCT_RECORD_TYPE_AVPCODE;
    avpCfg->avpEntry[4].dataType = AQ_ENUMERATED;
    avpCfg->avpEntry[4].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/*****************************************************************
                  Populating Acct Record Nmb AVP

*****************************************************************/
    avpCfg->avpEntry[5].avpCode  = AQ_ACCT_RECORD_NMB_AVPCODE;
    avpCfg->avpEntry[5].dataType = AQ_UNSIGNED_32;
    avpCfg->avpEntry[5].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/*****************************************************************
                 Populating Vendor Spec App ID AVP

*****************************************************************/
    avpCfg->avpEntry[6].avpCode  = AQ_VEN_SPEC_APPID_AVPCODE;
    avpCfg->avpEntry[6].dataType = AQ_GROUPED;
    avpCfg->avpEntry[6].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

    /* Allocating memory for AVPs within Grouped AVP */
     SGetSBuf(TSTREG, TSTPOOL, (Data **)&(tempAqAvpEntry),
     (sizeof(LaqAvpEntry) *(3)));
```

```
/****************************************************************
            Populating Vendor ID AVP within Grouped AVP

****************************************************************/
     avpCfg->avpEntry[6].groupedAvp.avpEntry =
(LaqAvpEntry*)tempAqAvpEntry;
     avpCfg->avpEntry[6].groupedAvp.nmbGrpAvp= 3;
     avpCfg->avpEntry[6].groupedAvp.avpEntry[0].avpCode =
AQ_VENDOR_ID_AVPCODE;
     avpCfg->avpEntry[6].groupedAvp.avpEntry[0].dataType =
AQ_UNSIGNED_32;
     avpCfg->avpEntry[6].groupedAvp.avpEntry[0].flags =
AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/****************************************************************
           Populating Auth App ID AVP within Grouped AVP

****************************************************************/
    avpCfg->avpEntry[6].groupedAvp.avpEntry[1].avpCode =
AQ_AUTH_APPID_AVPCODE;
    avpCfg->avpEntry[6].groupedAvp.avpEntry[1].dataType =
AQ_UNSIGNED_32;
   avpCfg->avpEntry[6].groupedAvp.avpEntry[1].flags = AQ_AVP_FLAG_M
| AQ_AVP_FLAG_P;

/****************************************************************
           Populating Acct App ID AVP within Grouped AVP

****************************************************************/
   avpCfg->avpEntry[6].groupedAvp.avpEntry[2].avpCode =
AQ_ACCT_APPID_AVPCODE;
   avpCfg->avpEntry[6].groupedAvp.avpEntry[2].dataType =
AQ_UNSIGNED_32;
   avpCfg->avpEntry[6].groupedAvp.avpEntry[2].flags = AQ_AVP_FLAG_M
| AQ_AVP_FLAG_P;

/****************************************************************
                      Populating User Name AVP

****************************************************************/
    avpCfg->avpEntry[7].avpCode  = AQ_USER_NAME_AVPCODE;
    avpCfg->avpEntry[7].dataType = AQ_UTF8STRING;
    avpCfg->avpEntry[7].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;

/****************************************************************
                 Populating Acct Subsession ID AVP

****************************************************************/
   avpCfg->avpEntry[8].avpCode  = AQ_ACCT_SUBSESSION_ID_AVPCODE;
   avpCfg->avpEntry[8].dataType = AQ_UNSIGNED_64;
   avpCfg->avpEntry[8].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;
```

```
/*****************************************************************
                   Populating Acct Session ID AVP

*****************************************************************/
   avpCfg->avpEntry[9].avpCode  = AQ_ACCT_SESSION_ID_AVPCODE;
   avpCfg->avpEntry[9].dataType = AQ_OCTET_STRING;
   avpCfg->avpEntry[9].flags    = AQ_AVP_FLAG_M | AQ_AVP_FLAG_P;
```

Appendix B

Diameter Performance

This section describes the performance analysis of Continuous Computing's Diameter Base Protocol software. The performance analysis has been done for specific hardware and software environments as described in this document. The performance depends on many factors, such as the system architecture, layer coupling, operating system, hardware organization, and compiler options. Also, the figures indicated correspond to performance of stand alone Diameter software with sample application running over it.

The performance of Continuous Computing's Diameter stack was measured in the hardware and software environments as described in Table B-1 and Table B-2. Therefore, the performance may vary from the performance described in this document depending on the environment.

B.1 Hardware Information

Table B-1 describes the hardware set up used for the performance analysis.

Table B-1: Hardware configuration for Diameter

Type	Description
Operating System	Red Hat Enterprise Linux
CPU	Single core CPU/32 bit, 2.8 GHz, 240 MB Memory, 1 MB Cache
Processor	Pentium 4

B.2 Software

Table B-2 describes the software parameters used for performance analysis.

Table B-2: Software configuration for Diameter

Type	Description
Operating System	Linux, Kernel version=2.4.21
Compiler	GNU C Compiler gcc version 3.2.3
Compiler option	-O3 -ansi -Wall -Wno-comment -pipe -Wstrict-prototypes -Wshadow -Wcast-qual -Wmissing-prototypes -pedantic -Wimplicit -Wunused
Protocol Layer	Diameter 1.2

B.3 Performance Results

B.3.1 Test Setup

Figure B-1 shows the test set up for measuring the performance.

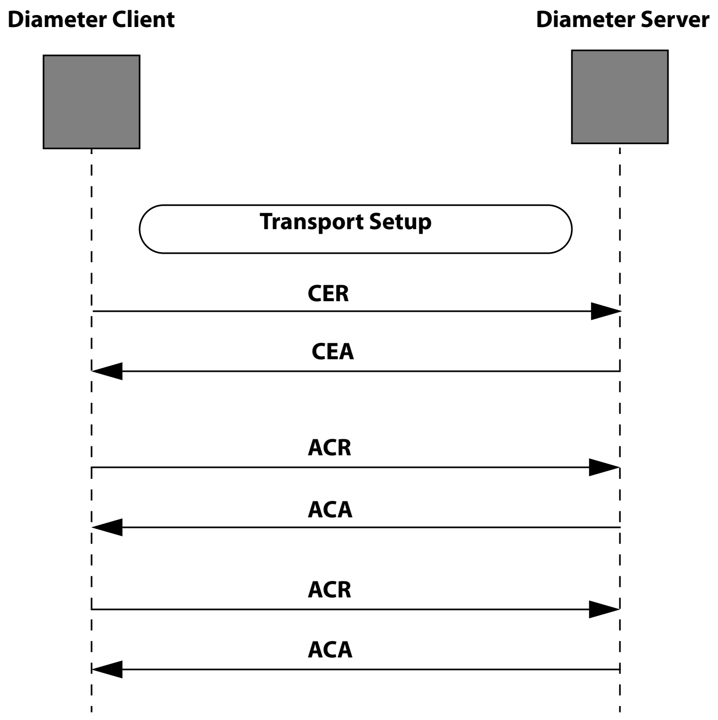


Figure B-1: Test setup for measuring performance figure

B.3.2 Test Procedure

The test procedure is as follows:

- Continuous Computing's Diameter stack is configured as Server with a thin sample application running over it.
- Another Diameter stack is configured as Client with a thin sample application running over it.
- Connection is established between Diameter Server and Client over TUCL.
- Once the CER and CEA are exchanged, ACR (Accounting Request) messages are sent from the Diameter Client.
- Diameter Server in-turn processes the ACR and send the ACA (Accounting Answer) in response.
- Around 4 million ACRs were sent and 4 million ACAs were received in response to the ACRs. The performance is calculated based on the time taken to process 8 million messages in tightly coupled mode. The performance figures are shown in Table B-3:

Table B-3: Diameter stack performance figures

Sl. No.	Messages per second (Sent + Received)	CPU Utilization	Average Message Length (bytes)	Comments
1.	13840	45%	270	With Diameter message and AVP validation against DM and AVP dictionaries.
2.	15384	45%	270	With Diameter message validation against DM dictionary, but without AVP validation against AVP dictionary.

Appendix C

Addendum

This section is an addendum to version 1.22a of the Diameter Base Protocol Service Definition.

Note: *The material in this addendum is integrated into the body of the document and noted below.*

Table C-1: List of Changes for this Addendum

Description of Change	Section Reference
Added new section for COTS configuration.	Section 8, "COTS Configuration".

References

Refer to these documents for more information:

1. RFC 3588 - Diameter Base Protocol.
2. Diameter Product Requirement Specification, Continuous Computing Corporation (p/n 1132349).
3. AQU Interface Service Definition, Continuous Computing Corporation (p/n 1100073).
4. SCT Interface Service Definition, Continuous Computing Corporation (p/n 1100036).
5. HIT Interface Service Definition, Continuous Computing Corporation (p/n 1100031).
6. System Service Interface Service Definition, Continuous Computing Corporation (p/n 1111001).

Index