The Salutation Consortium™

Salutation Architecture Specification (Part-1)

Version 2.1

TBD, 1999

Limitation of Liability

Even though the members of the Salutation Consortium have reviewed this Specification, the Consortium shall not make any warranty or representation, neither express nor implied, with respect to this Specification, its quality or accuracy and it specifically disclaims the warranties of merchantability and fitness for a particular purpose.

No representation of third party rights

The Salutation Consortium makes no representation or warranty whatsoever with regard to the Consortium member or third party ownership, licensing or infringement/non-infringement of intellectual property rights. Each user of this Specification, whether or not a Consortium member, should seek the independent advice of legal counsel with regard to any possible violation of third party rights.

Trademarks

All product names are trademarks of the respective product owners and/or companies.

Preface

Part 1 of the Salutation Architecture Specification, this document, defines the general framework of the architecture and the details of the Salutation Manager Protocol.

Part 2 defines the Salutation Personality Protocol and Attributes of Functional Units.

Part 3 defines the criteria of the Conformance to the Salutation Architecture Specification.

Revision

Version 2.0 (January 31, 1996)

Public release of the final version 2.0 specification part 1.

Version 2.0a (December 02, 1996)

Added the sixth scheme to the SLM-ID generation method. The new scheme uses IPX address.

Version 2.0b (October 15, 1997)

Refined the explanation of the Architecture overview and Salutation Manager Protocol and API.

Version 2.0c (June 01, 1999)

No change and no release

Version 2.1 (TBD, 1998)

Added the directory-based service discovery by utilizing Service Location Protocol (SLP) Version 2.

Table of Contents

1.Introduction

1.1.Architecture Objectives

The **Salutation Architecture** was created to solve the problems of ***service discovery and utilization*** among a broad set of appliances and equipment and in an environment of widespread connectivity and mobility.

The architecture provides a standard method for applications, services and devices to describe and to advertise their capabilities to other applications, services and devices and to find out their capabilities. The architecture also enables applications, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilize their capabilities.

Given the diverse nature of target appliances and equipment in an environment of widespread connectivity, the architecture is processor, operating system, and communication protocol independent, and allows for scalable implementations, even in very low-price devices.

1.2.References

This section identifies other documents on which this document relies.

ISO 8824	Information processing systems - Open systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)
ISO 8825	Information processing systems - Open systems Interconnection - Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)
RFC-1014	Sun Microsystems, “XDR: External Data Representation Standard”, June 1987
RFC-1057	Sun Microsystems, “RPC: Remote Procedure Call Protocol Specification Version 2”, June 1988
RFC-1700	Reynolds & Postel, “Assigned Numbers”, October 1994
IrLAP	Infrared Data Association, “Serial Infrared Link Access Protocol (IrLAP) Version 1.0”, June 23, 1994
IrLMP	Infrared Data Association, “Link Management Protocol (IrLMP) Version 1.0”, August 11, 1994
TinyTP	Infrared Data Association, “TinyTP : A Flow-Control Mechanism for use with IrLMP, Version 1.0”, October 25, 1995
RFC-2608	E. Guttman, C. Perkins, J Vaizades, and M. Day, "Service Location Protocol,

	Version 2", June 1999
RFC-2609	E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes", June 1999
RFC-2254	T. Howes, "The String Representation of LDAP Search Filters", December 1997
RFC-2251	M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", December 1997

1.3.Definition of Terms

This section defines basic terms used in this architecture document.

All Call Functional Unit

All Call Functional Unit Description Record is a reserved field used to specify a capability search requesting information about all Functional Units registered in a specific Salutation Manager.

Equipment

Equipment is a physical device or information appliance, such as, but not limited to, a printer, facsimile machine (FAX), telephone, Private Branch Exchange (PBX), Personal Digital Assistant (PDA), or computer (of any size).

Functional Unit

A Functional Unit is a logical subdivision of Equipment that performs one meaningful task such as “Print”, “FAX” or “OCR”.

Networked Entity

Networked Entity is a device, application, service or Functional Unit that has access to or may be accessed from other applications, services or devices. A Networked Entity may exist in the same Equipment or on different Equipment connected by a network.

Salutation Application

Salutation Application is an application program running in the Equipment that provides Service(s) to and/or uses the Service(s) of other Equipment under the protocol defined by the Salutation Architecture.

Salutation Architecture

The subject of this specification, the Salutation Architecture is a framework created to solve the problems of service discovery and utilization among a broad set of equipment.

Salutation Client/Service Application

A Salutation Client Application uses Services of other Equipment under the protocol defined by the Salutation Architecture.

A Salutation Service Application provides Services to other Equipment under the protocol defined by the Salutation Architecture. Service is a generic term describing a function or resource provided by one piece of Equipment to another. Some examples of Services are:

- Printing a document
- Sending a document by FAX
- Making address book data (name, address, telephone number, etc.) accessible (to look up, add, update, delete, etc.) from another piece of Equipment (or application)
- Sending a voice message
- Providing an image to coded character transformed via an optical character recognition (OCR) process
- Converting Digital Video Disk (DVD) encoding to full motion analog video signals.

One Salutation Application may function as a Service sometimes, as a Client at other times, or as a Service and a Client at the same time. Collectively, this is called a Salutation Client/Service Application. A Salutation Client/Service Application may be referred simply as Client or Service in this document.

Salutation Client/Service Equipment

Salutation Client Equipment is Equipment that has one or more running Salutation Client Applications.

Salutation Service Equipment is Equipment that has one or more running Salutation Service Applications.

The same Equipment can function as a Service sometimes, as a Client at other times, or as a Service and a Client at the same time. Collectively, this is called Salutation Client/Service Equipment. Salutation Client/Service Equipment may be referred simply as Salutation Equipment in this document.

2.Architecture Overview

2.1.Salutation Manager

A high level overview of the Salutation Manager is included in this section along with a description of service broker tasks.

2.1.1.General

As shown in Figure 2-1, the Salutation Architecture defines an entity called the **Salutation Manager (SLM)** that functions as a service broker for applications, services and devices called a Networked Entity. The Salutation Manager allows Networked Entities to discover and utilize the capabilities of other Networked Entities.

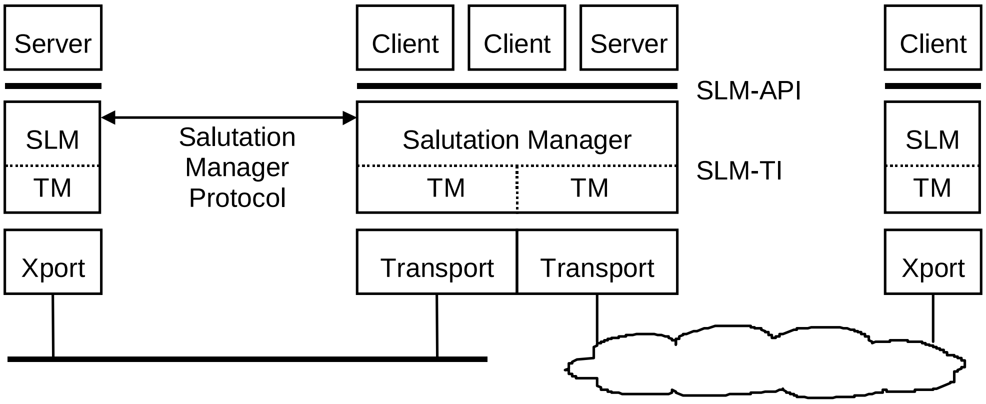

Figure 2-1: Model of the Salutation Manager

A Networked Entity may be a service provider, called a **Service**. The Service registers its capability with a Salutation Manager. A Networked Entity may be a service user, called a **Client**. The Client discovers Services and requests to use them through a Salutation Manager. A Networked Entity may serve as either a Client or a Service, or both.

The Salutation Manager provides a transport-independent interface, called the **Salutation Manager Application Program Interface (SLM-API)**, to Services and Clients. The architecture defines an abstract procedural SLM-API.

The Salutation Manager communicates with other Salutation Managers to perform its role as a service broker. The Salutation Manager-to-Salutation Manager communications protocol is defined by the Salutation Architecture and called the **Salutation Manager Protocol**.

The Salutation Manager Protocol uses Sun Microsystems' Open Networking Computing Remote Procedure Call version 2 protocol.

The Salutation Manager Protocol requires that the underlying transport protocol support multiple reliable bi-directional communications.

Each Salutation Manager has a universally unique identifier, **SLM-ID**. SLM-ID is a 16-octet-long opaque OCTET STRING used by Clients, Services, and Salutation Managers to uniquely identify a particular Salutation Manager in a transport independent way.

The Salutation Manager provides a transport-independent interface, called the **Salutation Manager Transport Interface (SLM-TI)**, to transport-dependent entities, called **Transport Managers**. The Transport Manager is introduced to make the Salutation Manager transport-independent. The Salutation Manager and Transport Manager(s) together perform the service broker role. In the current specification, there is no specific description about Transport Manager. In a future release, the part of SLM common and Transportation Manager will be defined separately and SLM-TI will be defined to connect both these parts.

A Salutation Manager works with one or more Transport Managers. Each Transport Manager discovers other remote Salutation Managers connected to the transport it supports. For each discovered remote Salutation Manager, the Transport Manager finds the SLM-ID of the remote Salutation Manager, registers the SLM-ID with the local Salutation Manager, and maintains the association between the transport address and the SLM-ID of the remote Salutation Manager.

The Transport Manager discovers other remote Salutation Managers in a number of ways such as the following:

- Transport Manager has a static table of the transport address of remote Salutation Managers. The format and structure of the table is outside the scope of this Architecture.

- Transport Manager broadcasts a query to find other remote Salutation Managers over the transport using the protocol defined by this Architecture. (This method is possible only if the transport supports a broadcast mechanism.)

- If there is a central server containing the directory of Salutation Equipment, Transport Manager may inquire the transport address of other Salutation Managers there. Though the protocol between the Transport Manager and such directory is outside this Architecture’s scope, how to utilize the Service Location Protocol (SLP) is covered in section 4.6.Guidelines to utilize Service Location Protocol Version 2 for Service Discovery.

- The Client specifies the type of transport and the transport address of a remote Salutation Manager through the SLM-API.

Each piece of Equipment has, at most, one Salutation Manager. When there is no local Salutation Manager, Clients/Services may use a remote Salutation Manager, a Salutation Manager in another piece of Equipment, through Remote Procedure Call as shown in Figure 2-2. It is up to the implementation of each Salutation Manager whether or not it provides the Remote Procedure Call interface to remote Clients.

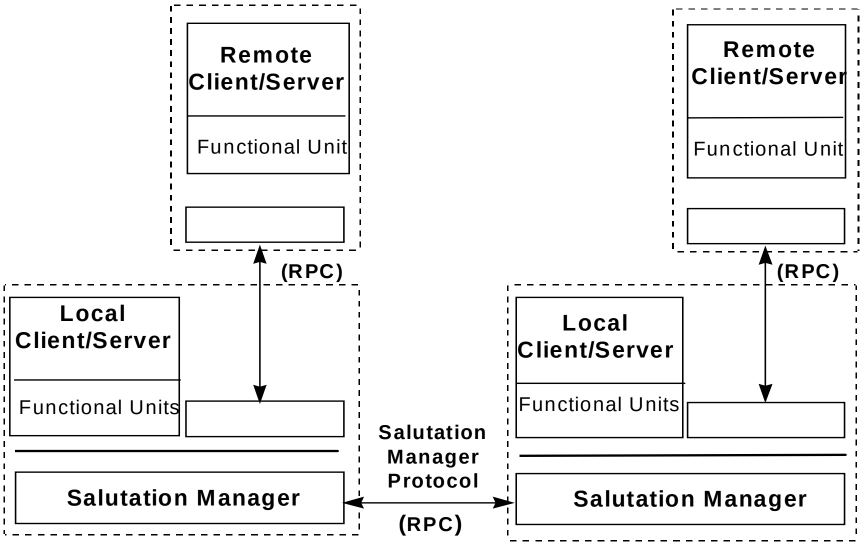

Figure 2.2: Use of Remote Procedure Call for Remote Clients/Services

2.1.2.Service Broker Tasks

To perform its function as service broker, the Salutation Manager provides four basic tasks:

- Service Registry
- Service Discovery
- Service Availability
- Service Session Management

2.1.2.1.Service Registry

The Salutation Manager contains a **Registry** to hold information about Services. The minimum requirement for the Registry is to store information about Services connected to the Salutation Manager. As depicted in Figure 2-2, these Services may reside in the local Salutation Equipment or may connect to the Salutation Manager via Remote Procedure Calls.

Optionally, the Salutation Manager Registry may store information about Services that are registered in other Salutation Managers. This expanded Registry function allows the local Salutation Manager to maintain a ‘directory’ of Salutation Equipment that is important to the local environment. For example, if the local Salutation Manager supports a print server, the Registry may include information about all Salutation print Services.

The Salutation Manager Registry may also serve a network directory function, providing information about all Salutation Equipment within the range of the local Salutation Manager, regardless of the Equipment type. In this case, one of the Salutation Managers in a network would be designated as the central directory. It would have the responsibility of finding and registering all Salutation Equipment. All requests by other equipment for Salutation resources would be directed toward this Salutation Manager which would respond accordingly.

The limit on Registry implementation is the size of the storage reserved for the Registry function.

2.1.2.2.Service Discovery

The Salutation Manager can discover other remote Salutation Managers and determine the Services registered there. **Service Discovery** is performed by comparing a required Services type(s), as specified by the local Salutation Manager, with the Service type(s) available on a remote Salutation Manager. Remote Procedure Calls are used to transmit the required Service type(s) from the local Salutation Manager to the remote Salutation Manager and to transmit the response from the remote Salutation Manager to the local Salutation Manager. Through manipulation of the specification of required Service type(s), the Salutation Manager can determine:

- The characteristics of **all the Services** registered at a remote Salutation Manager
- The characteristics of a **specific Service** registered at a remote Salutation Manager
- The presence of a Service on a remote Salutation Manager **matching a specific set of characteristics**.

2.1.2.3.Service Availability

The Salutation Manager can periodically check the availability of a Service. The local Salutation Manager requests the appropriate Salutation Manager to perform an **Availability Check**. The Availability Check is performed by exchanging Remote Procedure Call messages between the Salutation Managers. The period of the Availability Check is specifiable.

2.1.2.4.Service Session Management

When a Client wants to use a Service provided by Salutation Equipment, the Salutation Manager can establish a virtual data pipe between a Client and a Service. This is called a **Service Session**. Commands, responses and data are exchanged between Clients and Services on these data pipes in blocks called **Messages**. Messages have a defined format and are exchanged under a defined protocol. Such definitions of message exchange format and protocol are called **Personality Protocols**. The Salutation Manager may be instructed to operate in one of three distinct Personality Protocols while managing this data pipe as shown in Figure 2-3.

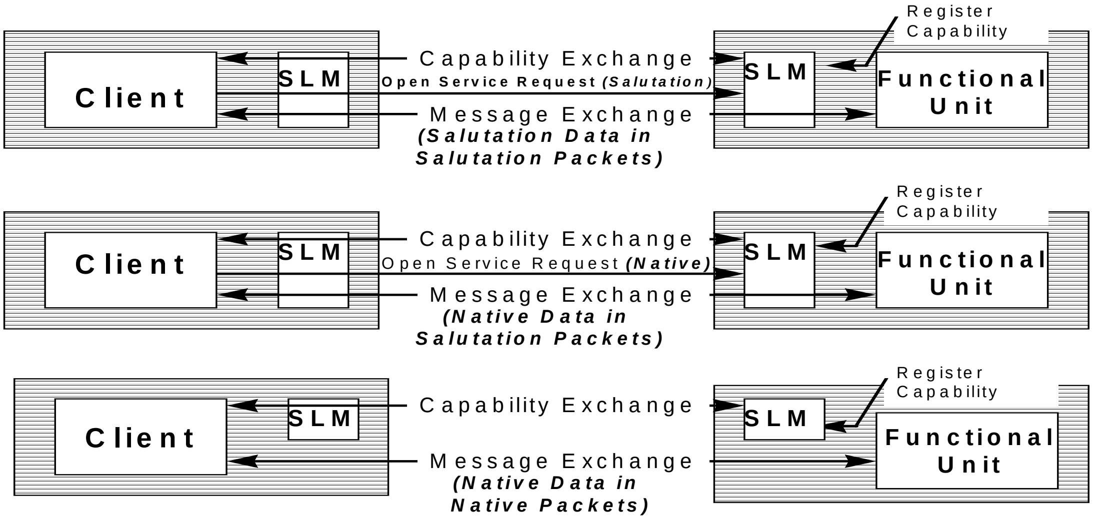

Figure 2-3: Personality Alternatives

- The Salutation Manager may set up the data pipe and then step into the background, allowing the Client and Service to manage the message stream and data formats. This is known as **Native Personality**. This personality is useful when Salutation Manager is used solely to discover the capabilities of other Network Entities, with the applications, services and devices managing the interactions between Clients and discovered Services.

  Messages are exchanged between Clients and Services directly, without the involvement of the Salutation Manager. Messages are NOT carried by the Salutation Manager Protocol under a Native Personality Protocol. Message exchange is native data in native packets.

  Note: Although the Salutation Manager Protocol is not used with the Native Personality Protocol, the Salutation Manager Protocol may be used by the Client to find or query the Service prior to requesting the Service. However, it is subject to the type of underlying transport and the native protocol as to whether the exchange of messages under a Native Personality Protocol and the use of Salutation Manager Protocol may actually coexist.

- The Salutation Manager may set up the data pipe and manage the message stream, while the data formats are selected and controlled by the Client and Service. This is known as **Emulated Personality**. This personality is useful when a common messaging protocol does not exist between a Client and a discovered Service.

  All Messages under an Emulated Personality Protocol are carried by the Salutation Manager Protocol. Message exchange is native data in Salutation packets.

  Under the Emulated Personality Protocol, Client Messages go through Salutation Managers, however the Salutation Manager never inspects the contents or semantics of Messages.

- The Salutation Manager may set up the data pipe, manage the message stream, and provide the data format definition for Client/Service interaction. This is known as **Salutation Personality**. This personality is the subject of the Salutation Architecture and provides a common messaging protocol and common data format between a Client and a discovered Service.

Under the Salutation Personality Protocol, message format and exchange protocol are defined by the Salutation Architecture. Some Salutation Personality Protocol commands, parameters, and protocols are common across Functional Unit types under a common framework. All Messages under the Salutation Personality Protocol are carried by the Salutation Manager Protocol. Message exchange is Salutation data in Salutation packets.

Under the Salutation Personality Protocol, Client Messages go through Salutation Managers, however the Salutation Manager never inspects the contents or semantics of Messages.

2.2.Architected Functional Unit

Network Entities may be subdivided by meaningful functionality called Functional Units. In principle, the architecture defines one meaningful function (from the Client's viewpoint) as one Functional Unit. For example, a function that rasterizes document data to image data may be defined as [Rasterize] Functional Unit, while a function that prints document data is defined as [Print] Functional Unit which includes both the rasterizing function and the image printing function.

A Functional Unit may be an entire Client or a part of a Client. Additionally, a Functional Unit may be an entire Service or a part of a Service. For example, the [Print] Functional Unit may make up an entire Print Service or be grouped with a [Scan] Functional Unit and [Fax Data Send] Functional Unit to form a Fax Service. Commands, responses, and data may be exchanged between a Client and a Functional Unit, a Functional Unit and a Service, or a Functional Unit and another Functional Unit.

The architecture provides abstract definitions of Functional Units. Abstraction of Functional Units are used to define all Service functions, as well as certain Client functions. A set of Attributes is defined for each abstraction of a Functional Unit. Each Attribute describes an aspect of the capability of the Functional Unit. For example, the [Print] Functional Unit has attributes such as whether double-sided print or various paper sizes are supported, the number of sorters, and so on.

In this Architecture the names of Functional Units are customarily enclosed in square brackets [ ]. Some examples of Functional Units are:

[Print]

[Fax Data]

[Address Book]

The Architecture defines a syntax and semantics for describing the attributes of each Functional Unit abstraction, called a Functional Unit Description Record, discussed in section 2.3.

2.3.Service Records

The following types of service records are defined:

- Service Description Record
- Functional Unit Description Record
- Attribute Record

The format of the Service Description Record, the Functional Unit Description Record and the Attribute Record are specified with Abstract Syntax Notation One as defined by ISO 8824, and are encoded with the Basic Encoding Rules for Abstract Syntax Notation One.

The indefinite form of the length octets should NOT be used when these records are encoded by the Basic Encoding Rules.

2.3.1.Service Description Record

The collection of Functional Units within a Network Entity define the Services available. The Architecture defines a syntax and semantics for describing the attributes of these Services called a **Service Description Record**. The Service Description Record is a set of Functional Unit Description Record abstract definitions.

In general, a **Service Description Record** consists of zero or more **Functional Unit Description Records** as illustrated in Figure 2-4.

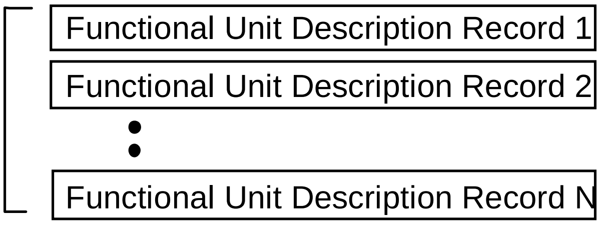

Figure 2-4: Service Description Record

There are three classes of a Service Description Record that are used in three contexts:

- **Registered** Service Description Record is a collection of all the Functional Unit Description Records registered and maintained on a given Salutation Manager. It is generated as a result of Functional Unit registration.
- **Requested** Service Description Record is a collection of Functional Unit Description Records representing Service requirements of a Client. It is generated as a result of a Service Discovery request. This record is Included in the input parameter of **slmSearchCapability(), slmSearchCapability2()** or **slmQueryCapability()**, or in the

Query Capability call message.

- **Reply** Service Description Record is a union between the local registered Service Description Record and requested Service Description Record. It is generated as a response to a Service Discovery request. The record is included in the output parameter of **slmQueryCapability()**, or in the *Query Capability* reply.

2.3.2.Functional Unit Description Record

A **Functional Unit Description Record** consists of a **Functional Unit ID** field and a **Functional Unit Handle** followed by zero or more **Attribute Records** as shown in Figure 2-5.

The Functional Unit ID field identifies the type of Functional Unit, for example, [Print], [FAX Data], etc. A special Functional Unit ID, called an **All Call Functional Unit ID**, is defined.

A unique Functional Unit Handle is assigned to each Functional Unit Description Record by the Salutation Manager. Therefore, a particular Functional Unit can be distinguished even if multiple Functional Units having the same Functional Unit ID are registered with the same Salutation Manager.

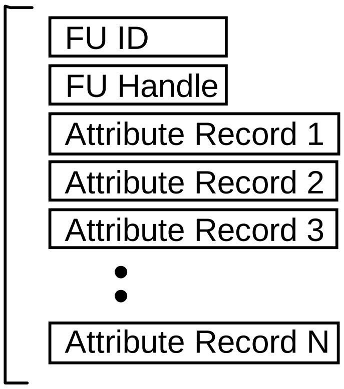

Figure 2-5: Functional Unit Description Record

All the Attribute Records defined for the type of Functional Unit must be present in a registered Functional Unit Description Record. The same attribute must not appear more than once in a Functional Unit Description Record .

A Functional Unit Description Record in a requested Service Description Record (called requested Functional Unit Description Record) may contain zero or more Attribute Records.

A Functional Unit Description Record in a reply Service Description Record (called reply Functional Unit Description Record) contains the union of all the Attribute Records in the requested Functional Unit Description Record and those in the registered Functional Unit Description Record.

2.3.3.Attribute Record

An **Attribute Record** consists of an **Attribute ID** field, identifying the type of the attribute, a **Compare Function ID/Compare Result** field specifying a function that compares the attribute, or indicates the result of the comparison and a **Value** field, containing the value of the attribute. The Attribute Record is depicted in Figure 2-6.

The **Compare/Function ID/Compare Result** field is used in two ways:

- Attribute Record in registered or requested Functional Unit Description Record - This field specifies the type of a function that compares the Attribute Value of the registered Functional Unit Description Record with that of the requested Functional Unit Description Record.

  The compare function specified in the registered Functional Unit Description Record is the default compare function. It is used unless another compare function is specified in the requested Functional Unit Description Record .

- Attribute Record in reply Functional Unit Description Record - This field indicates the result of the comparison.

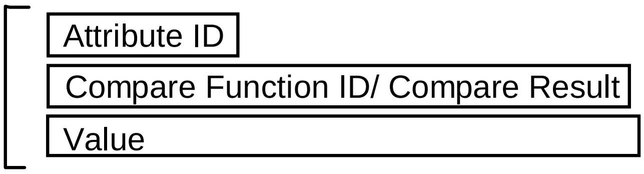

Figure 2-6: Attribute Record

2.4.Service Discovery

A Client or Functional Unit can locate a required Service by determining the attributes, or **capabilities,** of the Service's Functional Unit(s). The Service may reside in the same Equipment as the locating Client/Functional Unit, or it may be located in other Equipment. The process for determining the capabilities of a Service is called **Service Discovery** or **Capabilities Exchange**. Depending on whether or not the target Functional Unit is associated with the same Salutation Manager as the locating Client/Functional Unit, the Capabilities Exchange is either within the Salutation Manager or across two Salutation Managers as shown in Figure 2-7. Additionally, capability information may be provided to the locating Client/Functional Unit outside of the Salutation Architecture. For example, the capability information may be provided on a diskette or retrieved from a central directory.

2.5.Service Session

When a Client or Functional Unit wants to use a Service provided by a Functional Unit under a Salutation or an Emulated Personality Protocol, it requests a Salutation Manager to establish a **Service Session** between the Client/Functional Unit Client and the Functional Unit. The

Client/Functional Unit specifies a Personality Protocol to be used in the Service Session in the request. Depending whether or not the target Functional Unit is associated with the same Salutation Manager as the Client/Functional Unit, the Service Session is established either within the Salutation Manager or across two Salutation Managers as shown in Figure 2-7.

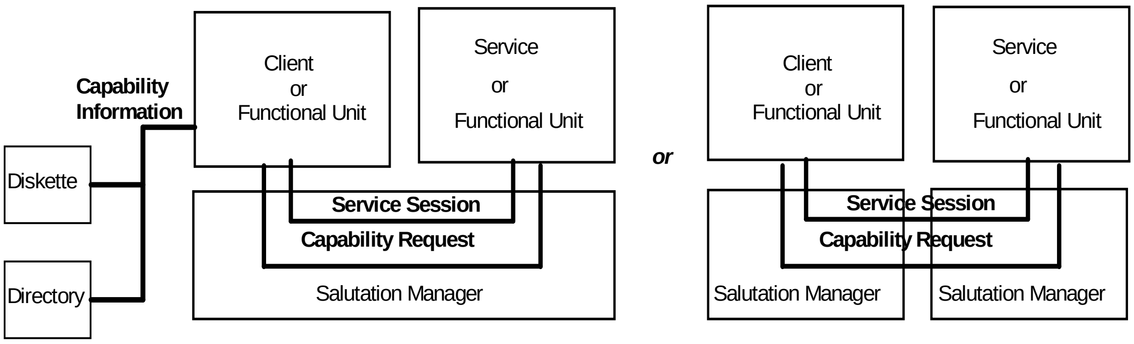

Figure 2-7: Establishing Service Sessions between Two Functional Units

As shown in Figure 2-8, there may be zero or more Clients and zero or more Functional Units associated with a Salutation Manager. Some of the Clients and Functional Units are in the same Equipment as the Salutation Manager, while others are not. Also, Clients or Functional Units may have more than one concurrent independent Service Session.

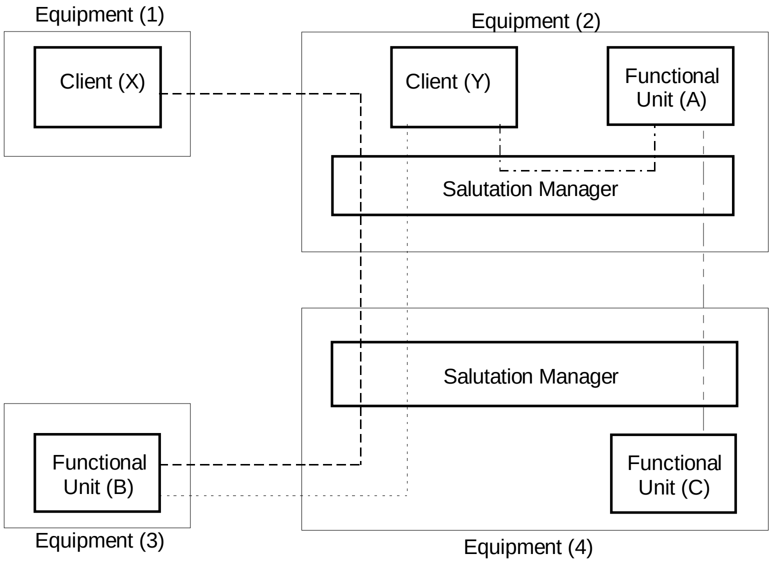

Figure 2-8: Use of the Salutation Manager in Multiple Equipment

2.6.Service Abstraction

The architecture defines a standard format, **Functional Unit Description Record**, to describe the capabilities of a Functional Unit instance.

The Service must register a Functional Unit Description Record to a Salutation Manager to make its capabilities available to other Services and Clients. The Salutation Manager assigns a unique value, **Functional Unit Handle**, to each registered Functional Unit instance.

The Functional Unit Description Record includes the type of Functional Unit, e.g. [Print], [FAX Data], etc. and a set of Attribute Records.

Each Attribute Record indicates a characteristic of the capability of the Service provided by the Functional Unit.

All the Attribute Records defined for the type of Functional Unit must be present in a registered Functional Unit Description Record. The architecture defines another standard format, the **Service Description Record**. This record enables the Salutation Manager to:

- maintain all the registered capabilities
- describe what Services are actually available in response to the Client's request

The record also enables a Client to:

- describe what kind of Services it is interested in when it requests a Salutation Manager
- find the detail capability of a Service
- search Services that have a specific capability

A Service Description Record contains a set of Functional Unit Description Records.

Figure 2-9 summarizes how the Service Description Record and Functional Unit Description Record are used.

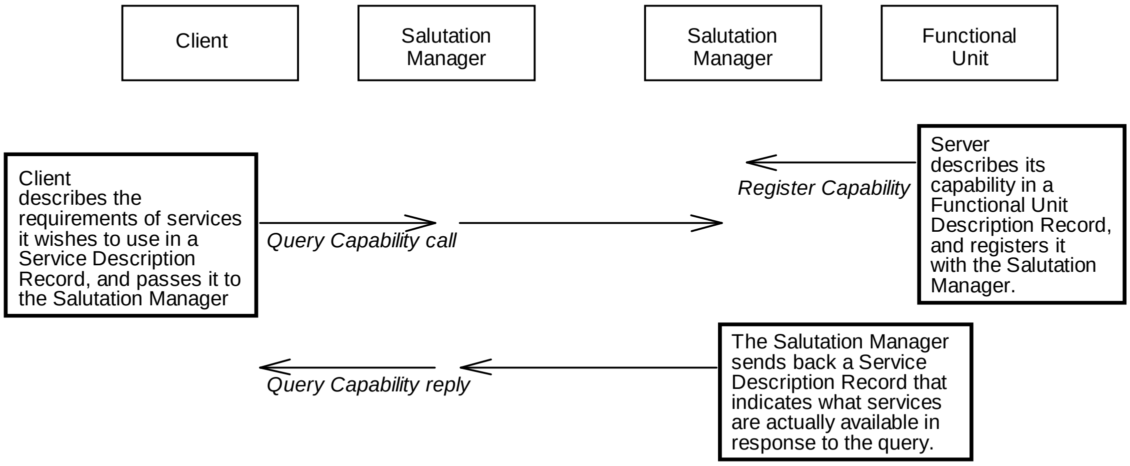

Figure 2-9: Using the Service and Description Records

2.7.Salutation Manager Protocol

The Salutation Manager communicates with other Salutation Managers using the Salutation Manager Protocol to perform its role as a Service broker. The Salutation Manager Protocol uses Remote Procedure Call, that is, a Salutation Manager makes a Remote Procedure Call to another Salutation Manager, which returns a Remote Procedure Call reply. The Remote Procedure Call/reply messages defined by the Salutation Manager Protocol are categorized into the following functional groups:

- Get Minor Version
- SLM-ID Exchange
- Capability Exchange
- Service Request
- Availability Check

2.7.1.2.7.1 Get Minor Version

The version number of the Salutation Manager Protocol specification under this release of the architecture is 2.1. The version number consists of a major version number, which is two (2), and a minor version number, which is one (1). The major version number is indicated by the Remote Procedure Call program version number. The minor version number of a remote Salutation Manager can be queried by sending the Get Minor Version Remote Procedure Call message.

The Salutation Manager sends the Get Minor Version call message to a remote Salutation Manager to query the minor version number the remote Salutation Manager supports.

2.7.2.SLM-ID Exchange

The Transport Manager sends the *Exchange SLM-ID* call message to a remote Salutation Manager to query the SLM-ID of the remote Salutation Manager. The called Salutation Manager returns its SLM-ID in the *Exchange SLM-ID* reply message.

This SLM-ID Exchange protocol is used by the Transport Manager to find the SLM-ID of other remote Salutation Managers. The calling Transport Manager specifies the SLM-ID of its own Salutation Manager in the message. By doing so, the called Salutation Manager does not have to issue another Exchange SLM-ID call message to discover the SLM-ID of the calling Salutation Manager.

2.7.3.Capability Exchange

The Capability Exchange protocol is used by Salutation Managers for Salutation Clients to discover the details of Services provided by Salutation Servers in other Equipment. The Salutation Client can get the following information on the Services through the Capability Exchange.

- The contents of Services (e.g., "print" Service with capabilities to process PostScript files, double-sided print, a sort of up to 10 copies, and A4/Letter/Legal/B5 paper sizes)
- Necessary information to use the Service, i.e. what Personality Protocols, data formats, etc. are supported

These Service details are described in a Service Description Record. Capability Exchange is not a prerequisite condition for a Client to use the Services of Servers. Even though the Service Description Record of Services, or equivalent information, is made known to a Client by means outside the Salutation Architecture scope (e.g., by manually transferring a file containing the Service Description Record by diskette), the Client can still use the Services through the Service Request protocol described in the next section.

The Capability Exchange is done by exchanging the *Query Capability* Remote Procedure Call/reply message that contains a Service Description Record between the Salutation Manager associated with a Client and the Salutation Manager associated with Functional Units.

The Salutation Manager in the Client Equipment sends a *Query Capability* call message with a Service Description Record from a Client to the Salutation Manager in the Server Equipment. The query describes the requirements of the Service(s) the Client wishes to utilize and requests the detail of the Service in the Server Equipment.

When the Salutation Manager in the Service Equipment receives a *Query Capability* call message, it compares the received Service Description Record against its Service Description Record consisting of registered Functional Unit Description Records. The Salutation Manager then returns a Service Description Record that contains all the matched Functional Unit Description Records, in the Remote Procedure Call reply message, to the Salutation Manager in the Client Equipment.

This approach provides three basic techniques for determining the capabilities of Service on other Salutation enabled Equipment. These approaches are to discover all Functional Units registered in the other Equipment, discover all Functional Units of a specific type registered in the other equipment, and discover if a Functional Unit consisting of a specific set of capabilities is registered in the other Equipment. The type of approach is dependent on the format of the Service Description

Record contained in the Query Capabilities call.

- ALL CALL: This technique provides a search mechanism for the discovery of all Functional Units registered in another Salutation Manager. This technique uses a reserved Functional Unit Description Record, named All Call, in the Service Description Record of the Query Capabilities call to specify that the Service Description Record in the Query Capabilities reply is to contain all Functional Unit Description Records registered in the other Equipment. If the All Call Functional Unit Description Record is used, it must be the only Functional Unit Description Record in the Service Description Record.

- TYPE CALL: This technique provides a search mechanism for Functional Unit(s) of a specific type. A Functional Unit Description Record containing no attributes may be placed in the Service Description Record of the Query Capabilities call. This signifies that the other Equipment is to reply with a Service Description Record containing all the registered Functional Unit Description Records of the same Functional Unit type.

- MATCH CALL: This technique provides a search mechanism for a Functional Unit(s) which matches a specific set of attribute values. A Functional Unit Description Record specifying one or more attributes may be placed in the Service Description Record of the Query Capabilities call. This signifies that the other Equipment is to reply with a Service Description Record containing all the registered Functional Unit Description Records having these (and other) attribute values. The specificity of the search may be adjusted by adding or subtracting attribute values as may be required.

The Salutation Architecture recognizes a need for a global search mechanism to extend the point-to-point technique described above. This technique will be described in subsequent releases of the Architecture.

The Salutation Manager in the Client Equipment passes the received Service Description Record to the Client that requested the *Query Capability*.

2.7.4.Service Request

Once a Client receives the Service Description Record of a Server Salutation Manager, either by Capability Exchange or by any other means, it can start communication with the Server to use the Functional Units.

The following Remote Procedure Call messages are defined to utilize a Service provided by a Server Salutation Manager.

The **local Salutation Manager** is the Salutation Manager which the Client SLM-API is calling, and the **remote Salutation Manager** is any other Salutation Manager. The local Salutation Manager may actually be in a remote piece of Equipment if the Client is calling the SLM-API through Remote Procedure Call.

2.7.4.1.Open Service

The Open Service call message is sent from the Salutation Manager in the Client Equipment to the Salutation Manager in the Server Equipment to request the establishment of a Service Session between the Client and the Functional Unit. It is also used between two Functional Units. The following parameters are included in the *Open Service* call message:

- **Client Service Handle** - Identifies the opening Service session
- **Target Functional Unit Handle** - Identifies the target Functional Unit for which a Service session is to be established
- **Personality Protocol ID** - Identifies the Personality Protocol to be used in the opening Service session
- **Requester SLM-ID** - Identifies the Salutation Manager sending the *Open Service* call message
- **Requester Functional Unit Handle** - Identifies the Client requesting to open a Service session. This parameter is zero if the Client has not registered with the Salutation Manager as a Functional Unit.
- **Credential** - Identifies the user requesting to use the Service provided by the target Functional Unit
- **Verifier** - Authenticates the Credential

Refer to "The Client calls slmGetSLMIDandFUHbyURL() to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

User Identification and Authentication" in section 2.9 for details on the Credential and Verifier parameters.

When the Salutation Manager receives an *Open Service* call message, it returns the *Open Service* reply message with the following parameters:

- **Result Code** - Indicates if the requested Service is accepted or rejected

  If the request is rejected, the reason code is returned. An example of an error reason is "resource temporarily not available". This error may occur when a necessary resource is busy servicing another Client or local use and the Service has no mechanism to queue additional requests.

- **Server Service Handle** - Identifies the opening Service session. This parameter is zero if the Service request is rejected.

2.7.4.2.Transfer Data

The *Transfer Data* call message contains a command or a response that is sent/received between the Client and the Functional Unit, or between two Functional Units. The message conforms to the format and protocol as defined by the Personality Protocol used in the Service session.

No Remote Procedure Call reply message is returned for *Transfer Data* call message.

2.7.4.3.Close Service

When the Client has completed using the Service, it requests the Salutation Manager to send the *Close Service* call message to the Salutation Manager at the other end of the Service session. The Salutation Manager that receives the *Close Service* call message returns the *Close Service* reply message to acknowledge it.

In exceptional cases, either Salutation Manager may send the Close Service call message to the other Salutation Manager without the request from the Client.

2.7.5.Availability Check

When Salutation Clients need to periodically determine whether another Salutation application is available, the Salutation applications request their respective Salutation Managers to perform an Availability Check between the Salutation Managers. This is effective when a Client sends a long-term request like SubscribeEvent, and a Functional Unit wants to check if a Client still requires (alive to receive) “Event Notice.”

The Availability Check is performed by exchanging Remote Procedure Call messages between the Salutation Managers. The period of the Availability Check is specifiable.

When requested by Salutation application(s), the Salutation Manager periodically sends a Functional Unit called *Check Functional Unit Availability* call message to another Salutation Manager to report the list of Functional Units that are currently registered with the sending Salutation Manager, and to request the list of Functional Units that are currently registered with the receiving Salutation Manager.

When a Salutation Manager receives a *Check Functional Unit Availability* call message, it functions as the Client and returns the *Check Functional Unit Availability* reply message which includes the list of Functional Units that are currently registered with the Salutation Manager.

2.8.SLM-API

The Salutation Architecture defines the **SLM-API**, which is the application programming interface provided by the Salutation Manager to Salutation applications.

Using the SLM-API, Salutation applications can be written to communicate with each other through the Salutation Manager Protocol independently of the transport layer. The SLM-API facilitates Salutation application development and portability.

The following functions are provided by the Salutation Manager to Clients and Servers through the SLM-API:

2.8.1.Service Registration

The Server, or even the Client when necessary, calls ***slmRegisterCapability()*** or ***slmUnregisterCapability()*** to register or unregister itself, respectively, with the local Salutation Manager as a Functional Unit.

While the Client is registered as a Functional Unit its capability may be included in the response to a

Query Capability or *Search Capability* request (described below) from another Client. Also, the Client may receive an *Open Service* request (described below) from a Client or another Functional Unit at any time.

2.8.2.Service Discovery

The Client calls ***slmSearchCapability()*** or **slmSearchCapability2()** to ask the local Salutation Manager to search for Salutation Managers containing registered Functional Units with a particular capability. The local Salutation Manager returns the list of SLM-IDs or the list of pair of SLM-ID and FU handle to the Client. The Salutation Manager with the SLM-ID included in the list has a Functional Unit that can provide the Service requested by the Client.

If the Client does not specify any particular capability requirements when it calls ***slmSearchCapability()*** or **slmSearchCapability2()**, the list of all the SLM-ID or the list of pair of SLM-ID and FU handle of Salutation Managers known to the local Salutation Manager is returned.

The Client calls ***slmQueryCapability()*** to discover what Functional Units are registered (and the capabilities of the registered Functional Units) at a Salutation Manager specified by the Client. The Client may specify either the local Salutation Manager or a remote Salutation Manager. The specified Salutation Manager automatically responds to this query as well as to the ***slmSearchCapability()*** or **slmSearchCapability2()** without involving the specified Salutation Manager.

2.8.3.Service Request

The Client or Functional Unit calls ***slmOpenService()*** to ask the local Salutation Manager to initiate a Service session with a specific Functional Unit registered at either the local Salutation Manager or a remote Salutation Manager.

The Salutation Manager, with which the specified Functional Unit is registered, calls ***fnOpenService()***, a function exported by the Functional Unit, to notify the Functional Unit of this *Open Service* request. The Functional Unit may either accept or reject the request. The result is communicated back to the Client through the return parameter of the ***slmOpenService()*** call.

Once the Service session is established, the Client or the Functional Unit calls ***slmTransferData()*** to ask its respective local Salutation Manager to send Client-specific data to the other end of the Service session.

The Salutation Manager at the other end (which can be the same Salutation Manager) calls ***fnReceiveData()***, a function exported to the Client or the Functional Unit, to pass the client-specific data received from the other end of the Service session.

The Client and the Functional Unit repeat the *Transfer Data* process as many times as required.

The Client or the Functional Unit calls ***slmCloseService()*** to ask the local Salutation Manager to terminate the Service session.

The specified Salutation Manager calls ***fnCloseService()***, a function exported to the Functional Unit or the Client, to notify that the Service Session is terminated.

2.8.4.Availability Check

The Functional Unit calls ***slmStartAvailabilityCheck()*** to ask the local Salutation Manager to periodically check if a specific Functional Unit, registered at either the local Salutation Manager or a remote Salutation Manager, is up and running (still registered).

When the Salutation Manager finds that the specified Functional Unit is no longer available, it calls ***fnNotifyFUUnavailability()***, a function exported to the Functional Unit, to notify of this fact.

The Functional Unit calls ***slmCancelAvailabilityCheck()*** when the Availability Check is no longer necessary.

2.8.5.Miscellaneous

The Client calls ***slmGetVersion()*** to get the version number of the SLM-API architecture implementation. The version number returned by the Salutation Manager shall be ‘00020001’ in hexadecimal under the Salutation Architecture V2.1.

The Client calls ***slmGetLocalSLMID()*** to get the SLM-ID of the local Salutation Manager.

The Client calls ***slmGetSLMIDbyAddr()*** to get the SLM-ID of the Salutation Manager in the form of its network address.

The Client calls ***slmGetSLMIDandFUHbyURL()*** to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

2.9.User Identification and Authentication

The concept of **user identification** is often paired with **authentication**, since some Services are available only to selected users. Service requests made by a user may be later changed or canceled only by the same user. Also, when an event associated with a particular user occurs, a notification is sent to the user, regardless of the user’s location.

The following two parameters are defined by the architecture to address the user identification and authentication requirements:

- **Credential -** Credential identifies a particular user.
- **Verifier -** Verifier authenticates the credential parameter.

Each is a structure whose first parameter is the Authentication Flavor which indicates what type of user identification and authentication method is used. The only two Authentication Flavors defined in this release of the Architecture are **NULL** and **User ID and Password**. NULL occurs when the Credential/Verifier parameter structure contains nothing else. User ID and Password occurs when the rest of the Credential/Verifier parameter structure contains the User ID or Password respectively of a specific user.

When a Client issues an Open Service request to a Functional Unit, it specifies the Credential and Verifier parameters. These parameters are passed to the target Functional Unit through Salutation Manager(s). The target Functional Unit can use these parameters to authenticate the user associated with the Client.

The following attributes are included in the Functional Unit Description Record of all types of Functional Units when the Functional Unit Description Record is registered with a Salutation Manager.

- **Authentication Flavors -** This attribute contains the list of authentication flavors supported by the Functional Unit.

  Under this release of the architecture, the value of this attribute shall be one of the following.

  - **No Authentication**

    The list is empty, or contains only the "NULL" authentication flavor.

    When a Client issues an Open Service request to the Functional Unit, it does not need to specify the User ID and Password of the user associated with the Client. The Functional Unit ignores the Credential and Verifier parameters in the Open Service request.

  - **Optional Authentication**

    The list contains both "NULL" and "User ID and Password" authentication flavors.

    When a Client issues an Open Service request to the Functional Unit, it is optional to specify the User ID or Password of the user associated with the Client. The Functional Unit authenticates the user associated with the Client if the User ID and Password are specified. The Functional Unit typically restricts available Services if the User ID and Password are not specified.

  - **Mandatory Authentication**

    The list contains only "User ID and Password"

    When a Client issues an Open Service request to the Functional Unit, it needs to specify the User ID and Password of the user associated with the Client. The Functional Unit authenticates the user using the specified User ID and Password.

    A special Functional Unit, **[Client] Functional Unit**, is defined to represent the Client. The User ID attribute is included in the Functional Unit Description Record of [Client] Functional Unit when the Functional Unit Description Record is registered with an Salutation Manager.

- **User ID -** Contains the **User ID** of the user associated with the Client. The network login name is recommended for use as the User ID, if applicable. However the implementation may choose to use another scheme. For example, the full name of the user, or even the user class name such as "general" or "administrator" may be used.

The User ID shall be NULL if no user is associated with the Client.

Not all Clients need to register themselves as [Client] Functional Units. However, if registered, it is possible for a Client to search for a specific User ID in the network by using the ***slmSearchCapability()*** or **slmSearchCapability2()** function.

2.10.Flow of Salutation Manager API Processing

This section depicts the interactions between a Client and a Functional Unit beginning with a Capabilities Request, followed by an Availability Check and terminating with a Service Request. This sequence will first be shown for the case where a Functional Unit and the Client are registered with the same Salutation Manager (Local Service Discovery). Then the case where the Functional Unit and the Client are registered on different Salutation Managers will be depicted (Remote Service Discovery).

2.10.1.Local Service Discovery

If the Client specifies the local Salutation Manager in a *Query Capability* request, the flow is as shown in Table 2-1.

Table 2-1: Local Salutation Manager in a Query Capability

<table>
<tr><th>Client</th><th>Salutation Manager</th><th>Functional Unit</th></tr>
<tr><td></td><td colspan="2"><== slmRegisterCapability() call
slmRegisterCapability() return ==></td></tr>
<tr><td colspan="2">slmQueryCapability() call ==>
<== slmQueryCapability() return</td><td></td></tr>
</table>

If the local Salutation Manager is specified in the Start Life Check request, the flow is as shown in Table 2-2.

Table 2-2: Local Salutation Manager in Start Life Check Request

Client Functional Unit	Salutation Manager	Service Functional Unit
slmStartLifeCheck() call ==> <== slmStartLifeCheck() return		
	<== slmStartLifeCheck() call slmStartLifeCheck() return ==>	
(If the Service Functional Unit is found to be unavailable) <== fnNotifyFUUnavailability() call fnNotifyFUUnavailability() return ==>		
	(If the Client Functional Unit is found to be unavailable) fnNotifyFUUnavailability() call ==> <== fnNotifyFUUnavailability() return	
(Otherwise, eventually ...) slmCancelLifeCheck() call ==> <== slmCancelLifeCheck() return		
	(Otherwise, eventually ...) <== slmCancelLifeCheck() call slmCancelLifeCheck() return ==>	

If the local Salutation Manager is specified in the Open Service, Transfer Data and Close Service request, the flow is as shown in Table 2-3.

Table 2-3: Local Salutation Manager in Open Service, Transfer Data and Close Service Request

<table>
<tr><th>Client
(or Functional Unit)</th><th>Salutation Manager</th><th>Functional Unit</th></tr>
<tr><td colspan="2">slmOpenService() call ==></td><td></td></tr>
<tr><td></td><td colspan="2">fnOpenService() call ==>
<== fnOpenService() return</td></tr>
<tr><td colspan="2"><== slmOpenService() return</td><td></td></tr>
<tr><td colspan="2">slmTransferData() call ==></td><td></td></tr>
<tr><td></td><td colspan="2"><== slmTransferData() return
fnReceiveData() call ==>
<== fnReceiveData() return</td></tr>
<tr><td colspan="2"><== slmTransferData() return</td><td></td></tr>
<tr><td></td><td colspan="2"><== slmTransferData() call</td></tr>
<tr><td colspan="2"><== fnReceiveData() call
fnReceiveData() return ==></td><td></td></tr>
<tr><td></td><td colspan="2">slmTransferData() return ==></td></tr>
<tr><td colspan="3">:</td></tr>
<tr><td colspan="2">slmCloseService() call ==></td><td></td></tr>
<tr><td></td><td colspan="2">fnCloseService() call ==>
<== fnCloseService() return</td></tr>
<tr><td colspan="2"><== slmCloseService() return</td><td></td></tr>
</table>

2.10.2.Remote Service Discovery

The flow of Remote Service Discovery messages and calls are depicted in Table 2-4.

Table 2-4: Remote Service Discovery Flow

<table>
<tr><th>Client</th><th>Client-side Salutation Manager</th><th>Service-side Salutation Manager</th><th>Functional Unit</th></tr>
<tr><td></td><td colspan="2">Exchange SLM-ID call ==>
<== Exchange SLM-ID reply
(may be broadcasted if supported)</td><td></td></tr>
<tr><td></td><td></td><td colspan="2"><== slmRegisterCapability() call
slmRegisterCapability() return ==></td></tr>
<tr><td colspan="2">slmSearchCapability() call ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Query Capability call ==>
<== Query Capability reply
:
(This step is repeated for each known SLM.
The reply data are cached for the next step.)</td><td></td></tr>
<tr><td colspan="2"><== slmSearchCapability() return</td><td></td><td></td></tr>
<tr><td colspan="2">slmQueryCapability() call ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Query Capability call ==>
<== Query Capability reply
:
(This step is optional, depending on the caching capability of the Client's Salutation Manager.)</td><td></td></tr>
<tr><td colspan="2"><== slmQueryCapability() return
:
(This step is repeated for each Salutation Manager found by the Search Capability.
The Salutation Manager returns the cached data.)</td><td></td><td></td></tr>
</table>

Remote Service Availability Check flow is illustrated in Table 2-5.

Table 2-5: Remote Availability Check Flow

<table>
<tr><th>Client Functional Unit</th><th>Client-side Salutation Manager</th><th>Service-side Salutation Manager</th><th>Service Functional Unit</th></tr>
<tr><td colspan="2">slmStartLifeCheck(receiver, ...) call ==>
<== slmStartLifeCheck() return</td><td colspan="2"><== slmStartLifeCheck(sender, ...) call
slmStartLifeCheck() return ==></td></tr>
<tr><td></td><td colspan="2"><== Check Functional Unit Availability call

(periodically sent)
(This step is omitted if the local Salutation Manager is specified.)</td><td></td></tr>
<tr><td colspan="2">(If the Service Functional Unit is found to be unavailable)

<== fnNotifyFUUnavailability() call
fnNotifyFUUnavailability() return ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Check Functional Unit Availability reply ==>

(periodically sent)
(This step is omitted if the local Salutation Manager is specified.)</td><td></td></tr>
<tr><td></td><td></td><td colspan="2">(If the Client Functional Unit is found to be unavailable)

fnNotifyFUUnavailability() call ==>
<== fnNotifyFUUnavailability() return</td></tr>
<tr><td colspan="2">(Otherwise, eventually ...)

slmCancelLifeCheck() call ==>
<== slmCancelLifeCheck() return</td><td colspan="2">(Otherwise, eventually ...)

<== slmCancelLifeCheck() call
slmCancelLifeCheck() return ==></td></tr>
</table>

The Remote Open Service, Transfer Data and Close Service Request flow of messages and calls is illustrated in Table 2-6.

Table 2-6: Remote Open Service, Transfer Data and Close Service Request

<table>
<tr><th>Client
(or Functional Unit)</th><th>Client-side Salutation Manager</th><th>Service-side Salutation Manager</th><th>Functional Unit</th></tr>
<tr><td colspan="2">slmOpenService() call ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Open Service call ==></td><td></td></tr>
<tr><td></td><td></td><td colspan="2">fnOpenService() call ==>
<== fnOpenService() return</td></tr>
<tr><td></td><td colspan="2"><== Open Service reply</td><td></td></tr>
<tr><td colspan="2"><== slmOpenService() return</td><td></td><td></td></tr>
<tr><td colspan="2">slmTransferData() call ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Transfer Data call ==></td><td></td></tr>
<tr><td colspan="2"><== slmTransferData() return</td><td colspan="2">fnReceiveData() call ==>
<== fnReceiveData() return</td></tr>
<tr><td></td><td></td><td colspan="2"><== slmTransferData() call</td></tr>
<tr><td></td><td colspan="2"><== Transfer Data call</td><td></td></tr>
<tr><td colspan="2"><== fnReceiveData() call
fnReceiveData() return ==></td><td colspan="2">slmTransferData()return ==></td></tr>
<tr><td colspan="4">:</td></tr>
<tr><td colspan="2">slmCloseService() call ==></td><td></td><td></td></tr>
<tr><td></td><td colspan="2">Close Service call ==></td><td></td></tr>
<tr><td></td><td></td><td colspan="2">fnCloseService() call ==>
<== fnCloseService() return</td></tr>
<tr><td></td><td colspan="2"><== Close Service reply</td><td></td></tr>
<tr><td colspan="2"><== slmCloseService() return</td><td></td><td></td></tr>
</table>

3.SLM-ID Generation

Each Salutation Manager has a universally unique identifier, SLM-ID, which is used by Clients and other Salutation Managers to uniquely identify the Salutation Manager in a transport independent way. This chapter defines how a Salutation Manager implementation generates its own SLM-ID.

Each Salutation Manager generates its own SLM-ID value. SLM-ID is a 16-octet-long opaque OCTET STRING. In order to guarantee the universal uniqueness of a generated value, Salutation Manager shall use one of the schemes defined in the following table.

Table 3-1: Salutation Manager ID Generation

<table>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th></tr>
<tr><td>01</td><td colspan="15">Sub-scheme ID (in the 2nd octet), followed by Manufacturer ID (in the 3rd through Nth octets), a NULL octet (in the “N+1”th octet), and a unique value (in the “N+2”th through 16th octets) generated under a manufacturer-dependent rule

Sub ID Manufacturer ID
01 <i>Organizationally Unique Identifier (OUI)</i> administered by IEEE, 3 octets (The list is available at <URL:ftp://ftp.ieee.org/info/stds/info.stds.oui>)
02 <i>Private Enterprise Number</i> administered by Internet Assigned Numbers Authority (IANA), 2 octets (The list is available at <URL:ftp://ftp.isi.edu/in-notes/iana/assignments/enterprise-numbers>)
03 <i>Internet Domain Name</i> encoded in ISO 8859-1 coded character set, <i>n</i> octets, (e.g. “ibm.com”) This sub scheme cannot be used if the domain name is too long to leave sufficient octets for the unique value part that follows the Manufacturer ID.

(This scheme may be used by any Equipment.)</td></tr>
<tr><td>02</td><td colspan="15">MAC address, followed by a random value through the 16th octet

(This scheme may be used by Equipment that has LAN adapters.)</td></tr>
<tr><td>03</td><td colspan="15">IrLAP device address (4 octets), followed by a random value through the 16th octet

(This scheme may be used by Equipment that supports the IrDA protocol.)</td></tr>
<tr><td>04</td><td colspan="15">Telephone number string starting with the country code without any intervening non-numeric characters and spaces, followed by a random value through the 16th octet
- Telephone number string uses ISO 8859-1 coded character set
- The random value part may be generated based on the ISDN sub-address, if applicable.

(This scheme may be used by Equipment that has its own telephone number.)</td></tr>
<tr><td>05</td><td colspan="15">IP address (4 octets), followed by a random value through the 16th octet.

(This scheme may be used by Equipment that has its own IP address.)</td></tr>
<tr><td>06</td><td colspan="15">IPX address (10 octets), followed by a random value through the 16th octet.

(This scheme may be used by Equipment that has its own IPX address.)</td></tr>
</table>

Notes:

- The value of the first octet is shown in hexadecimal.

- The first octet of SLM-ID indicates which scheme is used. However, this is only indicated to avoid the risk of generating the same SLM-ID value using different schemes. Once the SLM-ID is generated, the entire SLM-ID is opaque. Salutation Managers or Clients shall not retrieve any information out of an SLM-ID value.

- If more than one scheme is applicable, the Salutation Manager implementation may choose any scheme to generate its SLM-ID.

- The “IrLAP device address” of the scheme 02 is not universally unique. However, it should be unique in practice considering the mobile nature of the IrDA protocol and the sufficiently long (11 octets) random value.

The SLM-ID is expected to be relatively static between successive initializations of the Salutation Manager, because certain Salutation Manager implementations may choose to cache the SLM-ID and Service Description Record received from other remote Salutation Managers for a certain period.

NULL SLM-ID is defined as the SLM-ID in which the value is zero in all the sixteen octets. It is used only in the parameters of SLM-API functions for a special purpose as described in the “ in section 5.

4.Salutation Manager Protocol Specification

The Salutation Manager communicates with other Salutation Managers to perform its role as a Service broker. The Salutation Manager-to-Salutation Manager communication protocol is defined by the Architecture as the Salutation Manager Protocol.

The Salutation Manager Protocol uses Sun Microsystems' Open Networking Computing Remote Procedure Call version 2 protocol. Remote Procedure Call/reply messages are exchanged between Salutation Managers according to the Salutation Manager Protocol.

This chapter defines the format and protocol of Remote Procedure Call messages exchanged between Salutation Managers.

4.1.Remote Procedure Call Message Description

Two parameters of the Remote Procedure Call message are already set. The **remote program number** for the Salutation Manager is 300558 (decimal) and the **remote program version number** for the Salutation Manager is two (2) for this version of the Salutation Architecture.

Subsequent sections describe each Remote Procedure Call message in detail.

4.1.1.Get Minor Version

The Salutation Manager sends the ***Get Minor Version*** call message to a remote Salutation Manager to query the minor version number that the remote Salutation Manager supports.

The called Salutation Manager returns the **Minor Version Number** it supports. The returned value shall be one (1) under this release of the Salutation Architecture.

It is intended that the major version number remains the same and only the minor version number is incremented when the Salutation Manager Protocol is enhanced with backward compatibility in the future.

The *Get Minor Version* call message includes no parameter.

4.1.2.SLM-ID Exchange

When a Salutation Manager is brought up (for example, the Equipment is powered on), the Transport Managers associated with the Salutation Manager find the SLM-ID of other remote Salutation Managers by sending an ***Exchange SLM-ID*** call message to each known remote Salutation Manager.

The *Exchange SLM-ID* call message includes the parameter **SLM-ID** of the Salutation Manager sending the *Exchange SLM-ID* call message.

The *Exchange SLM-ID* reply message includes the parameter **SLM-ID** of the Salutation Manager sending the *Exchange SLM-ID* reply message.

4.1.3.Capability Exchange

Upon a request from a Client, the Salutation Manager sends a ***Query Capability*** message to another Salutation Manager to discover the Services available from the Service Equipment. The Salutation Manager in the Service Equipment receiving this message responds by sending back a reply message.

The *Query Capability* call message includes the parameter **Service Description Record.** The Client specifies in what capabilities it is interested in this data structure.

The *Query Capability* reply message includes the parameter **Service Description Record.** The Salutation Manager in Service Equipment specifies what capabilities are actually available in this data structure.

The format of the Service Description Record is described in section 2.3.1.

In the Service Equipment, the Salutation Manager maintains a Service Description Record that consists of all the registered Functional Unit Description Records.

When the Salutation Manager receives a *Query Capability* message, it compares the Service Description Record in the message with its own Service Description Record, then constructs a new Service Description Record to be returned in the reply message.

The rule to compare and construct a Service Description Record is described below.

4.1.3.1.Handling the Query Capability Message

The Service Description Record in a *Query Capability* call message, called **requested Service Description Record**, contains zero or more Functional Unit Description Record(s).

The Salutation Manager in the Service Equipment maintains a Service Description Record, called **registered Service Description Record**, that contains all the currently registered Functional Unit Description Record(s).

When the Salutation Manager in the Service Equipment receives a *Query Capability* call message, it compares the requested Service Description Record with the registered Service Description Record and builds a new Service Description Record, called the **reply Service Description Record**, which contains zero or more Functional Unit Description Record(s).

Similarly, Functional Unit Description Records in requested, registered, or reply Service Description Records are called requested, registered, or reply Functional Unit Description Records respectively.

A registered Functional Unit Description Record contains all the Attribute Records that are defined for the type of the Functional Unit. Part 2 of the architecture specification describes what attributes are defined for each type of Functional Unit.

A requested Functional Unit Description Record contains zero or more Attribute Record(s), not necessarily all the defined Attribute Records.

The Salutation Manager in the Service Equipment handles a received *Query Capability* call message by using the algorithm illustrated in Figure 4-1.

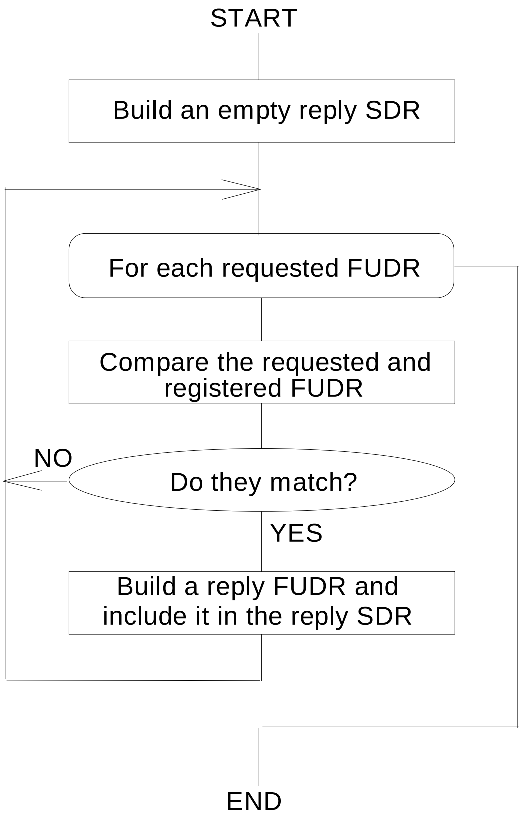

Figure 4-1: Algorithm Used Handling a Query Capability Message

In the following sections, each step is described in more detail.

4.1.3.2.Comparison of Functional Unit Description Records

If no Functional Unit Description Record exists in the requested Service Description Record, no reply Functional Unit Description Record is built. Otherwise, proceed to the following steps that are repeated for each **requested** Functional Unit Description Record in the requested Service Description Record. If there is a requested Functional Unit Description Record that has an **All Call Functional Unit ID**, the requested Functional Unit Description Record must be the only Functional Unit Description Record in the requested Service Description Record.

For a given requested Functional Unit Description Record, the following steps are repeated for each **registered** Functional Unit Description Record.

Step 1: Compare the Functional Unit ID fields

The requested Functional Unit Description Record has the **All Call** Functional Unit ID, or

The registered Functional Unit Description Record has the same Functional Unit ID as the requested Functional Unit Description Record.

If both of the above conditions fail, the two Functional Unit Description Records do not match (MISMATCH). Continue to the *next registered Functional Unit Description Record*. Otherwise, proceed to the following steps.

Step 2: Compare the Attribute Records in the requested Functional Unit Description Record with those in the registered Functional Unit Description Record according to the rule described in the "Comparison of Attribute Record" section below.

Step 3: If the two Functional Unit Description Records mismatch, continue to the *next registered Functional Unit Description Record*. Otherwise, proceed to the following step.

Step 4: A reply Functional Unit Description Record with Attribute Records is built. It contains the union of all the Attribute Records in the requested Functional Unit Description Record and all the Attribute Records in the registered Functional Unit Description Record . The "Comparison Result" field of each Attribute Record is set to one of the following values according to the rule described in the "Comparison of Attribute Record" section below.

- True (The comparison was performed and two Attribute Records matched.)
- Attribute Not Registered
- Compare Function Not Defined
- Wrong Data Type
- Attribute Not Requested

4.1.3.3.Comparison of Attribute Records

Given a registered Functional Unit Description Record and a requested Functional Unit Description Record, the following steps are taken to find out whether the two Functional Unit Description Records match. If any of the steps indicates MISMATCH, the two Functional Unit Description Records do not match.

Step 1: For each Attribute Record in the requested Functional Unit Description Record, do the following until MISMATCH is detected or until all Attribute Records are processed:

a) Check for the existence of Attribute Records in the registered Functional Unit Description Record

- If an Attribute Record with the same Attribute ID exists in the registered Functional Unit Description Record, proceed to the following steps,
- Otherwise, this attribute is considered to match. Continue to the *next attribute*. (This attribute is added to the reply Functional Unit Description Record with the Comparison Result field set to "Attribute Not Registered".)

b) Determining a Compare Function ID to Use

A Compare Function ID, which indicates a compare function, is selected as follows:

- If a value in the Compare Function ID field of the requested Attribute Record is not UNSPECIFIED, the value in the field is used.
- Otherwise, a value in the Compare Function ID field of the registered Attribute Record is used. (The value must not be UNSPECIFIED.)

c) Locating a compare function corresponding to the selected Compare Function ID

A compare function with the Compare Function ID is searched.

- If the Compare Function ID is not known to the Salutation Manager, the Comparison Result field is filled in with a reason code indicating "Compare Function Not Defined". In this case, the two records are considered to match. Continue to the *next attribute*.
- Otherwise, proceed to the following steps.

d) Comparison of values in Attribute Records

The compare function is called with two arguments: 1) the Attribute Record of the registered Functional Unit Description Record, and 2) the Attribute Record of the requested Functional Unit Description Record.

- If the comparison is performed and the result of the comparison is FALSE, the two records do not match (MISMATCH).
- Otherwise, the comparison is not performed, or the result is TRUE. In these cases, the two records are considered to match. The Comparison Result field is filled in with TRUE or a reason code indicating why the Attribute Records were not compared. The reason code **Wrong Data Type** is defined (values of the attribute have a wrong data type for the compare function).

Step 2: If all the requested Attribute Records match, the two Functional Unit Description Records match, and a reply Functional Unit Description Record is built. Otherwise, the two Functional Unit Description Records do not match, and no reply Functional Unit Description Record is built.

Step 3: If any Attribute Record in the registered Functional Unit Description Record has not been compared because the corresponding record does not exist in the requested Functional Unit Description Record, the Compare Result field is filled with a reason code indicating "Attribute Not Requested". Figure 4-2 illustrates the compare Attribute Compare function.

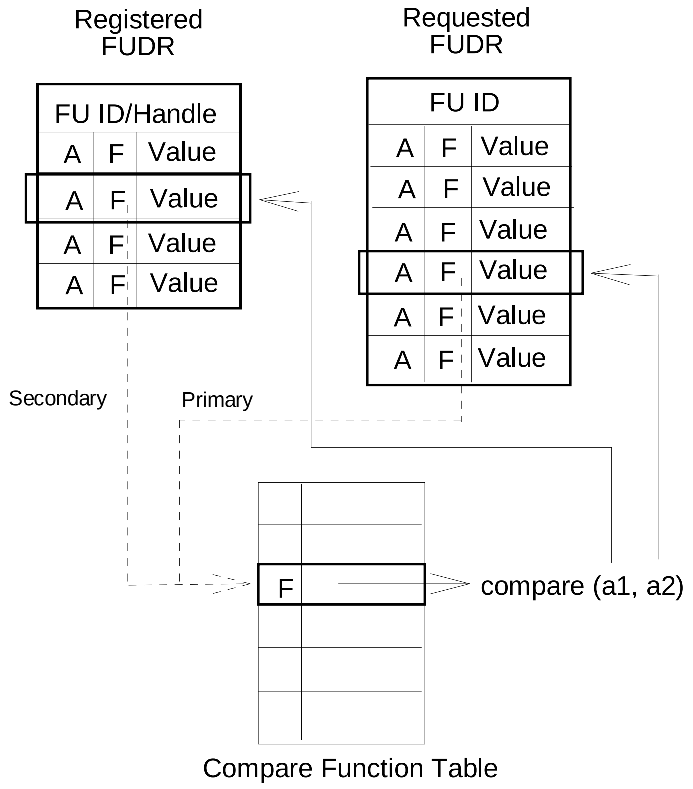

Figure 4-2: Comparing Attribute Records

4.1.4.Service Request: Session Management

Open Service and ***Close Service*** request messages and replies are defined for session management.

The *Open Service* request is typically initiated by a Client to a Functional Unit, but may also be used between two Functional Units. *Open Service* is used to request the opening of a Service session.

A reply message flows back as the response to an *Open Service* message. A positive reply establishes a Service session. A negative reply may also be received, for example, “resource temporarily not available” or “target functional unit handle not found”.

The *Open Service* call message includes the following parameters.

- **Client Service Handle**

The sending Salutation Manager generates a non-zero value, Client Service Handle, which uniquely identifies the opening Service session among all the Service sessions the Salutation Manager currently has open.

This Client Service Handle will be included in all the *Transfer Data* and *Close Service* call messages subsequently received within the opening Service session from the Salutation Manager at the other end of the Service session.

- **Target Functional Unit Handle**

This handle identifies the Functional Unit for which a Service session is to be established.

- **Personality Protocol ID**

This protocol identifies the Personality Protocol to be used in the opening Service session. The value for each Personality Protocol is defined by Part 2 of the architecture specification document.

- **Requester SLM-ID**

The Requester SLM-ID identifies the Salutation Manager sending the *Open Service* call message. This Salutation Manager is the local Salutation Manager of the Client that is requesting the opening of a Service session.

- **Requester Functional Unit Handle**

This handle identifies the Client that is requesting the opening of a Service session. If the Client has not registered itself with the Salutation Manager, zero (0) shall be specified.

Once the Open Service request is accepted, the Server Service Handle in the Open Service reply will be used to communicate with each other. So multiple Open requests can be recognized by each Server Service Handle assigned to each Open Service request.

- **Credential**

Credential identifies the user that is requesting to use the Service provided by the target Functional Unit.

This parameter is a structure of which the first parameter is “Authentication Flavor”, which is either NULL or “User ID and Password”. If no user is associated with the requester, or if the target Functional Unit does not require verification, NULL is specified. Otherwise, “User ID and Password” is specified in the “Authentication Flavor”, and the rest of this Credential parameter contains the “User ID” of the requesting user.

- **Verifier**

This parameter is used by the target Functional Unit to authenticate the Credential.

This parameter is a structure of which first parameter is “Authentication Flavor”, which is either NULL or “User ID and Password”. If no user is associated with the requester, or if the target Functional Unit does not require verification, NULL is specified. Otherwise, “User ID and Password” is specified in the “Authentication Flavor”, and the rest of this Verifier parameter contains the “Password” associated with the “User ID”.

Refer to “The Client calls slmGetSLMIDandFUHbyURL() to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

User Identification and Authentication” in section 2.9 for details on the Credential and Verifier parameters.

The *Open Service* reply message includes the following parameters.

- **Result Code**

If the replying Salutation Manager finds an error in the received *Open Service* call message, it sets an error reason code in this parameter. Otherwise, the result code returned from the target Functional Unit is set. The result code indicates whether the *Open Service* request is accepted or the reason if it was rejected. A reason might be, for example, “resource temporarily not available” or “target functional unit handle not found”.

- **Server Service Handle**

The replying Salutation Manager generates a non-zero value, Server Service Handle, which uniquely identifies the opening Service session among all the Service sessions the Salutation Manager currently has open.

This Server Service Handle will be included in all the *Transfer Data* and *Close Service* call messages subsequently received within the opening Service session from the Salutation Manager at the other end of the Service session.

Close Service may be requested by either end of a Service session to the opposite end to close the Service session. A reply message is sent back as the acknowledgment to the *Close Service* request message.

The *Close Service* call message includes the following parameters.

- **Client/Server Service Handle**

This handle was assigned at the exchange of *Open Service* call/reply messages, and identifies the particular Service session to be closed.

Normally, the Client or the Functional Unit that initiated the *Open Service* request initiates also the closing of the Service session. In exceptional cases, the Functional Unit that accepted the *Open Service* request or either side of the Salutation Manager may initiate closing the Service session. Since Close Service request can be issued by a Client or a Service, Client Service Handle or Server Service Handle may be used. If a Client issues Close Service request, Server Service Handle will be used to recognize the target Service. If a Service issues Close Service request, Client Service Handle will be used to recognize the target Client.

- **Reason Code**

The Reason Code indicates one of the following:

- ♦ **Normal**: The Client or the Functional Unit that initiated the Service session has completed using it and is now closing it.
- ♦ **Application Exception**: The Functional Unit that accepted the Service session cannot

continue to provide its Service due to an exceptional condition.

- **Salutation Manager Exception**: The Salutation Manager (of either side of the Service session) cannot continue to support the Service session due to an exceptional condition.

The *Close Service* reply message includes no parameter.

4.1.5.Service Request: Session Data Transfer

The ***Transfer Data*** message is defined to transfer Client and Functional Unit commands and responses. It flows either way in a Service session.

The *Transfer Data* call message includes the following parameters.

- **Client/Server Service Handle**

This handle was assigned at the exchange of *Open Service* call/reply messages, and identifies a particular Service session. Client Service Handle is used for *Transfer Data* messages sent from the Service to the Client. Server Service Handle is used for *Transfer Data* messages sent from the Client to the Server.

- **Application Data**

This is the Client and Functional Unit data. This data is transparent to the Salutation Manager. The Salutation Manager does not inspect this content.

There is no reply message to a *Transfer Data* call message.

4.1.6.Availability Check

Sometimes a Client needs to make a request that takes a very long time before a response is returned. For example, an administrator of a printer may request the printer to provide notification when the toner in the printer gets low. This type of request is called a **long-term request**.

Before a Client makes a long-term request, it must first register itself to the Salutation Manager as a [Client] Functional Unit. In the following description in this section, such a Client is called the Client Functional Unit while the Functional Unit that receives a long-term request is called the Service Functional Unit.

While a long-term request is effective, it is necessary to periodically check if the Client Functional Unit and the Service Functional Unit are up and running so that the Client Functional Unit will be able to know that its long-term request is no longer valid when the Service Functional Unit is de-activated Also, the Service Functional Unit will be able to know that it can ignore the request when the Client Functional Unit is de-activated.

For example, the Client Functional Unit or the Service Functional Unit may be de-activated when its Equipment is restarted or when the program crashes.

The Client Functional Unit and the Service Functional Unit request their respective Salutation Managers to periodically check if the Functional Units are available. The Salutation Managers perform this **Availability Check** between them, and notify the Functional Unit if the associated Functional Unit is found to be unavailable. ***Check Functional Unit Availability*** call/reply messages are defined for the Availability Check protocol.

The basic flow of the Availability Check operation is as follows.

1) If the Client Functional Unit wants to be notified when the Service Functional Unit is de-activated, it tells the local (Client-side) Salutation Manager to expect to receive a *Check Functional Unit Availability* call message periodically. The interval of sending *Check Functional Unit Availability* call messages is specified by the Client Functional Unit and is called the **Check Interval**.

2) The Client Functional Unit sends a command for a long-term request to the Service Functional Unit. The command contains the Check Interval as a parameter.

   Note: If the long-term request is rejected by the Service Functional Unit, the Client Functional Unit tells the local Salutation Manager to stop expecting the *Check Functional Unit Availability* call message.

3) The Service Functional Unit tells the local (Service-side) Salutation Manager to periodically send a *Check Functional Unit Availability* call message.

4) The Service-side Salutation Manager periodically sends a *Check Functional Unit Availability* call message to the Client-side Salutation Manager. The message includes the list of Functional Units that are currently registered with the Service-side Salutation Manager (the Service Functional Unit being one of them).

5) The Client-side Salutation Manager checks if the Service Functional Unit is included in the list of available Functional Units. If the Service Functional Unit is not included, the Salutation Manager notifies the Client Functional Unit of the unavailability of the Service Functional Unit, and stops expecting the *Check Functional Unit Availability* call message.

6) The Client-side Salutation Manager sends a *Check Functional Unit Availability* reply message to the Service-side Salutation Manager. The message includes the list of Functional Units that are currently registered with the Client-side Salutation Manager (the Client Functional Unit being one of them).

7) The Service-side Salutation Manager checks if the Client Functional Unit is included in the list of available Functional Units. If the Client Functional Unit is not included, the Salutation Manager notifies the Service Functional Unit of the unavailability of the Client Functional Unit and stops sending the *Check Functional Unit Availability* call message.

8) If the Client-side Salutation Manager does not receive a *Check Functional Unit Availability* call message for longer than the Check Interval (plus a safe margin), it notifies the Client Functional Unit that the Service Functional Unit is unavailable, and stops expecting the *Check Functional Unit Availability* call message.

9) If the Service-side Salutation Manager does not receive the response to the *Check Functional Unit Availability* for too long a time, it notifies the Service Functional Unit that the Client Functional Unit is unavailable, and stops sending the *Check Functional Unit Availability* call

message. “Too long a time” is longer than the Check Interval (plus a safe margin) since the Availability Check has begun or the last *Check FU Availability* message is received, whichever is the latter.

10) When the long-term request is completed or canceled, both the Client Functional Unit and the Service Functional Unit tell their respective Salutation Managers to stop the exchange of *Check Functional Unit Availability* messages.

The flow described above is applicable when there is only one pair of Client Functional Units and Service Functional Units between a pair of Equipment. In general, given a pair of Equipment, some Client Functional Units in Equipment-A may have made long-term requests to Service Functional Units in Equipment-B while some Client Functional Units in Equipment-B may have made long-term requests to Service Functional Units in Equipment-A. In such cases, the Availability Check protocol is optimized to reduce the number of messages exchanged for the Availability Check.

For example, as shown in Figure 4-3, if:

- Client Functional Unit (A1) made a long-term request to Service Functional Unit (B1) with a 15 second interval Availability Check,
- Client-Functional Unit (A2) made a long-term request to Service Functional Unit (B2) with a 2 minute interval Availability Check,
- Client-Functional Unit (B3) made a long-term request to Service Functional Unit (A3) with a 20 minute interval Availability Check, and
- Client-Functional Unit (B4) made a long-term request to Service Functional Unit (A4) with a 1 minute interval Availability Check.

then, eventually the Salutation Manager in Equipment-B sends a *Check Functional Unit Availability* call message to the Salutation Manager in Equipment-A at the interval of no greater than 15 seconds which is the shortest Check Interval among the four requests. A single exchange of *Check Functional Unit Availability* call/reply message validates the availability of all the eight Functional Units. The number of messages for the Availability Check is not multiplied by the number of long-term requests. No matter how many long-term requests there are between the same pair of Equipment, the Availability Check is performed based on the shortest Check Interval among the long-term requests.

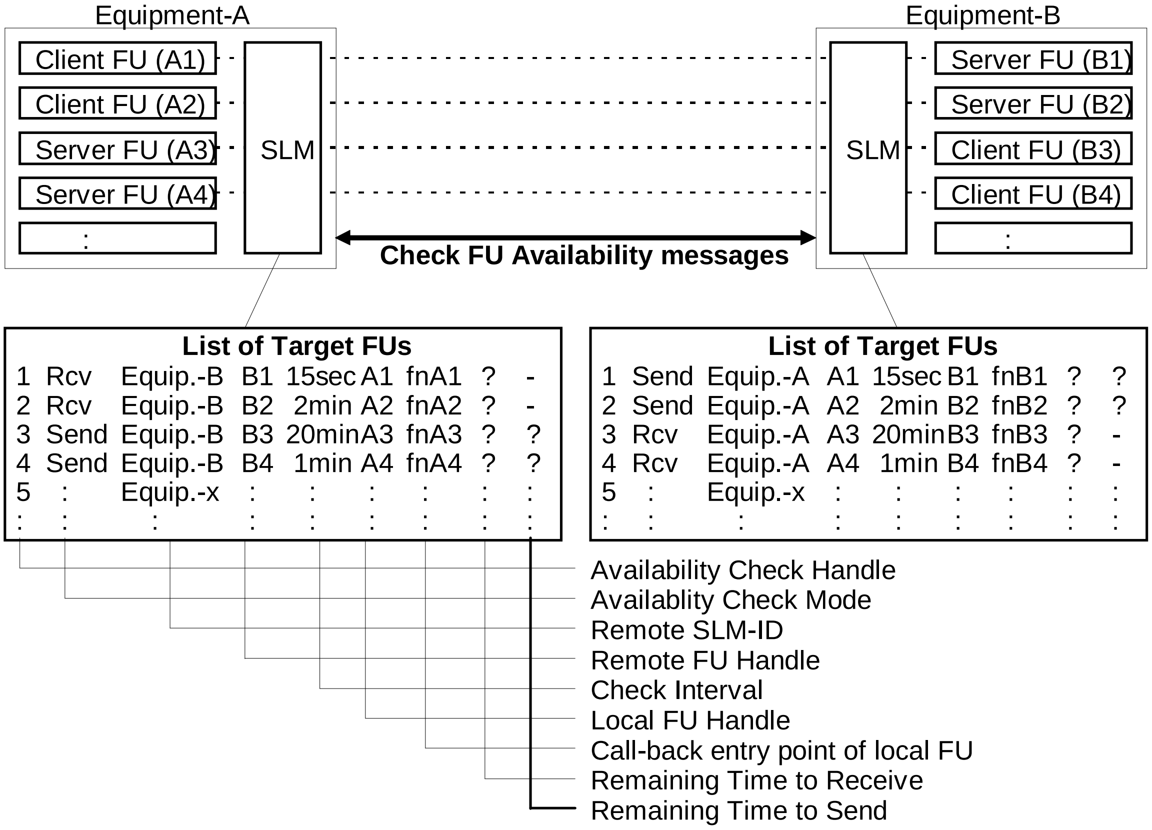

Figure 4-3: Availability Check

The Salutation Manager maintains the **List of Target Functional Units** for the Availability Check. Each entry in the List consists of the following fields as shown in the above figure:

Availability Check Handle

This field uniquely identifies an entry in the List. When a local Functional Unit requests an Availability Check with a remote Functional Unit, the Salutation Manager creates an entry in the List and assigns a unique value to this field.

Availability Check Mode

This field is either Sender or Receiver, and indicates whether the Salutation Manager is to send or to expect *Check Functional Unit Availability* call messages. Given a pair of Functional Units for Availability Check, one shall be Sender and the other shall be Receiver. The local Functional Unit requesting Availability Check specifies the value of this field.

Remote SLM-ID

This field identifies the remote Salutation Manager with which the target remote Functional Unit is registered. The local Functional Unit requesting the Availability Check specifies the value of this field.

Remote Functional Unit Handle

This field identifies the remote Functional Unit which is the target of the Availability Check. The local Functional Unit requesting Availability Check specifies the value of this field.

Check Interval

This field indicates the maximum interval for exchanging *Check Functional Unit Availability* messages. The local Functional Unit requesting Availability Check specifies the value of this field.

Local Functional Unit Handle

This field identifies the local Functional Unit in the Equipment which is the target of the Availability Check. The local Functional Unit requesting Availability Check specifies the value of this field.

Callback Entry Point of Local Functional Unit

The function indicated by this field is called by the Salutation Manager when the remote Functional Unit, the target of the Availability Check, is found to be unavailable. The local Functional Unit requesting the Availability Check specifies the value of this field.

Note that the actual mechanism for the Salutation Manager to call this function depends on the Operating System and/or the implementation of Salutation Manager and may not actually be a "call" interface.

Remaining Time to Receive (in seconds)

This field indicates the amount of time left before the availability of the remote Functional Unit must be validated. Such validation occurs when a *Check Functional Unit Availability* call or reply message is received and the target remote Functional Unit is found in the “List of Available Functional Units” parameter of the received message.

The value in this field is decreased by one every second until it reaches zero. When it reaches zero, the Salutation Manager calls the local Functional Unit‘s callback entry to notify it of the unavailability of the remote Salutation Manager.

The field is initialized and is reset with the "Check Interval" value plus a safe margin whenever the availability of the remote Functional Unit is validated. The “safe margin” value should be determined by Salutation Manager implementations taking the characteristics of the underlying transport into account.

Remaining Time to Send (in seconds)

This parameter is applicable only to the entries for which the Availability Check Mode is the Sender.

This parameter indicates the amount of time left before a *Check Functional Unit Availability* call or reply message must be sent to the remote Salutation Manager to report the availability of the local Functional Unit.

The value in this field is decreased by one every second until it reaches zero. When it reaches zero, the Salutation Manager sends a *Check Functional Unit Availability* call message to the remote Salutation Manager and includes the list of available Functional Units.

This field is initialized and is reset with a value no greater than of the "Check Interval" value whenever a *Check Functional Unit Availability* call or reply message is sent to the remote Salutation Manager identified by the “Remote SLM-ID” field.

Only the "Availability Check Handle" and "Remaining Time" fields are set by the Salutation Manager. All the other field values are given by the local Client/Service-Functional Unit. An entry in the List is created or deleted by the request from the local Client/Service-Functional Unit. In addition, the Salutation Manager automatically deletes the entry when either the remote or the local Functional Unit specified in the entry is found to be unavailable. The Salutation Manager performs the Availability Check using the List as follows.

Whenever the "Remaining Time to Send" field of any Sender entry (entry with "Availability Check Mode" = Sender) becomes zero, the Salutation Manager scans its "List of Target Functional Units" to select all the Sender entries that have the same "Remote SLM-ID" value as that of the expired entry. For each entry in the selected entries, the Salutation Manager then resets the "Remaining Time to Send" with a value no greater than that of the "Check Interval".

The Salutation Manager sends a *Check Functional Unit Availability* call message to the Salutation Manager specified by the "Remote SLM-ID". The message contains the parameter **Requester SLM-ID** which identifies the Salutation Manager sending the *Check Functional Unit Availability* call message and a **List of Available Functional Units** which lists the Functional Unit Handles of Functional Units that are currently registered with the Salutation Manager. The remote Salutation Manager receiving the *Check Functional Unit Availability* call message will know that the Functional Units specified by this parameter are available.

When a *Check Functional Unit Availability* call message is received, the Salutation Manager performs the following two activities. First, the Salutation Manager scans its "List of Target Functional Units" to select all the entries that have the same "Remote SLM-ID" value as that of the sender of the message. Second, for each entry in the selected entries, the Salutation Manager checks if the "Remote Functional Unit Handle" of the entry exists in the "List of Available Functional Units" of the received message. If the "Remote Functional Unit Handle" does not exist, it means that the remote Functional Unit is unavailable. The Salutation Manager notifies the local Functional Unit by calling the callback function and removes the entry from the List. Otherwise, the availability of the remote Functional Unit is validated. The Salutation Manager resets the "Remaining Time to Receive" with the "Check Interval" value (plus a safe margin).

For each entry in the selected entries, if the entry’s “Availability Check Mode” is Sender, the Salutation Manager resets the "Remaining Time to Send" with a value no greater than that of the "Check Interval".

The Salutation Manager sends back a *Check Functional Unit Availability* reply message containing the **List of Available Functional Units** parameter. This is a list of the Functional Unit Handles of Functional Units that are currently registered with the Salutation Manager. The remote Salutation Manager receiving the *Check Functional Unit Availability* reply message will know that the Functional

Units specified by this parameter are available.

When a *Check Functional Unit Availability* reply message is received, the Salutation Manager performs the following steps. The Salutation Manager scans its "List of Target Functional Units" to select all the entries that have the same "Remote SLM-ID" value as that of the sender of the reply message. For each entry in the selected entries, the Salutation Manager checks if the "Remote Functional Unit Handle" of the entry exists in the "List of Available Functional Units" of the received message. If the "Remote Functional Unit Handle" does not exist, the remote Functional Unit is unavailable. The Salutation Manager notifies the local Functional Unit by calling the callback function and removes the entry from the list. Otherwise, the availability of the remote Functional Unit is validated. The Salutation Manager resets the "Remaining Time to Receive" with the "Check Interval" value (plus a safe margin).

If the "Remaining Time to Receive" field of any entry becomes zero, it means that the remote Functional Unit is considered to be unavailable because the *Check Functional Unit Availability* call or reply message to validate the availability of the remote Functional Unit has not been received in too long a time. The Salutation Manager notifies the local Functional Unit by calling the callback function and removes the entry from the list.

4.2.Salutation Manager Protocol Specification in Formal Language

4.2.1.Common Parameters

```
typedef opaque                  sdr<>;     /* Service Description Record */

typedef opaque                  slm_id[16];

enum                            auth_flavor_code
{
    NULL                              = 0,
    USERID_AND_PASSWORD          = 1
};

union                           credential_parm
switch (auth_flavor_code        auth_flavor)
{
    case NULL:                        void;
    case USERID_AND_PASSWORD:    string user_id<>;
    /* in the future, other parameters (e.g. struct) may be added here */
};

union                           verifier_parm
switch (auth_flavor_code        auth_flavor)
{
    case NULL:                        void;
    case USERID_AND_PASSWORD:    string password<>;
    /* in the future, other parameters (e.g. struct) may be added here */
};

enum                            open_service_result_code
{
    OK                                                  = 0,
    TARGET_FU_HANDLE_NOT_FOUND                  = 1,
    PERSONALITY_ID_NOT_SUPPORTED                        = 2,
    RESOURCE_TEMPORARILY_NOT_AVAILABLE                  = 3,
    EXCEED_MAX_NO_OF_SERVICE_SESSIONS           = 4,
    AUTHENTICATION_FLAVOR_NOT_SUPPORTED                 = 5,
    AUTHENTICATION_FAILURE                              = 6,
    SLM_EXCEPTION                                       = 7
    /* Values larger than 127 may be defined for Functional Unit-specific result codes */
};

enum                            close_service_reason_code
{
    NORMAL                      = 0,
    APPL_EXCEPTION              = 1,
    SLM_EXCEPTION               = 2
};
```

4.2.2.Salutation Manager Protocol Specification

```
struct                          open_service_call_args
{
    unsigned int                client_service_handle;
    unsigned int                target_fu_handle;
    unsigned int                personality_protocol_id;
    slm_id                      requester_slm_id;
    unsigned int                requester_fu_handle;
    credential_parm             credential;
    verifier_parm               verifier;
};

struct                          open_service_reply_args
{
    open_service_result_code    result_code;
    unsigned int                server_service_handle;
};

struct                          close_service_call_args
{
    unsigned int                service_handle;
    close_service_reason_code   reason_code;
};

struct                          transfer_data_call_args
{
    unsigned int                service_handle;
    opaque                      appl_data<>;
};

struct                          check_fu_availability_call_args
{
    slm_id                      requester_slm_id;
    unsigned int                fu_handle_list<>;
};

typedef unsigned int            fu_handle_list<>;
```

```
program SMP
{
    version VERS_2
    {
        void
        SMP_NULL(void) = 0;

        unsigned int                    /* one (1) shall be returned under SLA V2.1 */
        SMP_GET_MINOR_VERSION(void) = 1;

        slm_id
        SMP_EXCHANGE_SLM_ID(slm_id) = 2;

        sdr
        SMP_QUERY_CAPABILITY(sdr) = 3;

        open_service_reply_args
        SMP_OPEN_SERVICE(open_service_call_args) = 4;

        void
        SMP_CLOSE_SERVICE(close_service_call_args) = 5;

        void
        SMP_TRANSFER_DATA(transfer_data_call_args) = 6;

        fu_handle_list
        SMP_CHECK_FU_AVAILABILITY(check_fu_availability_call_args) = 7;

    } = 2;  /* Remote Procedure Call program version number for Salutation Manager under SLA V2.X */

} = 300558;  /* Remote Procedure Call program number for the Salutation Manager Protocol */
```

4.3.Salutation Manager Protocol Exception Handling

The following exceptions are handled by the Salutation Manager that either sends or receives an Remote Procedure Call message.

Table 4-1: Salutation Manager Protocol Exception Handling

Exception	**Remote Procedure Call Message**	**Salutation Manager Action**
Unknown Client/Service Service Handle in *Transfer Data* call message	Received by Salutation Manager	Message discarded
Unknown Client/Service Service Handle in *Close Service* call message	Received by Salutation Manager	Normal reply message returned

No reply to *Query Capability* call message (Time out)	Sent by Salutation Manager	Error returned to the Client
No reply to *Open Service* call message (Time out)	Sent by Salutation Manager	Error returned to the Client
No reply to *Close Service* call message (Time out)	Sent by Salutation Manager	Salutation Manager processing continues as if a valid reply message had been received
No reply to *Check Functional Unit Availability* call message (Time out)	Sent by Salutation Manager	Salutation Manager processing continues as if a valid reply message, in which the “List of Available Functional Units” parameter is empty, had been received
No reply to *Exchange SLM-ID* call message (Time out)	Sent by Salutation Manager	No action

4.4.Guidelines on Remote Procedure Call Usage

The following sections discuss discovering Salutation Managers by broadcast Remote Procedure Call and managing transport connections.

4.4.1.Discovering Salutation Managers by Broadcast Remote Procedure Call

When a Salutation Manager is implemented on a platform that supports the Remote Procedure Call broadcast protocol, the Transport Manager can discover other remote Salutation Managers together with their SLM-IDs by broadcasting an *Exchange SLM-ID* call message using the Port Mapper Remote Procedure Call Service. Both a packet-based protocol, like UDP, and the Port Mapper Remote Procedure Call Service are required for this functionality.

This feature is optional to Salutation Manager implementations.

Note: Except for the broadcast *Exchange SLM-ID* message to discover other Salutation Managers, all the Remote Procedure Call messages under the Salutation Manager Protocol must be transferred over a “reliable” transport (e.g. TCP instead of UDP).

4.4.2.Transport Connection Management

The Transport Manager should open or close an underlying transport connection (e.g. a TCP connection) as follows:

The Salutation Manager opens a transport connection before sending a *Query Capability, Check Functional Unit Availability, or Exchange SLM-ID* point-to-point call message. It closes the connection when a reply message is received, or when no reply message is received and a time-out error occurs.

The Client Salutation Manager opens a transport connection before sending an *Open Service* call message to a Service-side Salutation Manager. This connection is used to send succeeding *Transfer Data* call messages and a *Close Service* call message to the Service-side Salutation Manager and to receive a *Close Service* reply message. The Client-side Salutation Manager closes this connection when:

- a negative *Open Service* reply message is received,
- no *Open Service* reply message is received and a time-out error occurs,
- a reply message to a *Close Service* call message it has sent is received,
- no reply message is received and a time-out error occurs after it has sent a *Close Service* call message, or
- a *Close Service* call message is received in the other transport connection (described in the next bullet) associated with the same Service session.

The Server Salutation Manager opens a transport connection back to the Client Salutation Manager, too, after receiving an *Open Service* call message and before sending a positive *Open Service* reply message to the Client Salutation Manager. This connection is used to send succeeding *Transfer Data* call messages. Also, in exceptional cases, the connection is used to send a *Close Service* call message to the Client Salutation Manager and to receive a *Close Service* reply message. This connection is NOT used to send the *Open Service* reply message. The Server Salutation Manager closes this connection when:

- a *Close Service* call message is received,
- a reply message to a *Close Service* call message it has sent is received, or
- no reply message is received and a time-out error occurs after it has sent a *Close Service* call message.

4.5.Guidelines on Specific Transport Usage

Recommendations for the use of specific transports such as TCP/IP and Infrared are discussed in this section.

4.5.1.TCP/IP

The **registered port number** for the Salutation Manager-to-Salutation Manager communication (the Salutation Manager Protocol) is 1605 (decimal).

4.5.2.Infrared

The Salutation Architecture recommends the use of protocol specifications defined by the Infrared Data Association (IrDA) as follows when the Salutation Manager Protocol is implemented over infrared media.

The Salutation Manager Protocol is positioned on top of the TinyTP protocol defined by the IrDA.

The Exclusive Mode of the Link Management Protocol (IrLMP) shall not be used.

The Salutation Manager (actually the Transport Manager) registers the following object to the Information Base through LM-IAS Services:

Class Name = “SLM”, with the following attribute

Attribute Name :“IrDA:IrLMP:LsapSel”

Attribute Type : Integer

Attribute Value : The Salutation Manager’s LSAP-SEL value

The Salutation Manager uses **LM_DiscoverDevices** and **LM_GetValueByClass** Services to discover other Salutation Managers in proximity.

The Salutation Manager uses **TTP_Connect**/**TTP_Disconnect** to open/close a TinyTP connection to another Salutation Manager. Refer to “Transport Connection Management” in section 4.4.2 for the timing of connection/disconnection.

The Salutation Manager uses **TTP_Data** to send an Remote Procedure Call/reply message.

Refer to the IrDA-related reference documents described in “References” in section 1.2 for more detailed information on the protocol specifications defined by the IrDA.

4.6.Guidelines to utilize Service Location Protocol Version 2 for Service Discovery

4.6.1.Overview

The Salutation Manager searches for Service Location Protocol Version 2 (called SLP hereafter) directory agents through multicast, broadcast, or manual configuration. If it finds any, the Salutation Manager will use SLP protocol instead of Salutation Manager Protocol to register and deregister Functional Units with the SLP directory. Furthermore, the Salutation Manager will use SLP to search for services requested by client applications.

The Salutation API is designed to make Salutation applications unaware of the underlying transport and discovery protocols. Therefore the application programmer does not need to know if a directory exists or not. Furthermore, since the SLP directory agent can be a gateway to a LDAP-based directory, the Salutation API and Salutation Manager provide a single application interface to all three of these protocols. Salutation, SLP, and LDAP are all complementary with Salutation providing a single API into each.

4.6.1.1.SLP Components

There are three discreet components to SLP. These are Service Agents (SA), User Agents (UA) and Directory Agents (DA). A Service Agent is a process working on the behalf of one or more services to advertise the services. A User Agent is a process working on the user's behalf to establish contact with a useful service. A Directory Agent collects and caches service advertisements from SAs. The

UA may retrieve service information from a Directory Agent or directly from a Service Agent. The Salutation Manager that is implemented to the specification described in this document will include an SLP SA and UA.

4.6.1.2.SLP Message Types

SLP supports a number of message types. In discussing the basic operation of SLP entities in a Salutation environment, we will describe the use of SLP messages for registering and de-registering services, requesting services, and replying to service requests.

- SrvReg: A Service Agent issues one SrvReg message for each instance of a service that it provides. These messages contain a URL for the service, a set of attribute/value pairs that describe the service, and a lifetime for the service. Each SrvReg message is unicast to each known Directory Agent.
- SrvAck: A Directory Agent returns a SrvAck message when a SrvReg message has been processed. SrvAck messages are unicast to the Service Agent that sent the registration.
- SrvDeReg: A Service Agent issues a SrvDeReg when an instance of a service becomes unavailable. A SrvDeReg message is unicast to each Directory Agent with which the Service Agent has previously registered.
- SrvRqst: A User Agent issues a SrvRqst message to locate a service. The request may include a set of attribute/value pairs for selecting services that meet certain criteria. A SrvRqst message may be unicast to a Directory Agent or multicast to all Service Agents.
- SrvRply: A Directory Agent or Service Agents respond with SrvRply when a service provided matches the criteria specified in a SrvRqst. A SrvRply that contains no URLs may be generated in response to a SrvRqst message.

SLP also supports messages for requesting the attributes of a specific service, requesting a list of all available services, and for Directory Agents to advertise themselves. Complete explanation of these messages can be found in the Service Location Protocol Version 2 specification (RFC-2608)

4.6.1.3.How Salutation Utilizes SLP

There are a different ways a SA/UA may discover a DA. In addition to statically configuring the SA/UA with the address of the DA, a SA/UA may request the address of a DA using the Dynamic Host Configuration Protocol (DHCP). While these two options exist, it is important that SLP be able to operate without manual configuration. For this reason, a SA/UA may use the SrvRqst to find a DA. The SrvRqst is multicast to the IANA assigned multicast address for Service Location Protocol. The "service" requested in this message is called "directory-agent". Because this is a multicast request, it may receive more than one unicast reply. The resulting list of directory agents can then be used for registering/deregistering services and for other service requests.

As part of the FU registration process, a Service Agent sends a SrvReg message to each Directory Agent it has discovered. This registration contains the service URL for the specific instance of the Functional Unit (FU), as well as attribute/value pairs that describe the capabilities of the FU. Directory Agents that have been configured in the network cache these registrations. Once a registration has been cached, a DA responds with a SrvAck message. Service registrations also contain a lifetime. If the service has become unavailable but was unable to de-register itself, lifetime values allow a DA to expire cached registrations. This situation should only occur if an SA is unable to issue an SrvDeReg message. During normal operation SAs will periodically refresh their registrations with subsequent SrvReg messages. These "refresh" messages are not required to contain a full set of attributes, though may contain updated information if the values have changed.

When a service is required, a UA makes requests from one of DAs it has discovered. Let us look at printers as an example. If a UA is trying to locate a printer service, a SrvRqst is constructed. This message contains the service type "printer", with an optional list of attributes and values. The attribute/value pair describes the type of printer desired. This message is unicast to a DA. The DA receiving the SrvRqst performs a lookup on the cached registrations, attempting to match the attributes and values requested. A SrvRply is then unicast to the requesting UA. This reply contains zero or more service URLs, depending on the match results. The client may then use the service URL to locate the printer.

It is also possible that no Directory Agents have been discovered. If this is the case, a UA will multicast an SLP SrvRqst to all available Service Agents. Each Service Agent that meets the criteria in the SrvRqst will respond with the URL of the Service. The Salutation Manager will then use the SLP multicast convergence algorithm to collect all responses.

In addition to unicasting a SrvRqst to a DA or multicasting it to SAs, a Salutation Manager will still need to broadcast if it wants to discover FUs that are registered with Salutation Managers under SLA V2.0.

A Salutation Manager will remove all the duplicates in responses from a DA or SAs and broadcast.

4.6.2.Specification

The following sections specify how the Salutation Manager will exactly use SLP, and changes to the Salutation Manager API.

4.6.2.1.Initialization

The following parameters associated with SLP will be made known to the Salutation Manager in a manner outside of the scope of this document. (For example, the Salutation Manager may be implemented to provide a user interface for an administrator to setup these parameters manually.)

- language code

  The default value for the language code should be "en". (It does not preclude, for example, Japanese-only implementation.)

- character encoding

  The default character encoding should be UTF-8.

- scope (indicates a location, administrative grouping, proximity in a network topology or some other category for administration purpose)

  The default scope name should be DEFAULT, as specified by the SLP specification.

- lifetime

The default lifetime for a service shall be implementation specific. For example, a PDA should have a shorter lifetime than a device such as an large office MFP.

4.6.2.2.Service Registration

When an FU is registered, a SA sends one or more SrvReg messages to each DA it has discovered. A SrvReg message will include a service type parameter which consists of an abstract type and a concrete type. The SA will always send a SrvReg message of which abstract type is "salutation" and concrete type is "<fu-type>" where <fu-type> is, for example, "print" if the FU is a [Print] FU. If the FU is of a type that has a corresponding abstract type defined by IETF, as in the case of a [Print] FU which corresponds to the IETF-defined abstract type "printer", the SA may optionally send a SrvReg message of which abstract type is the one defined by IETF and concrete type is the IETF-defined protocol name for each protocol the FU supports. For example, when a [Print] FU that supports LPR, IPP, and the Salutation personality protocol is registered, the SA may send SrvReg messages with the following service types:

service:printer:lpr
service:printer:ipp
service:printer:spp

in addition to a SrvReg message with the following service type:

service:salutation:print

A SrvReg message will include also the list of attributes. The list will include an attribute that will specify the type of FU (the integer value of the FU ID -- it should not be confused with the FU Handle -- followed by all the capability attributes of the FU.

The SrvReg message will include the following parameters:

- A service: URL parameter in the following form:

service: URL = <srvtype>://<host>/<fu-name>;slmid=<slm-id>;fuh=<fu-handle>

<srvtype> = service:salutation:<fu-type>

-- this form is always used

<srvtype> = service:<abstract-type>[.<naming-auth>]:<protocol>

-- this form may optionally be used for each protocol supported by the FU if the FU type and the protocol have corresponding abstract type and protocol name defined by IETF or a recognized naming authority

<fu-type> = the name of the type of Functional Unit in all lowercase characters, e.g., "print" if a

[Print] FU is being registered. Space characters will be changed to "-", e.g., "doc-storage" if a [DOC Storage] FU is being registered. It will be "unknown" if the SA does not recognize the type of FU. (This case may happen if a new FU type is defined by a future version of the Salutation architecture, and such FU registers with a V2.1 Salutation Manager.)

<protocol> = the name of a protocol (e.g., lpr)

<naming-auth> = the name of a naming authority

<host> = a hostname (which should be used if possible) or dotted decimal notation for a hostname, followed by an optional ":" and port number.

<fu-name> = a string containing the name of the FU being registered (reserved characters, if any, will be escaped)

<slm-id> = an opaque value containing the SLM-ID of the Salutation Manager (escaped characters will be used)

<fu-handle> = an integer value containing the FU handle of the FU being registered

- An attribute list which is generated according to the following rule

attr-list = attribute / attribute "," attr-list

attribute = "(" attr-tag "=" attr-val-list ")"

attr-val-list = attr-val / attr-val "," attr-val-list

; if Salutation attribute type is SET OF ..., multiple attr-vals may be listed

attr-tag = ; decimal number of the Salutation attribute ID, or "fu-type" for the attribute to indicate the type of FU (the fu-type attribute will be the first attribute in the attribute list)

attr-val = intval / strval / boolval / opaque

; intval if Salutation attribute type is INTEGER or ENUMERATED

; strval if Salutation attribute type is DisplayString

; boolval if Salutation attribute type is BOOLEAN

; opaque if Salutation attribute type is OCTET STRING

intval = [-]1*DIGIT

strval = 1*safe-val

boolval = "true" / "false"

opaque = "\FF" 1*escape-val

safe-val = ; Any character except reserved.

reserved = "(" / ")" / "," / "\" / "!" / "<" / "=" / ">" / "~" / CTL

escape-val = "\" HEXDIGIT HEXDIGIT

The slm-id and fu handle attributes in the service: URL parameter of the SrvReg message are actually redundant because the hostname and FU name are enough to uniquely identify an FU. However, they are included to save network traffic because UA will not need to send a message to get SLM-ID and FU Handle to proceed to service request after search operation.

4.6.2.3.Service Deregistration

As part of the FU deregistration process, a Service Agent sends a SrvDeReg message to all the Directory Agents that the SA has previously registered with. The message will includes the same service: URL parameter as used in registering the FU. If an SA did not discover any Directory Agents, the SrvDeReg is not necessary on shutdown.

4.6.2.4.Search Operation

As part of the "Search Capability" process initiated by a Salutation client application, a User Agent sends SrvRqst message(s) to any one of the Directory Agents it has discovered. A separate SrvRqst message will be sent for each FUDR included in the search capability request. (Typically, only one FUDR is included in a single request.) Each SrvRqst message will include the following parameters:

- service type parameter in the following form:

  <service-type> = service:salutation[:<fu-type>]

  <fu-type> = the name of the type of Functional Unit in all lowercase characters, e.g., "print" if a [Print] FU is being searched. Space characters will be changed to "-", e.g., "doc-storage" if a [DOC Storage] FU is being searched.

  If the wildcard FU type is specified in the search request, the service type parameter will not include the ":<fu-type>" part.

- Optional <predicate> parameter in the form of LDAPv3 search filter [3].

  All the attributes in the FUDR will be used to form an "and" filter. The id and value of each attribute in the FUDR will be translated as described in the "Service Registration" section above. The compare function will be translated as follows:

Salutation Compare Function	**LDAPv3 Search Filter**
intEqualTo, boolEqualTo, strEqualTo	attr=value
intNotEqualTo, boolNotEqualTo, strNotEqualTo	!(attr=value)
intLessThanOrEqualTo	attr<=value
intLessThan	&(attr<=value)(!(attr=value))
intGreaterThanOrEqualTo	attr>=value

intGreaterThan	&(attr>=value)(!(attr=value))
strDoesContain	attr=*value*
setIntDoesContain, setStrDoesContain	&(attr=value1)(attr=value2)...
setIntIntersect, setStrIntersect	|(attr=value1)(attr=value2)...
others	cannot be mapped directly

Note: The Salutation compare functions that cannot be mapped to LDAPv3 search filters are currently not used as the default (i.e., most typically used) compare function by any attributes defined by the Salutation architecture V2.0. Use of these compare functions are discouraged, and the definition may be removed from a future version of the Salutation architecture.

Responses from the Directory Agent are returned by SrvRply messages. The Salutation Manager will build a list of the pair of slm-id and fu-handle from all the SrvRply messages, removing any duplicates, and return the list to the Salutation client application.

Note that the result of a directory-based search operation may be slightly different from that of directory-less search as follows:

- If any attribute tags specified in the predicate don't exist in the registered FU, it is considered as a match under Salutation V2.0, while it is considered as a mismatch under SLP. Therefore, when a new attribute for an FU is defined in a future version of the Salutation architecture, and if a Salutation client application searches for an FU specifying a value of this attribute as a condition, a directory-based search will never find FUs that are implemented to an old version of Salutation architecture.

5.Salutation Manager API Specification

This chapter describes an abstract definition of the SLM-API, the application programming interface provided by the Salutation Manager to Salutation applications.

Please refer to the programming specification document of each Salutation Manager implementation for the concrete definition, which varies somewhat depending on the underlying operating system and the programming language used.

5.1.Overview

The SLM-API consists of the following functions. The functions whose names begin with ***slm*** are provided by the Salutation Manager and called by Clients. The functions whose names begin with ***fn*** are exported to Clients and called by the Salutation Manager.

5.1.1.Service Registration

slmRegisterCapability()

slmUnregisterCapability()

5.1.2.Service Discovery

slmSearchCapability()

slmSearchCapability2()

slmQueryCapability()

5.1.3.Service Request

slmOpenService()

fnOpenService()

slmTransferData()

fnReceiveData()

slmCloseService()

fnCloseService()

5.1.4.Availability Check

slmStartAvailabilityCheck()

fnNotifyFUUnavailability()

slmCancelAvailabilityCheck()

5.1.5.Miscellaneous

slmGetVersion()

slmGetLocalSLMID()

slmGetSLMIDbyAddr()

slmGetSLMIDandFUHbyURL()

5.2.Salutation Manager API Description

Each function of the SLM-API is described in the abstract in this section.

When the Client calls the Salutation Manager or is called by the Salutation Manager through the SLM-API, the Salutation Manager of the SLM-API using the Client is called the **local Salutation Manager**. Any other Salutation Manager is called a **remote Salutation Manager**. Note particularly that the local Salutation Manager may actually be in a “remote” piece of Equipment if the Client is communicating with the local Salutation Manager through Remote Procedure Call.

5.2.1.Service Registration

- Register capability using **slmRegisterCapability()** is described in this section.

5.2.1.1.slmRegisterCapability(): Register Capability

The **slmRegisterCapability()** function is called by Services and Clients to register their Functional Units with the local Salutation Manager as Functional Unit Description Records.

The calling Client passes a Functional Unit Description Record, which describes its capability, to the Salutation Manager. The format of the Functional Unit Description Record structure is described in section 2.3.2.

The Salutation Manager returns a Functional Unit Handle which uniquely identifies the Functional Unit among all the Functional Units registered with the Salutation Manager.

When the Salutation Manager generates a new Functional Unit Handle value to return to the Client, it should generate a non-zero random value.

While a Service or Client has a Functional Unit registered with a local Salutation Manager, the Functional Unit’s capability may be included in the response to a *Query Capability* request received by the local Salutation Manager from another Client or Functional Unit, and the registering Service or Client may receive an *Open Service* request from another Client or Functional Unit at any time.

Input Parameters

Functional Unit Description Record

Describes the capability of the calling Client.

Callback Entry for Open Service Indication

The specified entry of the calling Client is called back by the Salutation Manager when an *Open Service* request for the Functional Unit is received.

Callback Entry for Close Service Indication

The specified entry of the calling Client is called back by the Salutation Manager when a *Close Service* request for the Functional Unit is received.

Callback Entry for Receive Data Indication

The specified entry of the calling Client is called back by the Salutation Manager when a *Transfer Data* request for the Functional Unit is received.

Preferred Functional Unit Handle

If the calling Client wants to be assigned any specific value as its Functional Unit Handle, the preferred value is specified in this parameter. Otherwise, zero (0) should be specified.

The specified value, if not zero, is returned by the Salutation Manager in the following output parameter unless it has already been assigned to another registered Functional Unit. Even if the specified value has already been taken, this call is not rejected but the Salutation Manager simply generates a new value and returns it.

When Clients are implemented in Equipment where persistent memory is available, it is recommended to utilize this parameter as follows:

- When the Client registers itself with a Salutation Manager the first time, it specifies zero in this parameter. When it registers with the Salutation Manager for the second time, it specifies the Functional Unit Handle value returned at the first registration in this parameter. This way the Functional Unit Handle value of each registering Client can be static across the re-initializations of the Equipment.

Output Parameters

Functional Unit Handle

The Salutation Manager generates a unique Functional Unit Handle value and returns it to the calling Client. If the function fails, zero (0) is returned.

5.2.1.2.slmUnregisterCapability(): Unregister Capability

The Client, which has registered itself with the local Salutation Manager by calling the **slmRegisterCapability()** function, calls this function to unregister itself from the local Salutation Manager.

Input Parameters

Functional Unit Handle

> The calling Client specifies the Functional Unit Handle of the Functional Unit to be unregistered. The value given by the **slmRegisterCapability()** function must be used.

Output Parameters

None.

5.2.2.Service Discovery

Service discovery is defined in this section including search and query capabilities.

5.2.2.1.slmSearchCapability(): Search Capability

The Client calls this function to ask the local Salutation Manager to search for Salutation Managers having a registered Functional Unit with a specific capability. The local Salutation Manager returns the list of SLM-IDs to the Client. The Salutation Manager of the SLM-ID included in the list has a Functional Unit that can provide the Service requested by the Client.

Input Parameters

SLM-ID

> This parameter shall be NULL, indicating the local Salutation Manager, under this version of the architecture.
>
> *It is intended to allow the Client to direct the search capability request to any specific Salutation Manager in a future version of the architecture.*

Service Description Record

> Describes the interested/required capabilities. A Service Description Record that contains a Functional Unit Description Record of All Call Functional Unit ID, with no Attribute Records, may be specified to get the list of all the SLM-IDs of Salutation Managers known to the local Salutation Manager.

Output Parameters

List of SLM-IDs

> An array of SLM-IDs. Each SLM-ID indicates a Salutation Manager with which an Functional Unit with the requested capability is registered. The list may include the local Salutation Manager itself which is indicated by either NULL SLM-ID or its explicit SLM-ID value. The list is empty if the requested Service is not found.

5.2.2.2.slmSearchCapability2(): Search Capability 2

The Client calls this function to ask the local Salutation Manager to search for Salutation Managers having a registered Functional Unit with a specific capability. The local Salutation Manager returns the list of pair of SLM-ID and FU handle to the Client. The Functional Unit identified by the returned FU Handle which is registered with the Salutation Manager of the associated returned SLM-ID can provide the Service requested by the Client It will enable Salutation client applications to proceed to requesting the service of a found FU without first submitting a Query Capability request to find out the FU handle.

Input Parameters

SLM-ID

> This parameter shall be NULL, indicating the local Salutation Manager, under this version of the architecture.
>
> *It is intended to allow the Client to direct the search capability request to any specific Salutation Manager in a future version of the architecture.*

Service Description Record

> Describes the interested/required capabilities. A Service Description Record that contains a Functional Unit Description Record of All Call Functional Unit ID, with no Attribute Records, may be specified to get the list of all the SLM-IDs of Salutation Managers known to the local Salutation Manager.

Output Parameters

List of FU location (pair of SLM-ID and FU handle)

> An array of FU locations each of which consists of the SLM-ID and FU handle. Each FU location indicates a Functional Unit that can provide the requested capability and a Salutation Manager the Functional Unit is registered with. The list may include the local Salutation Manager itself which is indicated by either NULL SLM-ID or its explicit SLM-ID value, and FU handle. The list is empty if the requested Service is not found.

5.2.2.3.slmQueryCapability(): Query Capability

The Client calls this function to discover registered Functional Units and their capabilities at a specific Salutation Manager. The Client may specify either the local Salutation Manager or a remote Salutation Manager.

The specified Salutation Manager automatically responds to this query without involving the registered Functional Units.

Input Parameters

SLM-ID

> Specifies the target Salutation Manager. If NULL is specified, the target Salutation Manager is the local Salutation Manager.

Service Description Record

Describes the interested/required capabilities.

Output Parameters

Service Description Record

Describes the available capabilities.

5.2.3. Service Request

The Service request functionality of the Salutation Manager is described in this section including open Service, open Service indication, transfer data, receive data indication, close Service and close Service indication.

5.2.3.1. slmOpenService(): Open Service

The Client or Functional Unit calls this function to ask the local Salutation Manager to initiate a Service session with a specific Functional Unit under a specific Personality Protocol. The target Functional Unit may be registered at either the local Salutation Manager or a remote Salutation Manager.

Input Parameters

Target SLM-ID

Identifies the Salutation Manager with which the target Functional Unit is registered. If NULL is specified, the target Salutation Manager is the local Salutation Manager.

Target Functional Unit Handle

Identifies the target Functional Unit with which the calling Client wants to establish a Service session.

Personality Protocol ID

Identifies the Personality Protocol to be used in the opening Service session.

Requester Functional Unit Handle

If the calling Client has registered itself with the local Salutation Manager, its own Functional Unit Handle is specified. Otherwise, zero (0) shall be specified.

Credential

The Authentication Flavor field of this parameter should be set to NULL in the following cases:

- The target Functional Unit's Authentication Flavors attribute indicates "No Authentication".
- The target Functional Unit's Authentication Flavors attribute indicates "Optional

Authentication", but the calling Client is not associated with a specific user.

Otherwise, the calling Client must specify "User ID and Password" in the Authentication Flavor field of this parameter and the user's User ID in the rest of the Credential parameter.

Refer to "The Client calls slmGetSLMIDandFUHbyURL() to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

User Identification and Authentication" in section 2.9 for details on this parameter.

Verifier

The Authentication Flavor field of this parameter should be set to NULL in the following cases:

- The target Functional Unit's Authentication Flavors attribute indicates "No Authentication".
- The target Functional Unit's Authentication Flavors attribute indicates "Optional Authentication", but the calling Client is not associated with a specific user.

Otherwise, the calling Client must specify "User ID and Password" in the Authentication Flavor field of this parameter and the user's Password in the rest of the Verifier parameter.

Refer to "The Client calls slmGetSLMIDandFUHbyURL() to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

User Identification and Authentication" in section 2.9 for the detail of this parameter.

Callback Entry for Close Service Indication

The specified entry of the calling Client is called back by the Salutation Manager when a *Close Service* request for the opening Service session is received.

Callback Entry for Receive Data Indication

The specified entry of the calling Client is called back by the Salutation Manager when Client data is received in the opening Service session.

Output Parameters

Return Code

Indicates whether or not the *Open Service* request is accepted by the target Functional Unit and the reason if it is not accepted.

Client Service Handle

The local Salutation Manager generates a non-zero unique Client Service Handle value and returns it to the calling Client. This Client Service Handle will be used in subsequent SLM-API calls to identify this particular Service session. If the function fails, zero (0) is returned.

5.2.3.2.fnOpenService(): Open Service Indication

When an *Open Service* request is received either from a local Client or a remote Salutation Manager, the Salutation Manager first checks if the parameters in the request are compatible with the registered capabilities. If the test passes, the Salutation Manager calls this callback entry to inform the Client of the *Open Service* request.

The called Client indicates whether or not it accepts the request and the reason if it does not.

Input Parameters

Server Service Handle

> The Salutation Manager generates a non-zero unique Server Service Handle value and gives it to the Client (Functional Unit). This Server Service Handle will be used in subsequent SLM-API calls to identify this particular Service session about to be opened.

Personality Protocol ID

> Indicates the Personality Protocol the requesting Client is assuming in the Service session to be opened.

Requester SLM-ID

> Indicates the requesting Client's local Salutation Manager. This value is set by its local Salutation Manager, not by the requesting Client. If NULL is specified, the requester Salutation Manager is the local Salutation Manager.

Requester Functional Unit Handle

> If the calling Client has registered itself with the local Salutation Manager, its own Functional Unit Handle is specified. Otherwise, zero (0) shall be specified.

Credential

> Identifies the user that is requesting to use the Service provided by the called Functional Unit.
>
> This parameter is a structure in which the first parameter is "Authentication Flavor" which contains either NULL or "User ID and Password". If no user is associated with the requester, or if the called Functional Unit does not require verification, NULL is specified. Otherwise, "User ID and Password" is specified in the "Authentication Flavor", and the rest of this Credential parameter contains the "User ID" of the requesting user.

Verifier

> Used by the called Functional Unit to authenticate the Credential.
>
> This parameter is a structure of which the first parameter is "Authentication Flavor", which is either NULL or "User ID and Password". If no user is associated with the requester, or if the called Functional Unit does not require verification, NULL is specified. Otherwise, "User ID and

Password” is specified in the “Authentication Flavor”, and the rest of this Verifier parameter contains the “Password” associated with the “User ID”.

Refer to “The Client calls slmGetSLMIDandFUHbyURL() to get the pair of SLM-ID and FU handle of the Salutation Manager translated from the IP host name and FU name.

User Identification and Authentication” in section 2.9 for the detail of the last two parameters.

Output Parameters

Return Code

Indicates whether or not the *Open Service* request is accepted by the called Functional Unit, and the reason, if it is not accepted.

5.2.3.3.slmTransferData(): Transfer Data

The Client calls this function to send data to the Client at the other end of a specific Service session.

Input Parameters

Client/Server Service Handle

Identifies the Service session sending the data. It is called Client Service Handle if the Client that has made the *Open Servi*