Salutation-Lite Code Programmers Guide

Final Beta Release (1.0.0)

November 8, 1999

Copyright

Limitation of Liability

The XtraWorX nor the Consortium shall not make any warranty or representation, neither express nor implied, with respect to this document, 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 and XtraWroX, LLC make 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.

Table of Contents

References

- Salutation Architecture Specification Part 1. (http://www.salutation.org/ordrspec.htm)
- Salutation Architecture Specification Part 2. (http://www.salutation.org/ordrspec.htm)
- Salutation Architecture Specification Part 4 (Proposed),
- Salutation Architecture Change Request: OpSys Functional Unit, November 9, 1999
- Salutation Architecture Change Request: Display Functional Unit, April 5, 1999
- Salutation Architecture Change Request: Determine the Value of a Subset of the Attributes of a Registered Functional unit, April 5, 1999
- Salutation Architecture Change Request: Response type Specification for Query Capabilities Call/Reply, November 9, 1999
- Salutation Service Discovery Protocol on IrDA: Application Note

Development Environment

The Salutation Lite Sample Code was developed using the following compilers:

Salutation-Lite Client	Microsoft Visual C++, Version 6
Salutation-Lite server on Windows CE	Microsoft Visual C++, Version 6, plus Windows CE Extensions
Salutation-Lite Server on Java	Java JVM 2.1

Salutation-Lite

Salutation-Lite is a footprint-reduced implementation of the Salutation Architecture. It is intended for the information appliance market, where storage space is limited, communication bandwidth may be low and power consumption is at a premium. Targets appliances are hand-held and palm-sized computers, portable phones, pagers, home appliances and local information servers.

Salutation-Lite may be used as a reference model for beginning a Salutation implementation. Salutation-Lite provides a reference implementation for the Salutation service discovery function. Salutation's availability check and session management may be added to this simple design.

Salutation-Lite demonstrates two new Functional Units; [Display] and [Operating Environment].

Salutation-Lite demonstrates two functions designed for limited bandwidth environments; Reply Flavor and Don'tCare Compare algorithm.

Salutation-Lite is implemented on IrDA protocol. The implementation has been easily ported to other protocol stacks.

Reducing the footprint

The Salutation Architecture supports three general functions.

- **Service Discovery**: The ability to advertise capabilities and find services which provide needed capabilities
- **Availability Check**: The ability to determine if a desired service is available
- **Session Management**: The ability to open, control, and maintain a session with a desired services

Salutation-Lite implements the service discovery function, enabling a device, application, or service to define its capabilities in a standard way, and allowing other devices, applications and services to interrogate these local services definitions. The Salutation functions of availability check and session management, which may not be necessary in all applications, are not supported in the Salutation-Lite implementation. These functions may be easily added to the Salutation-Lite design as a user option.

The Salutation Architecture specifies Remote Procedure Calls (RPC) for sending Salutation commands and tracking responses. Salutation-Lite does not use RPC. Instead, a BER encoded wrapper is placed around the Salutation generated content to specify the type of Salutation command and match responses with requests.

Salutation Architecture specifies 11 Application Program Interfaces (API) to access and control the functions of the Salutation Manager. Salutation-Lite supports only 4 of these APIs; slmRegisterCapability, slmUnregisterCapability, slmSearchCapability and slmQueryCapability[1]. Since Salutation-Lite does not support availability check and session management, supporting APIs are not needed.

To further reduce the Salutation-Lite implementation, it has been divided into a client Salutation-Lite Manager and a server Salutation-Lite Manager.

Where possible, the Salutation-Lite implementation uses system resources to perform necessary functions. For example, the WindowsCE version of Salutation-Lite uses Windows Socket interface to access the IrDA protocol.

Error checking is kept to a minimum. For example, it is assumed that the SDR and FUDR BER encodings have been generated correctly, and that attribute values are within architected limits.

[1] A prototype for the slmOpenService API has been included in this release.

Function Description

Protocol

The Salutation-Lite protocol is designed and implemented on the Infrared Data Association (IrDA) stack. To ease portability to other protocols, the high-level interfaces to the IrDA stack are used. For example, Windows Sockets is used to access IrDA stack in Windows implementations[2]. The flow diagram for Salutation-Lite on IrDA is shown in Appendix A.

Transport

Salutation-Lite is built on the IrDA protocol found in many of today's mobile information appliances. In the WindowsCE implementation, access to the IrDA protocol is through Windows Sockets. Other protocols, such as TCP/IP and Bluetooth are readily accessible with minimal changes to the Salutation-Lite code.

Exchange SLM ID

The Transport Manager handles the exchange of SLM IDs. The Client's Transport Manager generates an ExchangeSLM-ID protocol command when another IrDA device is encountered. Responses are correlated with the IrDA address of the responding device. The Salutation-Lite Client will not perform capability checking on local services.

Client

The Salutation-Lite Client exposes the slmSearchCabability and slmQueryCapability APIs. An application on the client can find Salutation servers (Lite or regular) using these APIs, and determine the capabilities of the functions on these Salutation servers.

The Salutation-Lite Client generates the Query Capability protocol command.

The Salutation-Lite Client does not process the SDR content passed in the APIs or returned from the transmission protocol. No error checking is performed.

Server

The Salutation-Lite Server processes QueryCapability protocol commands. It assumes that the information received is in the architected format and tolerances. No error checking is performed on the SDR content of the QueryCapability command.

Local capabilities are registered in the server via a slmRegisterCapability API call. While the Salutation Architecture requires that Functional Unit Description Record (FUDR) passed in the API be BER encoded, Salutation-Lite passes a structure defining the Functional Unit capabilities. The structure is flexible and may contain the definition of any Functional Unit (FU), including the newly defined [Display] and [OperatingEnvironment] FUs. Header files are provided which provide reference definitions for these latter two FUs. The user may dynamically alter the content of the structure in the server-side application through standard programming techniques, and then pass the structure through the slmRegisterCapability API to the lite version of the Salutation Manager.

The Salutation-Lite Server parses the received service Description Record (SDR) content received from the Salutation client via theQueryCapability protocol command, and compares it against the content of the local capability registry structure. The parser supports the newly defined QueryCapabilities Response Type and the Don'tCare Compare ID functions.

[2] A Wintel version has been created on TCP/IP, for example. This version will be offered in open source at a later date.

Design Specification

Figure 1 depicts the high-level design of Salutation-Lite.

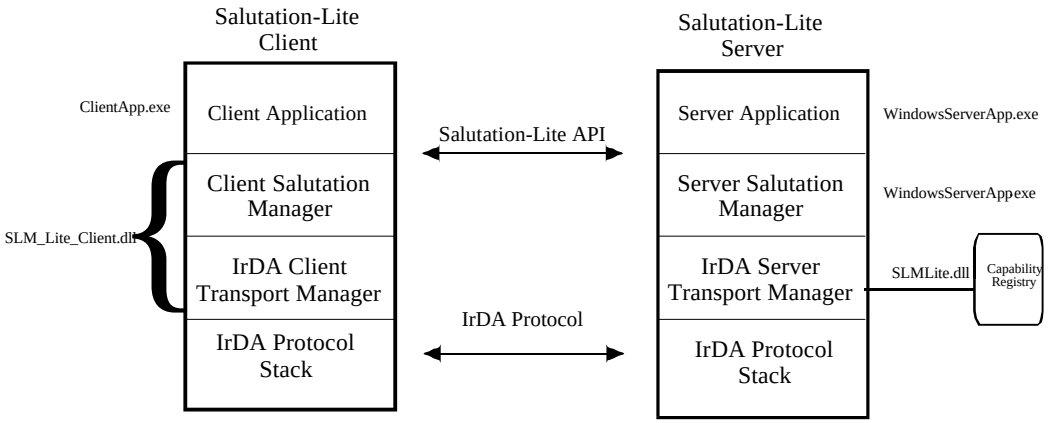


Figure1: High-Level design

Salutation-Lite Server

The Salutation-Lite Server is designed for the mobile information appliances. It provides a means for the Server to describe its capabilities. It processes requests for capability information from Salutation Clients. The Client need not be a Lite version of Salutation.

The Salutation-Lite Server consists of a Server Salutation Manager[3] part and an IrDA Server Transport Manager[4] part. The IrDA Server Transport Manager interfaces the Salutation Manager with the IrDAProtocol Stack using IR Sockets. The IrDA Server Transport Manager receives Salutation protocol requests from the client and passes them to the Server Salutation Manager for processing. The IrDA Server Transport Manager passes responses from the Server Salutation Manager to the IrDA Stack.

The unique Functional Unit capabilities of the Server are stored in a registry maintained by the IrDA Server Transport Manager. The registry is a program structure populated by the Server Application and passed to the Salutation-Lite Transport Manager via the slmRegisterCapability API. The Server Salutation Manager uses the Salutation Architecture defined algorithms to compare the capability request received from a Salutation Client with the capability information stored in the registry structure. The Server Salutation Manager builds a response to the capability request based on the results of the compare, and the value of the Reply Flavor attribute.

The Salutation-Lite Server runs in the background. With the exception of Message Boxes reporting IrDA initialization errors, the Salutation-Lite Server does not present any user interface.

The Server Application[5] is provided for testing purposes. It presents a simple Window for registering and unregistering capabilities wit the Server Salutation Manager. Although not part of the Salutation-Lite Client, this application is provided to demonstrate the use of the APIs. The application has a pull-down menu providing the ability to register and unregister Display and Operating Environment Functional Units.

Salutation-Lite Client

[3] Defined by the WindowsServer.cpp source code and WindowsServer.dll executable.
[4] Defined by the SLMLite.cpp source code and SLMLite.dll executable.
[5] Defined by the WindowsServerApp.cpp source code and WindowsServerApp.exe executable.

The Salutation-Lite Client is designed to support applications wishing to discover functionality on the server. The Salutation-Lite Client provides a means for an application to locale mobile information appliance of a specific type and determines the detailed capabilities of that appliance. The appliance must be Salutation enabled, but need not be a Lite version.

The Salutation-Lite Client consists of a Client Salutation Manager part and an IrDA Client Transport Manager part[6]. The IrDA Client Transport Manager interfaces the Client Salutation Manager with the IrDA Stack. The IrDA Client Transport Manager passes QueryCapability Salutation protocol commands to the IrDA Stack. The IrDA Client Transport Manager passes responses received from the server over the IrDA protocol to the Client Salutation Manager.

The Client Salutation Manager assumes a peer-to-peer connection with the server. Therefore, there is no attempt to correlate the server's SLM ID with that requested by the Client.

The Client's Salutation Manager exposes two APIs. slmSearchCapability allows an application to locate appliances having specific capabilities. slmQueryCapability allows an application to interrogate the capabilities of a specific appliance.

The Client Application[7] is provided for testing purposes. It presents a simple Client Window for reporting the results of a Query Capabilities command. Although not part of the Salutation-Lite Client, this application is provided to demonstrate the use of the APIs. The Client Application is designed to locate an appliance, interrogate its capabilities and display the results of the interrogation. The application has a pull-down menu providing the ability to register and search for and query the server for capabilities[8].

Design Details

Capability Registry Structure

The Attribute.h file defines and initializes the structure used to register the capabilities of the Salutation-Lite Server. The structure is defined as follows:

```
struct AttributeHeader
{
        long Name;
        int Compare;
        int Length;
        int Used;
};
union value
        {
                char strValue[64];
                int intValue[64];

        };
struct Attribute
{
      AttributeHeader Header;
      value Value;
};
```

[6] Both the Clinet Salutaiton Mangaer and the IrDA Client Transport Manager are defined in the SLM_Lite_Client.cpp source and SLM_Lite_Client.dll executable.
[7] Defined by the ClientApp.cpp source code and ClientApp.exe executable.
[8] A *prototype* for opening a service session with one of the fund capabilities is also provided.

The registry is an array of Attribute structures. It may be depicted as in Table 1. Each row represents one instance of the structure.

	Name	Compare	Length	Used	Value
FU 1					
Attribute 1..					
Attribute 2..					
Attribute..					
FU 2					
Attribute 1..					
Attribute 2..					
Attribute..					

Table 1: Capability Registry Structure

The first row contains information about the Functional Unit.

- The Name field contains the Architected ID of the FU.
- The Length filed contains the number of Attribute fields (rows) to follow.
- The Value field contains the Functional Unit Handle
- The other fields are not used.

Subsequent rows, equal to the number contained in the FU row's Length field, define attributes of the FU.

- The Name field contains the Architected ID of the Attribute.
- The Compare field contains the Architected ID for the compare function to be performed on the attribute.
- The Length filed contains the number of bytes in the Value field of this row.
- The Used field is used by the compare process and is initially set to 0.
- The Value field contains the value of the attribute.

Multiple FUs may be defined in one registry by continuing the definition process in the array. The total number of structures in the array is defined in an integer called "regFUCount".

It is this structure that is passed to the Server Salutation manager via the slmRegisterCapability API.

Pre-Configured Device Selection

Although the current open source implementation supports only the Wintel platform, the Attribute.h file also provides for automatic configuration for several existing handheld devices. Conditional logic is provided in this header file to initialize the Capability Registry Structure for several information appliances, including a 'generic' Windows98 platform. To select a pre-defined configuration, place *one* of the following lines of code at the top of the WindowsServer.cpp file.

```
#define _WIN32_WINDOWS  /* Windows98 configuration    */
#define _PHENOM         /* LGElectronics Phenom       */
#define _SHARP          /* Sharp Mobilon HC-4500      */
#define _TRIPAD         /* Sharp Mobilon Tripad       */
#define _EVEREX         /* Everex PK10                */
```

Server Compare Functions

The Salutation-Lite Server is responsible for comparing the capabilities requested by the client with the capabilities registered in the server. The comparison code parses the SDR received from the IrDA Communications Stack through the Transport Manager. The parser assumes that the SDR has been encoded correctly by the client and the transport layer corrects any transmission errors. Therefore, the parser does not perform any error checking.

The parser locates the First Functional Unit Description Record (FUDR) and determines its type (name). It then checks the Registry to determine if this FU type is registered. If it is not, this FUDR is skipped. If it is, the capability values described in the FUDR are compared with the capability values stored in the Registry.

In the later case, the parser determines the attribute type, its compare value and the attribute value. It then searches the Registry for a like attribute in the Same FU description. If found, the parser compares the parsed attribute value with the registered attribute value based on the compare algorithm specified. For details of this compare function, see Section 4.1.3 of the Part 1 Salutation Architecture Specification. Of special note in the Salutation-Lite implementation:

- The following comparison algorithms are supported
  - INTEQUALTO
  - STREQUALTO
  - BOOLEQUALTO
  - SETINTDOESCONTAIN
  - INTGREATERTHANOREQUALTO
- The parser recognizes the ReplyFlavor attribute (70). If present in the SDR received from the Client, the response SDR is constructed according to the attribute value.
  - 1 = MAXIMUM -- all registered attributes are reported in the reply, even though they were not requested in the SDR sent from the Client
  - 2 = NOMINAL -- only attributes requested in the SDR sent by the Client are reported in the reply
  - 3 = MINIMUM -- No attributes are reported in the reply.

  The default is MAXIMUM.
- The parser recognizes the Don'tCare compare value. The parser ignores the attribute value of the requested attribute and reports the attribute value stored in the Registry.

Based on the results of the compare, the Salutation-Lite Manager builds a Response SDR and passes it through the Transport Manager to the IrDA Stack for return to the Client.

Salutation-Lite Client

The Salutation-Lite Client is an application to send slmSearcCapability and slmQueryCapability API calls to the Client Salutation Manager. For the sample code, the SDR is hard-coded in the WindowsClientApp.cpp file. The pull-down menu provides

Salutation-Lite Client APIs

Five Salutation APIs are supported in the Salutation-Lite Client.

- slmSearchCapability -- Determine the SLM-ID of other Salutation Enabled product
- slmQueryCapability -- Determine the capabilities of a Salutation Enabled product
- slmOpenService – Prototype of API which opens a service session with a discovered Functional Unit.
- slmTransmitData – Note supported in this release.
- slmCloseSession – Not supported in this release.

Salutation-Lite Server APIs

Three Salutation APIs are supported in the Salutation-Lite Server.

- slmRegisterCapability – register the attributes of a Functional Unit
- slmUnregisterCapability – remove the registration of a Functional Unit.
- slmTransmitData – not supported in this release.

Client Salutation Manager

The Client Application sends a QueryCapabilities command to a Server, searching for [Display] and [Operating Environment] FUs. The Client then displays the attribute values returned.

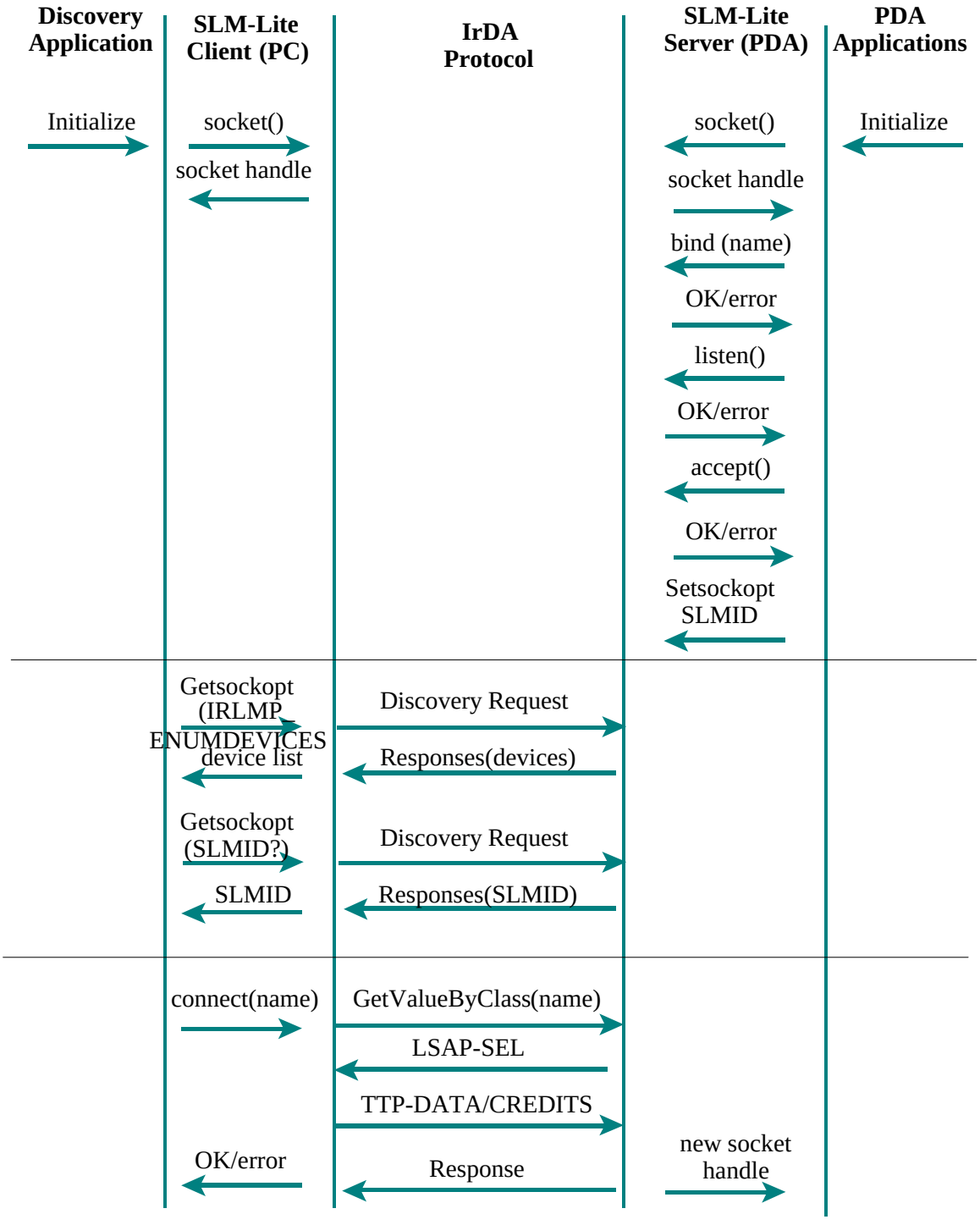
Other Design Considerations

- Server:

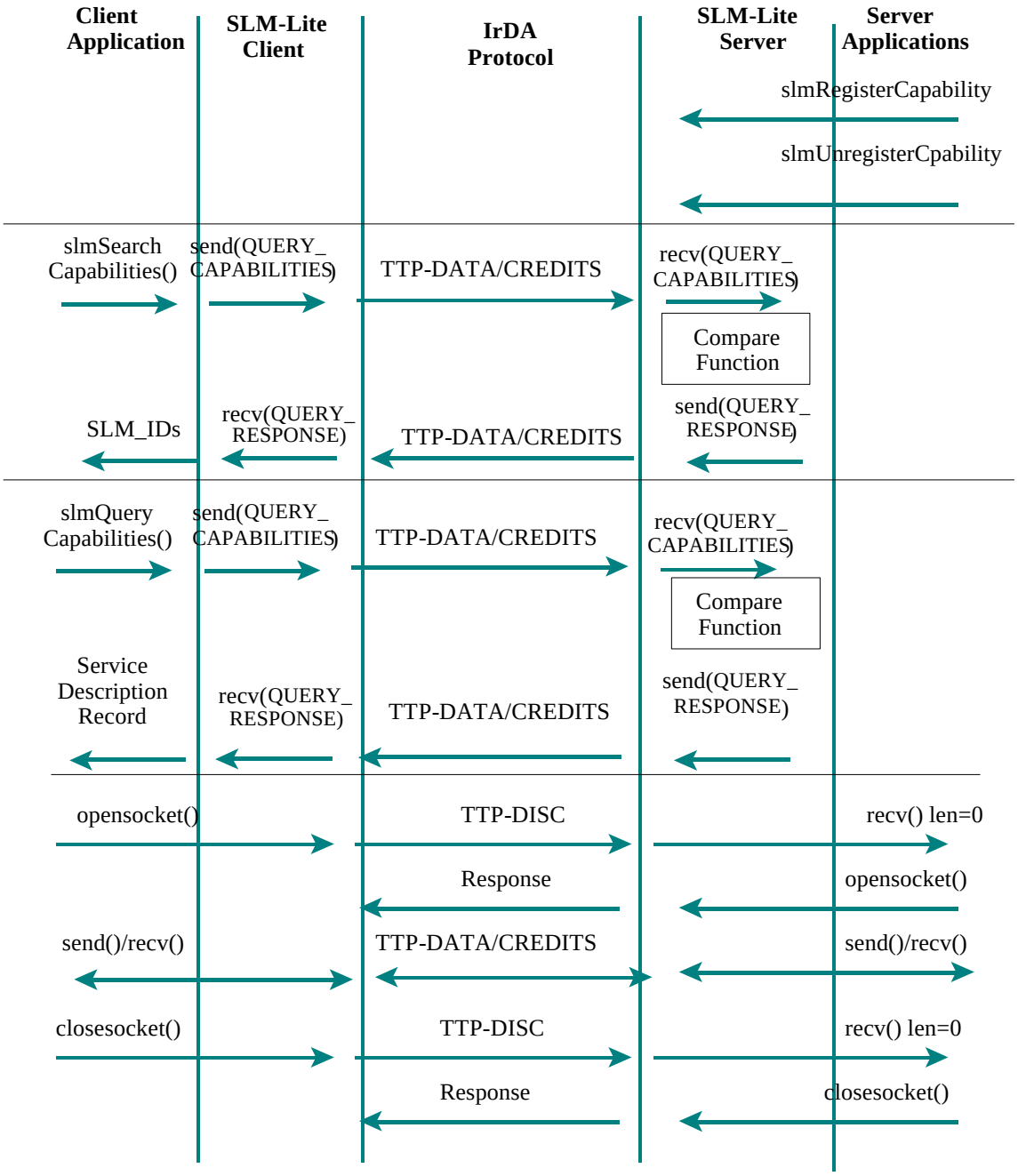
    - Only one Attribute of a given type may be specified within a registered FU
    - Registration storage limited to local FUs only
- Client:
    - Can only discover/communicate with one server at a time (assumes underlying transmission protocol limits communication to point-to-point)

Appendix A

Salutation Lite Protocol on IrDA



Part 1: Setup IrDA Link and locate Salutation enabled portable device



Part 2: SLM-Lite Search and Query Capability Calls, data transmission.

Appendix B

Salutation-Lite Function Support Table

The following table outlines the function support in the current releases

Platform	Windows	WindowsCE	Java
OpenSoruce Release	Release 1.0.0	Beta 2	Beta 2
slmSearchCapabilities API	Yes	No	No
slmQueryCapabilities API	Yes	No	No
slmOpenService API	Prototyped	No	No
slmTransmitData API	No	No	No
slmCloseService API	No	No	No
ExchangeSLMID Protocol Command	Yes	Yes	Yes
QueryCapabilities Protocol Command	Yes	Yes	Yes
OpenService Protocol Command	Prototyped	No	No
TransmitData Protocol Command	No	No	No
CloseService Protocol Command	No	No	No
Display FU	Yes	Yes	Yes
OpSys FU	Yes	Yes	Yes
Reply Flavor	Yes	Yes	Yes
Don'tCare Compare ID Value	Yes	Yes	Yes
BER Command Header	Yes	Yes	Yes