LEETECH

LeeTech AIM

Application Intelligent Middleware

LeeTech Software Inc.

Nobody provides better service than we do.

LeeTech AIM Reference Manual

Application Intelligent Middleware

2085 Hamilton Ave., Suite 200, San Jose, CA 95125

Released 01/03/01

The information contained in this document is subject to change without notice.

LEETECH SOFTWARE INC. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. LeeTech Software Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of LeeTech Software Inc.

Copyright © 1998-1999 LeeTech Software Inc.

Microsoft and Windows are registered trademarks of Microsoft Corporation.
ALLBASE/SQL, TurboIMAGE, IMAGE/SQL, MPE/iX, HP-UNIX and HP 3000 are all registered trademarks of Hewlett-Packard Company.
Oracle is a registered trademark of Oracle Corporation.
All other products or company names mentioned may be the trademarks of their respective owners.

TABLE OF CONTENTS

FIGURES

TABLES

1. Introduction to Client/Server

Client/Server has emerged as one of the dominant trends in the nineties and now accounts for a vast majority of the new IS projects.

Over the last decade, the information technology industry has seen phenomenal rates of technological and computing discipline changes. The advent of PCs and networks forever changed the ideas and methods of personal and corporate computing.

Since the beginning of corporate computing, several technological trends have surfaced. Everyone began using centralized computing on mainframes and mini-computers. When personal computers and local area networks gained prominence, the client-to-database server wave broke, empowering end-users with a wealth of decision-making information. In the latest wave are client-to-function (or application) servers.

The options and nuances of the latter-day technology can be confusing. Questions abound, such as, what exactly is client/server? What is driving it? What business needs brought it into existence?

A major concern is what will become of current legacy solutions. Those time-tested and reliable systems may now be bursting at the seams. Users may be clamoring for added functionality, or an easier-to-use system. How can that legacy investment in software, hardware and support—not to mention the business logic tied up in those systems driving an organization—be preserved?

A legacy investment can be preserved, by migrating to client/server solutions in an expedient and efficient manner. It is possible to migrate in a manner that preserves the baseline of the existing end-user services, and enhance it with the latest client/server technologies. LeeTech AIM (Application Intelligent Middleware) has a complete solution that transforms the legacy system to client/server.

The Fundamental Components of a Client/Server Environment

Like many rapidly evolving technologies, the terminology involved is over-worked, ill defined and largely misunderstood. In this manual, the discussion will include the underlying technologies and programming structures involved in building client/server applications, and explain the approach taken by LeeTech AIM to make high-performance enterprise client/server systems easier to develop, implement and operate.

"Client" and "server" are logical terms that do not necessarily refer to a PC, a mini-computer or a mainframe. In reality, any of these can be used as a client, as a server, or as a client and a server. Generally speaking, a client initiates requests and a server responds.

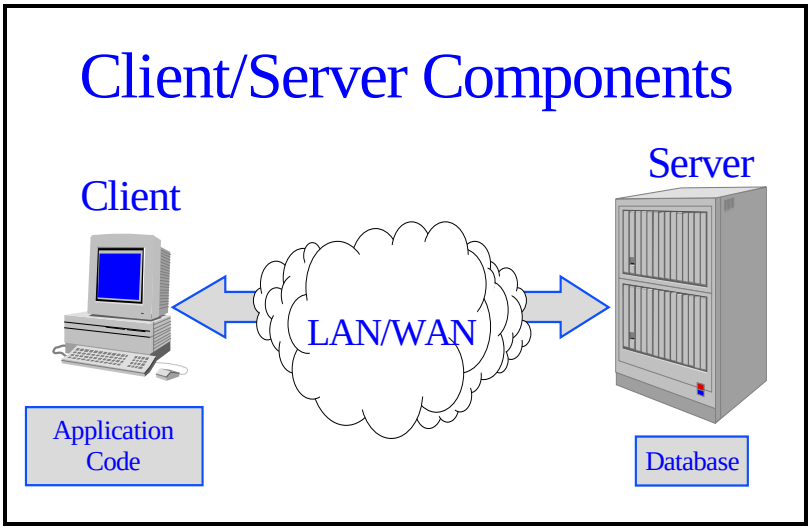

Figure 1-1. Client/Server Components

Client

A client is a program that initiates a server program for the purpose of requesting services from the program. The server program may be on the

same or a remote computer. A client program can simultaneously be a server to another program.

Server

A server is a program that does not execute unless it either is initiated by a client program, or is preloaded and running with a “stand-by” feature. In most cases, the server program does not send any data back to the client program unless it has been requested. A server program can simultaneously be a client to another program.

Middleware

Middleware consists of sets of software services parceled out among different hardware platforms. These services connect the application, the operating system, and network services on one or more system nodes to other platforms on the network. These bridge the legacy world to the PC world.

Middleware services sit just above the network layers providing a wealth of critical program-to-program, communications and data management services. Application developers can use a single set of application programming interfaces (APIs) to connect graphical development environments to legacy data and to applications hosted on mini-computers or mainframes.

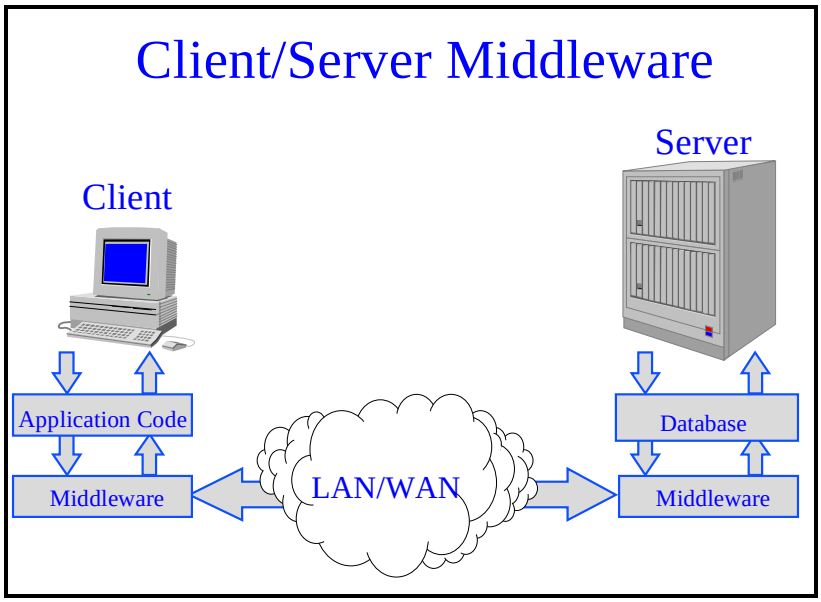

Figure 1-2. Client/Server Middleware

These services can even enable “peer” connectivity between disparate platforms without requiring specialized gateways. Mainframe or mini-computers, once viewed as “host-only,” can now act as a client by employing middleware services. These new “hosts-as-clients” can then request information from another network node (or even run applications on another node)—whether the other node is another mini-computer, mainframe or even a LAN server.

Security

Security is an important aspect of any computer system that allows users access from a network connection. With current technology, software and hardware devices exist that make it relatively easy to “sniff” a network. This process takes the TCP/IP packets off the network, copies them, and then puts them back on the network. The intended recipient computer does not know that the information has been copied. Of course, the implications are evident. Sensitive information is unknowingly made available to the wrong people. There are companies that do nothing but build protection

against this type of computer crime. In most cases, as soon as a firewall is put in place, someone figures out how to break through.

Our solution is safe and does not degrade the speed of the transport. Each packet is encrypted with a different algorithm and compressed. This means that if someone is "sniffing" the network traffic and picking up packets, the LeeTech AIM packets are unreadable at first glance. If someone is able to break the encryption method on a packet, the next packet will be encrypted differently and so forth.

Tiers

A tier is where a logical process is taking place and communicating with another tier via the Middleware. There must be at least 2 tiers for client/server. However, the number can be unlimited. Also, a multi-tier implementation could all reside on the same host system. A client process starts a server process that starts another server process, all on the same host computer. Various forms of client/server architecture follow this definition. Examples and explanations of these can be found farther along in this manual.

An example of typical client/server architecture would be like this:

A listener program, designed to listen to a pre-defined port on the server, is started.

When a client connects to the listener program, a server program is started to service the new connection.

When the server successfully starts, it takes the new connection from the listener.

From this point on, the client and server are communicating with each other to accomplish the task at hand. One "requests" either information or a sub-task from the other, then waits for the data or an indication the sub-task is completed.

Client/Server Architecture

Client/server architecture, providing seamless access to each other's resources, can be developed in several different configurations: two-tier, three-tier, multi-tier, etc. Each of the configurations discussed in this section maximizes the benefits of each individual computing platform within a given implementation. The goal of client/server architecture is to provide flexible, secure reliable solutions that are modular and can be implemented at an affordable cost. LeeTech/AIM provides various client/server models to meet different business requirements.

The Remote Data Access (RDA) Architecture

In this type of 2-tier client/server architecture, almost all development occurs on the client (tier 1). The programmer has little or no control over processing on the server side (tier 2). Remote Data Access (RDA) and Database Server are the most common of these 2-tier models. The entire application is developed at the client with RDA. This provides application platform and relational database independence. With Database Server, some shareable business logic is pushed down to the server (tier 2) by using standard techniques provided by the database vendors: such as triggers, constraints and stored procedures. This allows business rules to be more unified across the enterprise, but often at the expense of portability and relational database independence.

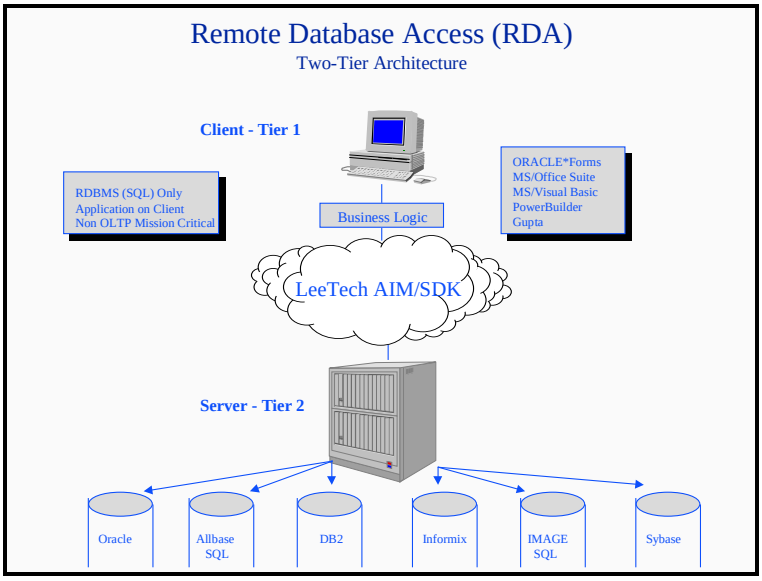

Figure 1-3. Remote Database Access (RDA)

With most implementations, the complexity of business logic that can be supported is limited. Any requirements for direct messaging or operating system interfaces generally cannot be supported.

More than Just SQL

Client/server has more potential than SQL-based applications and can be effectively applied in much broader classes of applications, including:

- Enterprise legacy applications that are difficult to re-write, or that have unique or unsupported system components.
- Distributed applications with multiple machines and/or complex transactions that do not fit the standard scenarios.
- Specialized applications that can benefit from client/server, but do more than just access data on the server side.
- Enterprise applications warranting special handling due to size and applications with special support and maintenance requirements.
- Mission critical databases that need access through the Internet/Intranet.

There are many ways to implement client/server applications. Decisions about data access, communications, front-ends, and locations of intelligence within the processing elements are highly variable and can be tailored to meet unique situations. Performance can be dramatically altered by changing the system structure or by using custom programs instead of generalized access languages.

The development task is complicated by the existence of legacy systems with years of existing data. The legacy systems are generally implemented on various interconnected hardware platforms, with varied applications, written in multiple programming languages, and use multiple database management systems. Any new applications must meet prevailing standards for user interfaces. They must also meet the user's expectations, generated by today's most popular PC software. Client/server is a bridge to link, consolidate, unify and standardize enterprise level data and the user interface. With a client/server framework, the systems can be built on time and within budget.

Two-Tier Server-to-Server Architecture

In a 2-tier Server-to-Server Architecture, a host system acts as a client to another host system. With this type of 2-tier architecture, a host system can start a process and communicate with another host system to either access data for processing or to pass commands to the other host system. This is called Server-to-Server (STS).

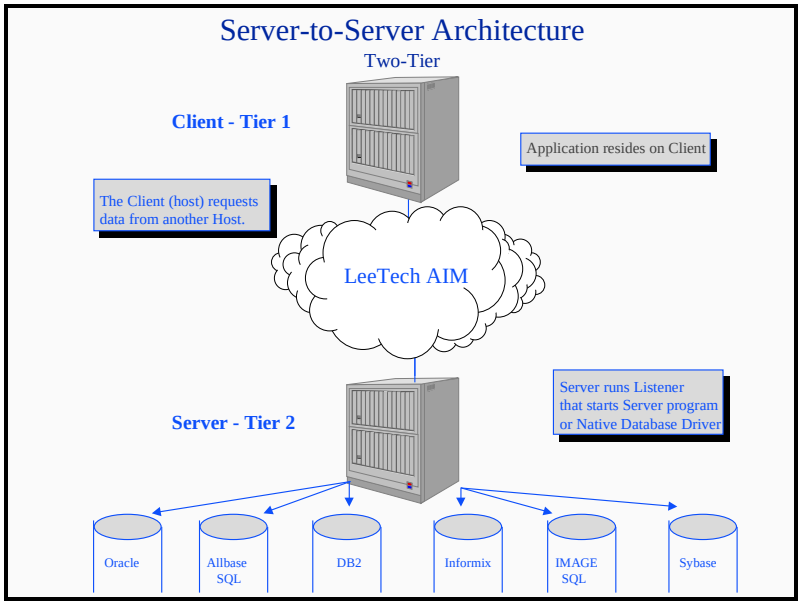

Figure 1-4. A Two-Tier Server-to-Server Architecture

***Note*:** In this example of a 2-tier system, the host client could be sending SQL statements to a host server native database driver. This is important for an application that needs to retrieve data that may even be a different type, from a database on a different server. A front-end program on the host client requests the records it needs directly from the host server database without having to do conversations and move the complete file from one host to the other.

Three-Tier Client/Server Architecture

In a 3-tier architecture, the application is broken into three parts:

- Tier 1 handles the user interface and data presentation.
- Tier 2 handles the main business logic.
- Tier 3 is the database repository.

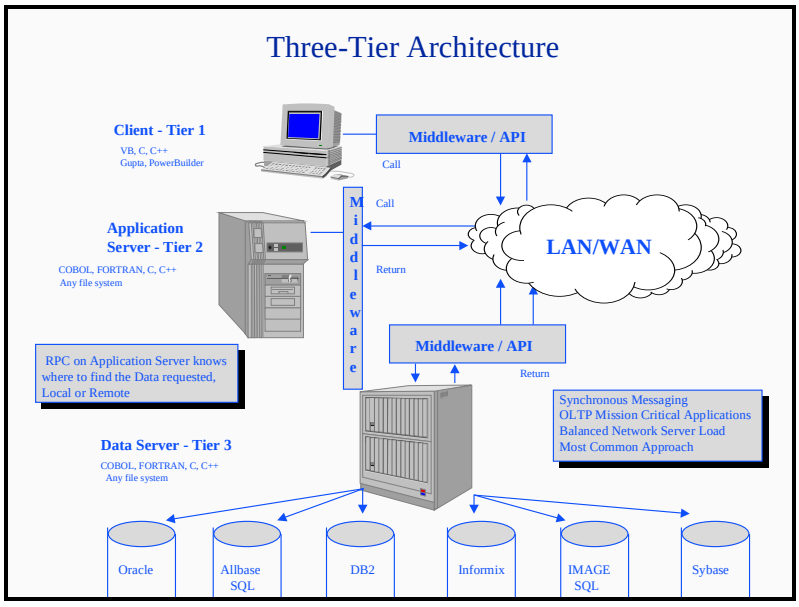

Figure 1.5 Three-Tier Architecture

Multi-Tier Client/Server Architecture

Multi-tier client/server architecture is built upon the 2-tier examples. Middleware is the key to client/server. The middleware vehicle can sit in between or among PCs, mainframes and minicomputers. All of them can be either a client, a server, or client and server.

The following diagram shows how LeeTech AIM is used to revamp a legacy system. Tier 3 is the computer where the legacy database resides and therefore would become the Database Engine. The application can be moved to an inexpensive NT computer that will become the application server, where most of the program logic is located. The user's PC will handle the GUI. This arrangement provides a tremendous amount of flexibility. The NT Application Server is free to handle other tasks, such as, being a client to a fax server or a printer server.

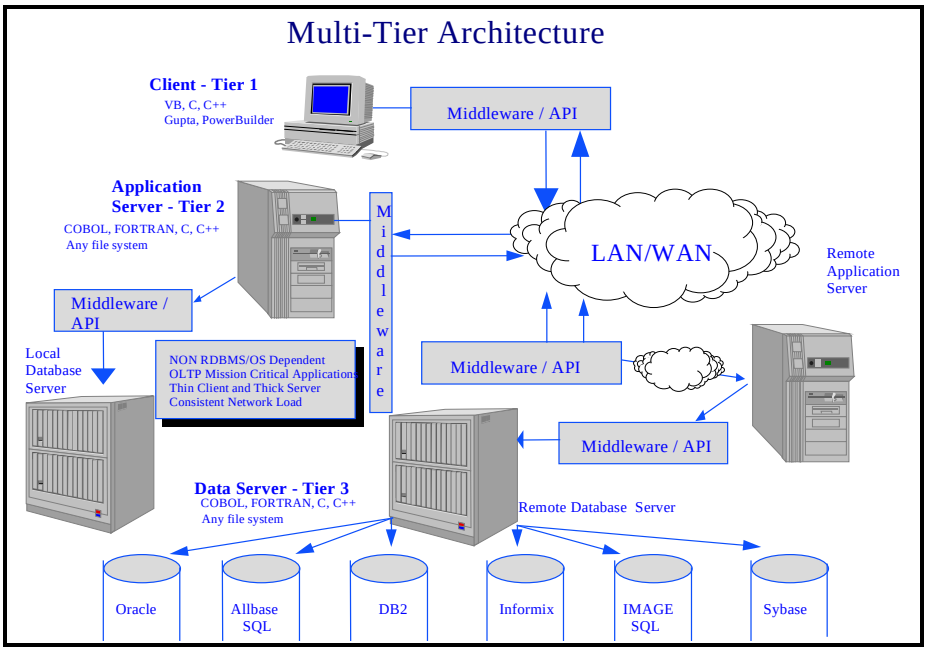

Figure 1-6. Multi-Tier Client/Server Architecture

Multi-Tier Server-to-Server Architecture

Most data processing centers today have a multitude of different hardware platforms and operating systems. The need to exchange data between these computers always exists. In real-time environments, it does not make sense to have an operator moving the data via tapes or upload/downloads. The ideal situation is for the system that needs the data to go and get it, when it needs it. The host client may need data from several computers at the same time; or it may need to control other computers in the network.

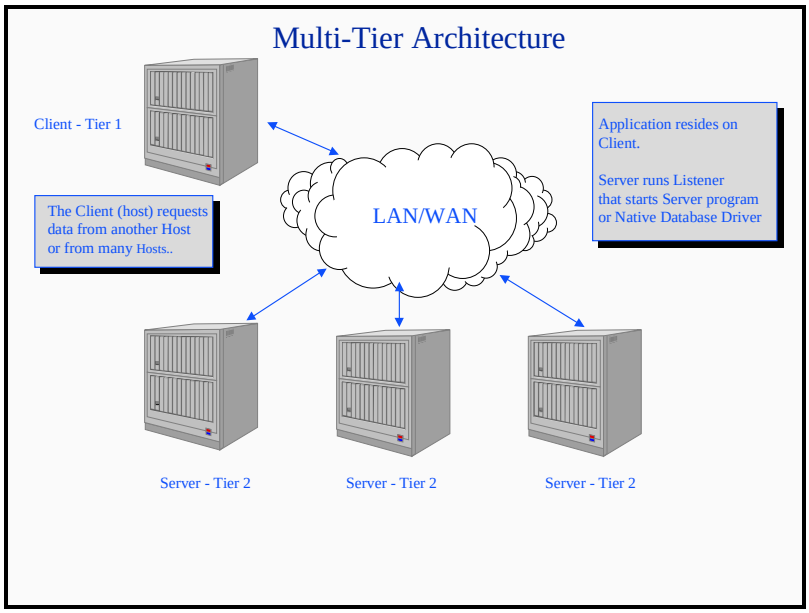

Figure 1-7. Multi-Tier Architecture

In some designs, the host server becomes a host client to another host server. This is similar to the 3-tier example above, except that the first client is a host rather than a PC. This makes efficient use of all of the host computers in the network. In this architecture, a true "distributed environment" can be designed and utilized for faster throughput of tasks.

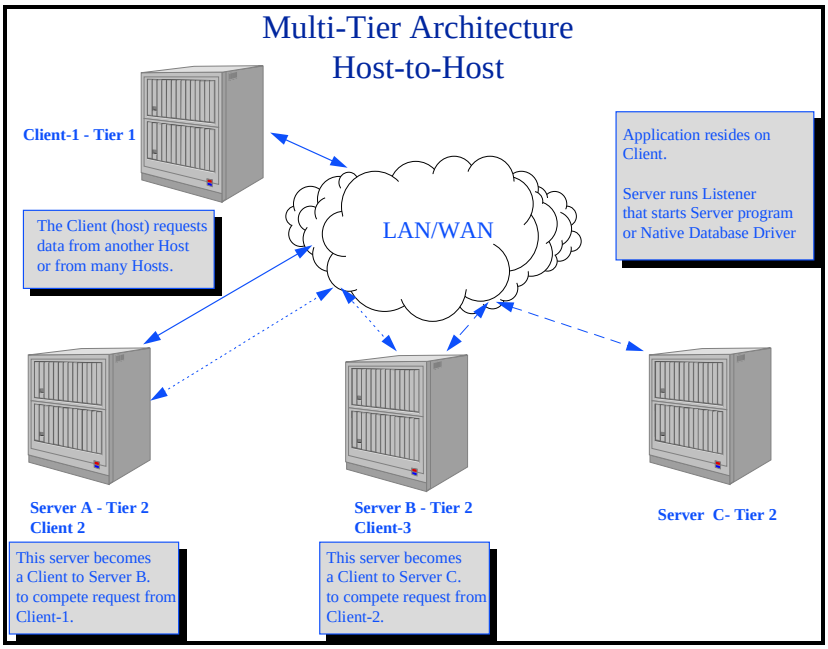

Figure 1-8. Daisy Chain Multi-Tier Architecture

Simplified Application Development

To simplify application development, the tools should treat a server program/process as a file. The client program/process should be able to open and close a server, or multiple server program(s)/process(es). Each message (request/reply or write/read) can be treated as a transaction. A server program can be constructed as a set of transaction blocks (object-oriented) which can be shared across platforms and database servers.

Internet/Intranet

The current trend is to make data available to users of the World Wide Web (WWW) or a private intra-company Web (Intranet). This is another example of multi-tier client/server. The PC is tier 1, the Web server is tier 2, and the database server is tier 3.

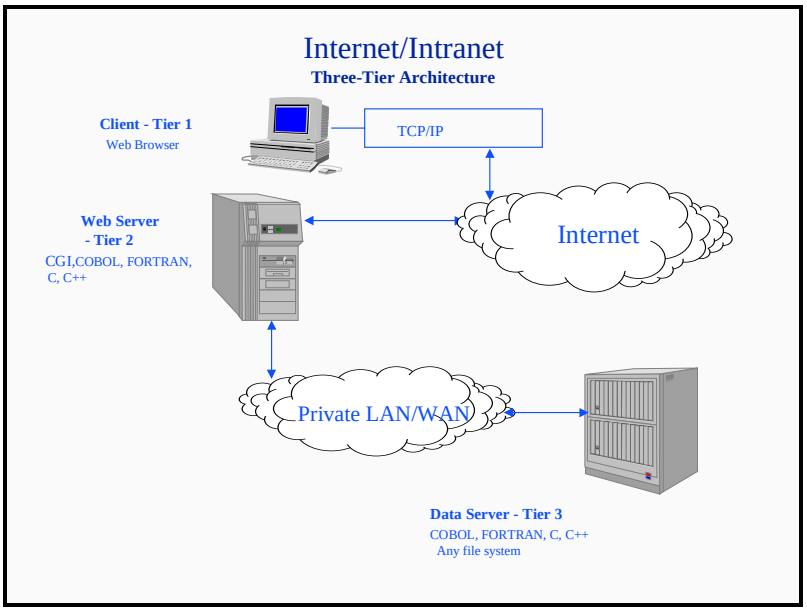

Figure 1-9. Internet/Intranet Three-Tier Architecture

There are numerous reasons for using this type of environment:

- The LAN is already in place in both of these scenarios.
- Browser software, such as Netscape and Microsoft Explorer, is inexpensive and easy to use.
- Web software for the server side is also inexpensive and easy to install.
- Existing programs for data retrieval and manipulation can be modified for the back-end.
- Version control is eliminated.

This environment is ideal for companies with remote users that often find it difficult to keep each user up to date with the current client software version. With the Internet/Intranet approach each time the user connects to the Web server, the server actually uploads a new version. Since the application actually resides on the Web server, distribution of client programs is not necessary.

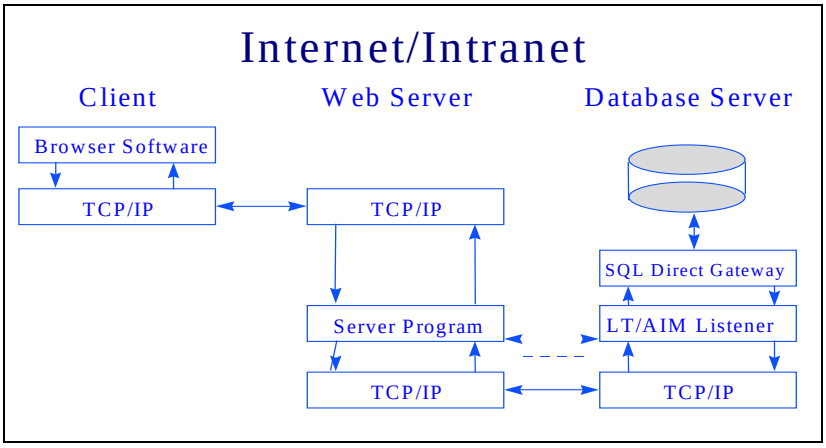

Figure 1-10. Internet/Intranet Tiers

If the need arises, all of the examples above may be multi-tier by using the middleware to connect to the appropriate computer.

Middleware Features

LT/AIM's middleware tool is targeted at enterprise and large applications, supplying more than a simple development API. It has the following components:

1. Open Systems Architecture supports multiple hardware platforms, operating systems and databases.
2. Support for broad systems architectures and application requirements.
3. Transparent communications on LANs, WANs, serial links and mobile links.
4. Simple and convenient client and server APIs.
5. Provision for both group development and separate development of client and server programs.
6. A system-wide testing facility that allows independent testing and debugging of client and server programs.
7. A system control console that provides operations support after implementation.

8. Built-in encryption of network communication between the client and the server.

2. LeeTech AIM/CSF (Client Server Foundation)

LeeTech AIM/CSF is a true client/server product. It requires the placement of the Application Programming Interfaces (APIs) shown below, in shaded boxes, within the application running on the client and the application running on the host, as shown in the following diagram.

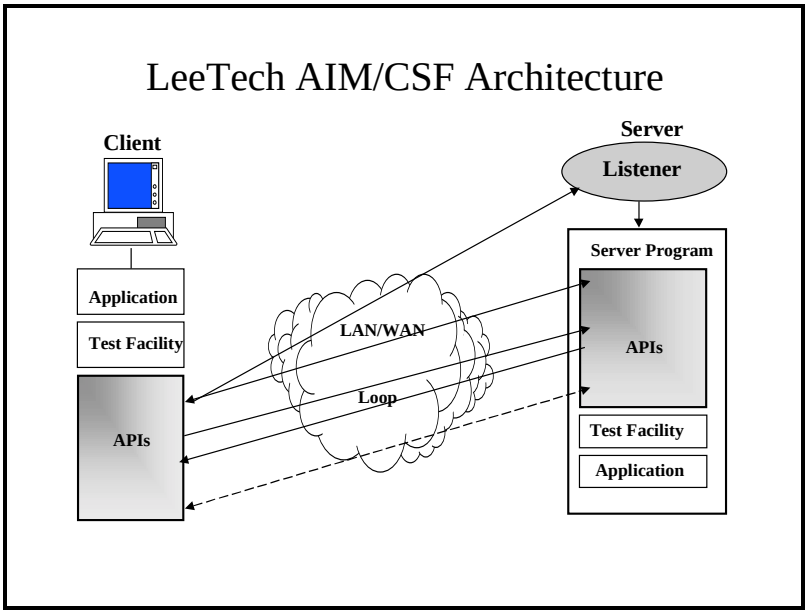

Figure 2-1. LT/CSF Middleware Connectivity

The following sections of this chapter discuss the various APIs required within the client and server programs.

Client API Overview

In client/server architecture, a client computer station issues a request for data to a server via a network. Upon receipt of the data request, the server retrieves the requested data from the server database or any data sources, and sends that data back to the requesting client. LeeTech AIM/CSF provides the client with an API in the form of a Dynamic Link Library

(DLL) that the application can call to handle message formatting, connection and transmission via network to the server.

LeeTech AIM/CSF provides a listener program that handles all incoming connection requests and wakes up the corresponding server programs which will process the data request. The server application programs then pass the data back to the client, who receives the data message and turns it over to the graphical user interface (GUI) application for processing.

LT/CSF supports Windows 3.1/ 95/98/NT, UNIX, HP MPE/iX and IBM AS/400.

General Client APIs

The client side of the LeeTech AIM/CSF library consists of the simplified communications verbs: *enter, send, recv and leave* in four functional API routines. These client routines are contained in the following library files:

Table 2-1. Client Routine Library Files

Platform	Server Routine	Description
UNIX	libcsfc.a	LeeTech AIM/CSF library for UNIX. This file must be linked with the client program. It is located in opt/leetech/lib.
HP3000 MPE	LTXL.PUB.LEETECH	LeeTech AIM/CSF XL for MPE/iX. LINKEDIT altprog command may be used to add this XL to the client program.
	LTRL.PUB.LEETECH	LeeTech AIM/CSF RL for MPE/iX. It must be linked with the client program.
Windows	LTWS32.DLL	Dynamic Link Library for the Microsoft client.

Note: The Client API Routine column shows two syntaxes. The first one is the recommended syntax; the second is provided for backward compatibility with applications that use the previous LT/CSF client library. The two, however, are functionally equivalent.

Table 2-2. Client API Routine

Client API Routine	Description
Connect to Server *lt_msg_enter_clt( )* *lt_msg_enter( )*	Initiates a new connection to the server program.
Send Data *lt_msg_send_clt( )* *lt_msg_send( )*	Sends a buffer of data from the client to the server.
Receive Data *lt_msg_recv_clt( )* *lt_msg_recv( )*	Receives a buffer of data from the server.
Close the Connection *lt_msg_leave_clt( )* *lt_msg_leave( )*	Informs the server that the client is leaving the environment.
Encrypt Transaction *lt_msg_secure( )*	Encrypts transactions for additional security.
Errors Reported *ItWinSockError*	Retrieves Windows socket errors (valid only for clients running Microsoft Windows)

Each of the API routines and their parameters are discussed in detail on the following pages. Error Numbers generated by the routines are detailed in the Error Messages section, and the *lt_msg_secure* routine is discussed in Chapter 5.

Note: In the following sections, the terms “character array” and “character string” are used. A character array refers to a variable of a fixed-size character buffer. A character string refers to a character array with a NULL **(‘% 0’)** terminator at the end.

Connecting to the Server (lt_msg_enter_clt)

Description The client makes initial contact with the server via the *lt_msg_enter_clt* routine. This is the first call in the client program.

Syntax	csf_id = lt_msg_enter_clt(host, service, program, error)

Parameters

csf_id Returned; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

It must be saved for subsequent *lt_msg_send_clt* and *lt_msg_recv_clt* calls. *lt_msg_leave_clt* will release it.

host Input (required); character string; passing by address/reference

Host contains the server host machine name or its IP address. It may be predefined in the hosts file on the client (See Installation - HOSTS file) or from a DNS server.

service Input (required); character string; passing by address/reference

Service contains the PORT_ID or the alias name of the PORT_ID which is listened to by the server listener. It must be predefined in the services file on the client (See Installation - SERVICES file).

program Input (required); character string; passing by address/reference

Character string contains the full path of the server program name.

error Returned; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by address/reference

0 = OK
9999 = Connections exceed the maximum Server user licenses currently used. Call LeeTech to upgrade the user licenses.
< 0 = Network connection error

Example

```
Dim HostName As String
Dim PrgName As String
Dim AppName As String
HostName = "190.0.01"
PrgName = "CUSTLOOK.CSF"
AppName = "CSF"
SD = lt_msg_enter(HostName, AppName, PrgName, Error_no%)
If (Error_no% <> 0) Then
    If (Error_no% > 0) Then
        Msg$ = "[ERROR-" + Str$(Error_no%) + "] Too many connections, “
+ _
               "please contact LeeTech to increase the number of “ + _
               "connections to your User License."
        MsgBox (Msg$)
        End
    Else
        Msg$ = "[ERROR-" + Str$(Error_no%) + "] Connection failure."
        MsgBox (Msg$)
        End
    End If
End If
```

Closing the Connection (lt_msg_leave_clt)

Description The *lt_msg_leave_clt* routine tells the server that the client is leaving the environment. It instructs the server to release the network resource that was originally assigned when connectivity was established by the *lt_msg_enter_clt* connection routine.

Syntax	**error = lt_msg_leave_clt(csf_id)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

csf_id Input (required); *lt_msg_enter_clt* returned value; passing by value

Example

```
Error_no% = lt_msg_leave(SD)
If (Error_no% <> 0) Then
    Screen.MousePointer = DEFAULT
    Msg$ = "Socket ID:" + (Str$(SD)) + ",  Error Number:" +
(Str$(Error_no%))
    MsgBox (Msg$)
End If
```

Sending Data to the Server (lt_msg_send_clt)

Description The *lt_msg_send_clt* routine passes a buffer of data to the server program on the host system.

Syntax	**error = lt_msg_send_clt(csf_id, buffer, buf_len)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

csf_id Input (required); *lt_msg_enter_clt* required value; passing by value

buffer Input (required); character array; passing by address/reference

Buffer contains the client's request. The maximum buffer size is limited to 30,000 bytes.

buf_len Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value

The length of the client’s request buffer.

Example

```
Dim LT_VERSION_ID As String * 6
Dim LT_TRANX_ID As String * 4
Dim LT_TRANX_TYPE As String * 2
Dim LT_TRANX_ERR As String * 4
Dim LT_MORE As String * 4
Dim LT_DATA As String * 4
Dim LT_DEBUG_FILENAME As String * 26

LT_VERSION_ID = "A.0.00"
LT_TRANX_ID = "0000"
LT_TRANX_TYPE = "00"
LT_TRANX_ERR = "0000"
LT_MORE = "0000"
LT_DEBUG_FILENAME = "bugs"
LT_DATA = LT_VERSION_ID + LT_TRANX_ID +
          LT_TRANX_TYPE + LT_TRANX_ERR + LT_MORE +
          LT_DEBUG_FILENAME
Error_no% = lt_msg_send(SD, LT_DATA, 46)
If (Error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(Error_no%) + "] Message send failure."
    MsgBox (Msg$)
    End
End If
```

Receiving Data from the Server (lt_msg_recv_clt)

Description The *lt_msg_recv_clt* routine is used to receive a buffer of data from the server program on the host system. When this function is called, the client waits for the server to send the data. This routine can be called successively in a loop, to retrieve more than one buffer of data without having to use *lt_msg_send_clt.*

Syntax	**error = lt_msg_recv_clt(csf_id, buffer, buf_len)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

csf_id Input (required); *lt_msg_enter_clt* required value; passing by value

buffer Input (required); character array; passing by address/reference; maximum buffer size is 30,000 bytes.

buf_len Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value

The buf_len parameter is the length of client's reply buffer. It may be the same as the length specified in the corresponding *lt_msg_send ( )* call in the server program. If the client buffer is greater than or equal to the server buffer, the whole server reply will be transmitted into the client's buffer; otherwise, LT/CSF returns an error indicating that the buffer is too small.

Example

```
LT_DATA = String$(2001, 0)
Error_no% = lt_msg_recv(SD, LT_DATA, 2000)
If (Error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(Error_no%) + "] Message receive failure."
    MsgBox (Msg$)
    End
End If
```

Errors Reported (ltWinSockError)

Description The *ltWinSockError* routine gets the last Windows socket error generated by the LT/CSF calls. This function call is only available for LeeTech AIM/CSF Windows's clients.

Syntax	**error = ltWinSockError(errno, desc, desc_len)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

errno Returned; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by address/reference. It contains windows socket error number.

desc Returned; character string; passing by address/reference. It contains the windows socket error message.

desc_len Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value. It is the size of the desc (256 maximum).

Examples

Visual Basic

```
Declare Function ltWinSockError Lib "ltws.dll" _
                          (ByRef errno AS Integer, _
                           ByVal Desc As String, _
                           ByVal Desc_len As Integer) As Integer

Dim Errno As Integer
Dim Desc As String * 200
Dim Desc_len As Integer

Desc_len = 200
error_no% = ltWinSockError (Errno, Desc, Desc_len)
If (error_no% = 0) Then
    Msg$ = Desc
    Exit Sub
End If
```

C

```
extern int FAR PASCAL ltWinSockError (int FAR *, char Far *, int);

int error_no;
char desc[200];
int desc_len;
int errno;

desc_len = 200;
error_no = ltWinSockError (&errno, desc, desc_len);
if (error_no == 0)
    printf ("%s", desc);
```

Encrypting Transactions (lt_msg_secure)

Description The *lt_msg_secure* call encrypts transactions for additional security.

Syntax	**error = lt_msg_secure (csf_id, opt)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

csf_id Input (required); *lt_msg_enter_clt* returned value; passing by value

opt Returned (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by address/reference

Set the opt = 0 to disable encryption.
Set the opt = 1 to enable encryption.

Example

Visual Basic

```
Public SD As Integer
Declare Function lt_msg_secure % Lib "ltws.dll (ByVal SD%, ByVal Opt%)

Dim opt As Integer
opt = 1
error_no% = lt_msg_secure(SD,opt)
If (error_no% <> 0) Then
    msg$ = "[ERROR-" + Str$(error_no%) + "] Security failure." MsgBox
(msg$)
Exit Sub
End IF
```

Sample Application

Customer Name File (MPE/iX)

This sample application contains 2 parts; the PC client part which contains the execution file LTCS.EXE and the source project LTCS.MAK is written in Microsoft Visual Basic. The HP3000 server which contains the program file LTVB.PUB.LEETECH and the source file LTVBS.PUB.LEETECH is written in COBOL. First, the client issues a network connection and triggers the LeeTech/CSF listener to create a server process which is LTVB.PUB.LEETECH. Then the client issues a transaction “0001” request to the server process to get a customer name list from a MPE/iX flat file

CUSTDTA.PUB.LEETECH that sends all records back to the client. Then the client puts them onto a spreadsheet for display.

1. On the HP3000 server, start the listener first.

   :STREAM JLTLSTN.PUB.LEETECH

5. Run LTCS.EXE on the PC client or click on the *VB Sample* icon under LeeTech/CSF group.

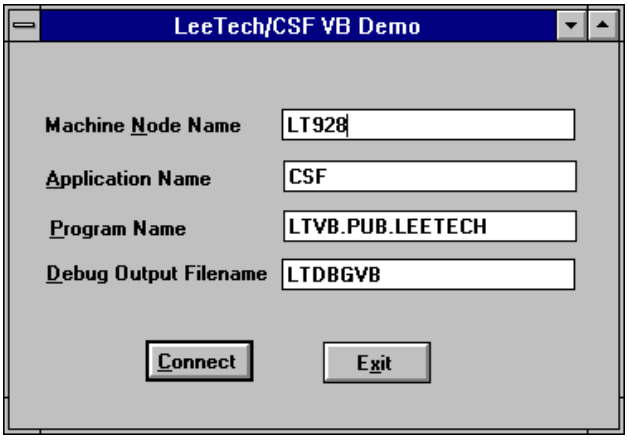

Figure 2-2. VB Demo Window

6. After entering Machine Node Name, Application Name (services/PORT_ID), server program name (LTVB.PUB.LEETECH) and Debug Output Filename (LTDBGVB), click on the **Connect** button.

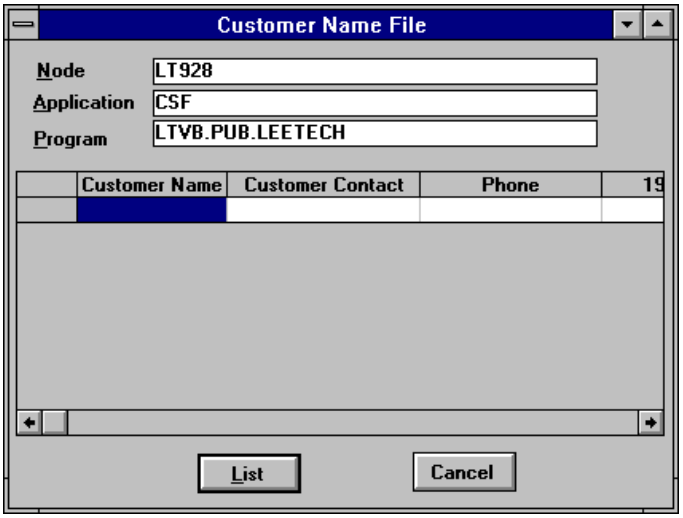

Figure 2-3. Customer Name File Window

7. Click on the **List** button to get a list of customers from the HP3000.

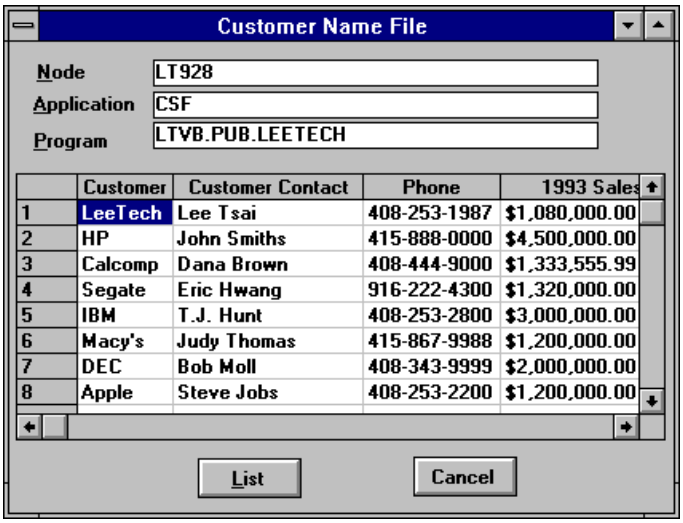

Figure 2-4. Listing of Customers

Server API Overview

LT/CSF supports application servers on a number of different computers. Most of these servers can also be clients to any of the other computers that LT/CSF supports. LT/CSF differentiates between the client and the server by which one initiates the request for services.

The LT/CSF supported server computers are:

- Most UNIX platforms — HP-UX, Sun/Solaris, SCO/UNIX
- HP 3000 – MPE/iX
- IBM/CICS
- Windows NT

General Server APIs

There are API routines sitting on the server side of the client-to-host-server connection that are used to communicate between the client and the server. These server routines are contained in the following library files:

Table 2-3. Server Routine Library Files

Platform	Server Routine	Description
UNIX	Libcsf.a	LeeTech AIM/CSF library for UNIX. This file must be linked with the server program. It is located in opt/leetech/lib.
HP MPE/iX	LTXL.PUB.LEETECH	LeeTech AIM/CSF XL for MPE/iX. Use LINKEDIT altprog command to add this XL to the server program.
	LTRL.PUB.LEETECH	LeeTech AIM/CSF RL for MPE/iX. It must be linked with the server program.
Windows NT	LTWS32.DLL	Dynamic Link Library for the Microsoft NT server.

Table 2-4. Server API Routine

Server API Routine	Description
Establish Connection *lt_msg_enter( )*	Establishes the network connection between the client and the server programs.
Close Connection *lt_msg_leave( )*	Disconnects the network connection between the client and the server programs.
Receive Data *lt_msg_recv( )*	Receives a request from the client.
Send Data *lt_msg_send( )*	Sends the response to a client's request back to the client.
Register Information to the Console *lt_msg_setinfo( )*	Registers the user's information at runtime to the LeeTech AIM/CSF connection console.
Switch from Default Logon *lt_switch_logon( )*	Switches the server program from the default Listener's logon into a specific user logon.

Establishing Connection (lt_msg_enter)

Description The *lt_msg_enter* establishes a network connection between the Client and Server programs. It registers the Server program in LT AIM/CSF's registry and takes over the connection from the listener. This routine should be called only once, at the beginning of the Server program.

Syntax	**lt_msg_enter (buffer, error)**

Parameters

buffer Input (required); maximum 35 character string; passing by address/reference

It contains information about the program that shows in the LT/CSF Monitor's registry to uniquely identify a specific server program (or so called user). The maximum length is 35 characters and the string must be terminated with a NULL character (%0).

error Returned value (required); integer (4-byte); passing by address/reference

0 = O

Examples

COBOL

```
 77  LT-DATA-LENGTH             PIC S9(09) COMP.
 77  LT-ERROR                   PIC S9(09) COMP.
 01  LT-DATA-BUFFER.
     02 LT-VERSION-ID           PIC  X(06).
     02  LT-TRANX-ID            PIC  9(04).
     02  LT-TRANX-TYPE          PIC  9(02).
     02  LT-TRANX-ERR           PIC  9(04).
     02  LT-MORE                PIC  9(04).
     02  LT-DATA                PIC  X(1980).


     MOVE "LeeTech/Core Test Program" TO LT-RUN-ID.
     STRING LT-RUN-ID          DELIMITED BY "  "
            LT-NULL-TERMINATOR DELIMITED BY SIZE
       INTO LT-DATA-BUFFER.
     MOVE ZERO TO LT-ERROR.
     CALL "lt_msg_enter" USING LT-DATA-BUFFER, LT-ERROR.
     IF LT-ERROR NOT = 0
        DISPLAY “Error in LT_MSG_ENTER”
     END-IF
```

FORTRAN

```
$ALIAS LT_MSG_ENTER (%REF, %REF)
$ALIAS LT_MSG_RECV  (%REF, %VAL, %REF)
$ALIAS LT_MSG_SEND  (%REF, %VAL, %REF)

      INTEGER*4 LT_DATA_LEN, LT_BUFF_LEN

      PARAMETER (LT_DATA_LEN = 174, LT_BUFF_LEN = 194)

      CHARACTER LT_DATA*172,
     +          LT_BUFF*192,
     +          LT_VERS_ID*6,
     +          LT_TRANX_ID*4,
     +          LT_TRANX_TYPE*2,
     +          LT_TRANX_ERR*4,
     +          LT_MORE*4,
     +          LT_WC_DESCR*20,
```

```
      LT_BUFF = "MICS Job selection"
      CALL LT_MSG_ENTER(LT_BUFF,ERR)
      IF (ERR .NE. 0) THEN
          WRITE (6,'("0** ERROR",I5," OCCURRED IN LT_MSG_ENTER")') ERR
          CALL PROGERR(-1,PROG,10.0)
          GOTO 999
      ENDIF
```

Closing the Connection (lt_msg_leave)

Description The *lt_msg_leave* routine disconnects the network connection between the Client and Server programs and drops the server program registration from LT/CSF's registry. It should be called at the end of the Server program.

Syntax	**lt_msg_leave ( )**

Examples

COBOL

```
CALL "lt_msg_leave".
```

FORTRAN

```
$ALIAS LT_MSG_LEAVE
 CALL LT_MSG_LEAVE
```

Receiving Data (lt_msg_recv)

Description The *lt_msg_recv* routine receives the request for data from the client.

Syntax	**lt_msg_recv (buffer, length, error)**

Parameters

buffer Input (required); character array; passing by address/reference

Receiving buffer for client's request; maximum of 30,000 characters

length Input (required); integer (4-byte); passing by value

It is the expected receiving buffer length.

Must be the same length or greater than the incoming Client request buffer size

error Returned value (required); integer (4-byte); passing by address/reference

0 = OK

Example

COBOL

```
     MOVE SPACES     TO LT-DATA-BUFFER.
     MOVE 242        TO LT-DATA-LENGTH.
     MOVE 0          TO LT-ERROR.
     CALL "lt_msg_recv" USING  LT-DATA-BUFFER,
                              \LT-DATA-LENGTH\,
                               LT-ERROR.
```

FORTRAN

```
$ALIAS LT_MSG_RECV (%REF, %VAL, %REF)

      L = 52
      CALL LT_MSG_RECV(LT_BUFF,L,ERR)
      IF (ERR .NE. 0) THEN
          WRITE (6,'("0** Error",I5," occurred in LT_MSG_ENTER")')  ERR
      ENDIF
```

Sending Data (lt_msg_send)

Description The *lt_msg_send* routine sends the data response from the server to the requesting client.

Syntax	**lt_msg_send(buffer, length, error)**

Parameters

buffer Input (required); character array; passing by address/reference

Response data to Client; maximum of 30,000 characters

length Input (required); integer (4-byte); passing by value
The response data length
The Client receiving buffer size must be set to the same or larger than the Server response data length

error Returned value (required); integer (4-byte); passing by address/reference

0 = OK

Examples

COBOL

```
    MOVE LOW-VALUES    TO LT-DATA-BUFFER
    MOVE CUSTOMER-NAME TO TCN-CUSTOMER-NAME
    MOVE CONTACT       TO TCN-CUSTOMER-CONTACT
    MOVE PHONE-NO      TO TCN-CONTACT-PHONE
    MOVE CUSTOMER-NO   TO TCN-CUSTOMER-NO
    MOVE CITY          TO TCN-CITY
    STRING ADDRESS-1      DELIMITED BY SIZE, " ", _
           ADDRESS-2      DELIMITED BY SIZE
      INTO TCN-ADDRESS
    MOVE STATE         TO TCN-STATE
    MOVE "A.0.00"      TO LT-VERSION-ID
    MOVE "0001"        TO LT-TRANX-ID
    MOVE "00"          TO LT-TRANX-TYPE
    MOVE "0000"        TO LT-TRANX-ERR
    MOVE "0001"        TO LT-MORE
    MOVE 204           TO LT-DATA-LENGTH
    MOVE 0             TO LT-ERROR
    CALL "lt_msg_send" USING  LT-DATA-BUFFER,
                             \LT-DATA-LENGTH\,
                              LT-ERROR
```

FORTRAN

```
$ALIAS LT_MSG_SEND (%REF, %VAL, %REF)

      LT_TRANX_ERR = "0001"
```

```
      LT_DATA = VERSION
      L = 52
      CALL LT_MSG_SEND(LT_MSG_BUFF,L,ERR)
      IF (ERR .NE. 0) THEN
          WRITE (6,'("0** Error",I5," occurred in LT_MSG_SEND")') ERR
    ENDIF
```

Registering Information to the Console (lt_msg_setinfo)

Description The *lt_msg_setinfo* routine registers user-supplied information to the console. The information should contain data that uniquely identifies the user's connection on the enterprise network. The LT/CSF Console uses this information to effectively identify and manage the enterprise client/server environment.

Note: The *lt_msg_setinfo* routine replaces the *lt_msg_bind* and *lt_msg_unbind* functions.

Syntax	**error = lt_msg_setinfo(info, rc)**

Parameters

error Returned value; integer (4-byte)

0 = OK

info Input (required); character string; passing by address/reference

This is the unique identifier string for each of the user's connections (35 maximum).

rc Returned (required); integer (4-bytes); passing by address/reference

Same as **error**; is used as an alternate method to receive error code.

Example

COBOL

```
01      LT-INFO      PIC X(35)
01      LT-RC        PIC S9(09) COMP.
01      LT-RETURN1   PIC S9(09) COMP.

    Call "lt_msg_setinfo"     USING LT-INFO, LT-RC
                              GIVING LT-RETURN1
```

Switching from the Default to a Specific Logon (lt_switch_logon)

Description The *lt_switch_logon* routine switches the server program from the default Listener's logon domain into a specific user logon.

Syntax	**error = lt_switch_logon (logon)**

Parameters

logon Input (required); character string; passing by address/reference

The string must be terminated with a NULL (%0) character.

UNIX and Windows NT format. All values are case sensitive. The first character in the logon string is used as the string delimiter, so the first, last and user name/password separation characters of the logon string must be exactly identical. They are not part of the logon data but they are required**.**

EXAMPLE $logon$password$

"$" is the string delimiter from the above example, any character may be chosen as the delimiter. As long as it is not part of the logon or password.

MPE Format:

User Name [8]
User Password [8]
Account Name [8]
Account Password [8]
Group Name [8]
Group Password [8]

It is a fixed sized, 48 character string with NULL terminator.

Examples

COBOL in MPE/iX

```
01  LOGIN-DATA.
    05  LOGIN-USER           PIC X(8) VALUE “MGR”.
    05  LOGIN-USER-PASS      PIC X(8) VALUE “PASS1”.
    05  LOGIN-ACCT           PIC X(8) VALUE “TEST”.
    05  LOGIN-ACCT-PASS      PIC X(8) VALUE “PASS2”.
    05  LOGIN-GROUP          PIC X(8) VALUE SPACES.
    05  LOGIN-GROUP-PASS     PIC X(8) VALUE SPACES.
    05  FILLER               PIC X(1) VALUE %0.
01 ERROR -NO                 PIC S9(9) COMP.

   CALL "lt_switch_logon" USING LOGIN-DATA
                          GIVING ERROR-NO.
```

C in UNIX

```
    char logon[80];

    strcpy(logon,"$mgr/pass1$");
    lt_switch_logon(logon);
```

Client APIs for IBM/CICS

In a client to IBM mainframe server architecture, LT/CSF provides the client with an Application Programming Interface (API) in a Dynamic Link Library (DLL), as well as a Link Library (LIB), that the application can call to handle message formatting, connection, and transmission via the network to the IBM mainframe server.

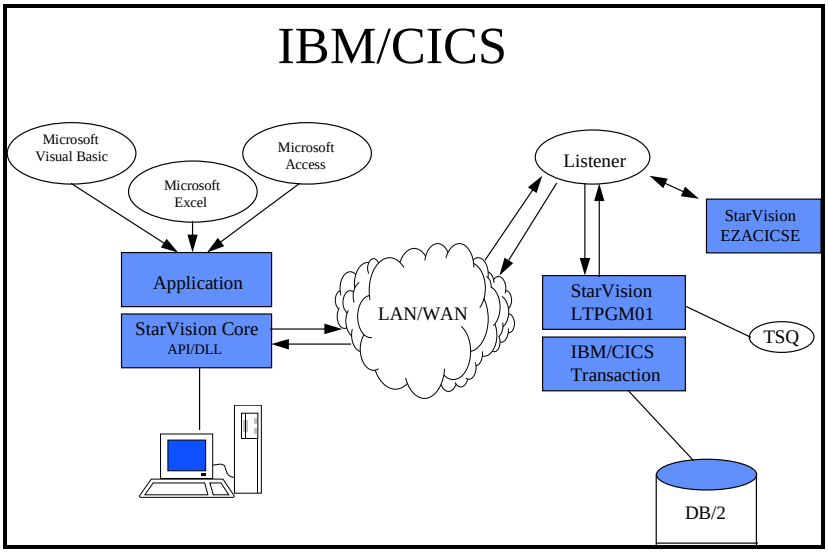

Figure 2-5. Client to IBM Mainframe Server Architecture

On the IBM mainframe server, LT/CSF has a main TCP/IP socket server that handles all CICS transactions within a specific CICS region. The socket server transmits incoming requests and outgoing replies from/to the client via the IBM/CICS TCP/IP listener. LT/CSF also has an IBM/CICS TCP/IP listener security *exit* routine which takes the pre-defined TERMID from the client and returns it back to the IBM/CICS TCP/IP listener to start the requested CICS transaction with the pre-defined TERMID.

LT/CSF for IBM/CICS contains API routines: *ltCICSenter, ltCICSsetopt, ltCICSgetopt, ltCICSsecurity, ltCICSexec* and *ltCICSleave*.

Table 2-5. IBM/CICS Client API Routine

API Routine	Description
Connect to the Server *ltCICSenter( )*	Establishes the network connection to the Listener and registers the unique user identification to the operation control console.

Table 2-5. Client API Routine (continued)

API Routine	Description
Set Variables and Options *ltCICSsetopt( )*	Sets options required by CICS.
Get Error Messages *ltCICSgetopt( )*	Returns error messages for CICS.
Sign-on Security *ltCICSsecurity( )*	Logs into system and passes the user ID and password to CICS.
Execute CICS Transactions *ltCICSexec( )*	Executes all CICS statements.
Close the Connection *ltCICSleave( )*	Informs the server that the client is leaving the environment and releases the assigned socket ID.

Connecting to the Server (ltCICSenter)

Description The *ltCICSenter* routine establishes the network connection to the listener and registers the unique user identification to the operation control console.

Syntax	**error = ltCICSenter (handle, host, service, lt_txid, info)**

Parameters

error Returned value ; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Return; character array (256 characters); passing by address/reference

host Input character string; passing by address/reference

Contains the IBM host machine name or IP address

service Input character string; passing by address/reference

Character string contains the CICS region name

lt_txid Input character string; passing by address/reference

Character string contains the default tran_code for LT/CSF middleware server module.

info Input character string; passing by address/reference

Character string contains the unique identifier of each end user connection. The maximum size of *info* string is 32 characters:

1 - 8:	Application ID, such as G/L, A/P, UPS, IST, etc.
9 - 9:	BLANK
10 - 17:	CICS Region Name
18 - 18:	BLANK
19 - 26:	MM/DD/YY
27 - 27:	BLANK
28 - 32:	HH:MM

Example

```
"PAYABLE  PAYCICS  04/02/95 16:30". If the info string length is 0,
then the ltCICSenter process will bypass the LeeTech Console
registration step and there will be no entry added to the info TSQ
file. The physical info TSQ file (Temporary Storage Queue on disk)
record size is 42 bytes.
```

Setting Variables and Options (ltCICSsetopt)

Description The *ltCICSsetopt* routine sets options required by CICS.

Syntax	**error = ltCICSsetopt(handle, option, option_value)**

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Character array (256 characters) returned value of *ltCICSenter* passing by address/reference

option Input; passing by value; integer (2/4-byte):
2 - LT_CICS_TERM_OPT
4 - LT_CICS_CONTROL

option_value Input character string; passing by address/reference

Table 2-6. ltCICSsetopt Options and Values

Value	Option	Option Value
2	LT_CICS_TERM_OPT	The value are either NULL_TERM (String$(4,0)) or any valid TERMID. Size: Character String of 4
4	LT_CICS_CONTROL	The value is the return value of *ltCICSgetopt.* Size: Character String of 10

Getting Error Messages (ltCICSgetopt)

Description The *ltCICSgetopt* routine returns error messages for CICS

Syntax	**error = ltCICSgetopt (handle, option, option_value)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Returned value of ltCICSenter

option Input; passing by value integer (2-byte for 16-bit OS, 4-byte for 32-bit OS):

3 - LT_CICS_RESP_CODE
4 - LT_CICS_CONTROL

option_value Output; passing by address/reference

Table 2-7. ltCICSgetopt Options and Values

Value	Option	Option Value
3	LT_CICS_RESP_OPT	The values are broken into two parts, the first 2 characters are the level 1 message and the remaining 8 characters are the level 2 detail message. Size: Character String of 10 Level 1: 00-no error 01-error Level 2: +0000001 Generic Error +0000011 Bad TERMID +0000016 TERMID is required +0000027 Bad Program ID +0000069 Bad User ID +0000070 Bad Password Refer to CICS/ESA Application Programming Reference, page 316, (EIBRESP) for more details.
4	LT_CICS_CONTROL	The value is an unique identifier assigned by LT/CSF server program to uniquely identify each user's registration record in TSQ file. Size: Character String of 10

Signon Security (ltCICSsecurity)

Description The *ltCICSsecurity* routine logs into system and passes the user ID and password to CICS

Syntax	**error = ltCICSsecurity(handle, termid, userid, password, retry)**

Note: This function MUST be called before the *ltCICSexec* is called.

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Returned value of *ltCICSenter* passing by address/reference

termid Input; passing by address/reference character string of 4: any valid termid

userid Input; passing by address/reference character string of 8: any valid userid

password Input; passing by address/reference character string of 8

retry Input; passing by value

Integer: maximum number of CICS sign-on retries

If retry = 0, then this process will retry *forever* until successful sign on.

Executing CICS Transactions (ltCICSexec)

Description The *ltCICSexec* routine executes all CICS statements.

Syntax	**error = ltCICSexec(handle, txid, request, request_length, reply, reply_length)**

Note: The *ltCICSsecurity* function MUST be called before this function is called.

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Returned value of *ltCICSenter* passing by address/reference

txid Input; passing by address/reference

Character string of 4: any valid CICS tran_code

request Input; passing by address/reference; character string of 1 - 30,000

request_length Input; passing by value integer: the actual length of request

reply Input; passing by address/reference; character string of 1 - 30,000

reply_length Input; passing by value integer: the actual length of reply

Closing the Connection (ltCICSleave)

Description The *ltCICSleave* routine informs the server that the client is leaving the environment and releases the assigned socket ID.

Syntax	**error = ltCICSleave(handle)**

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Returned value of *ltCICSenter* passing by address/reference

General Error Messages

LTECONFAIL **-2001**

Cause Connection failed.

1. The server hostname is invalid; the hostname is not defined in the local PC HOST file, the IP address is invalid, or the hostname is not defined in the network domain name directory.
2. The service (port_id) is invalid; the service (port_id) is not defined in the local PC SERVICES file, or the server listener which listens to the specified service (port_id) is not up and running.
3. The server program is invalid.

Action Correct the problem, then retry *lt_msg_enter_clt().*

LTECSFAIL **-2002**

Cause LeeTech cannot establish the initial client/server connection. Either the client DLL and server listener version are out of sync or a security breach is attempted.

Action Verify the client and server versions and retry *lt_msg_enter_clt().*

LTEGETSERV **-2003**

Cause	Invalid service (port_id), the value is not in the local PC SERVICES file which is used by the WINSOCK.DLL.
Action	Verify the SERVICES file being used, make sure the service (port_id) is defined in that file, and retry *lt_msg_enter_clt().*

LTEHOSTNOTFOUND **-2004**

Cause	Invalid hostname, the value is not in the local PC HOST file which is used by the WINSOCK.DLL.
Action	Verify the HOST file being used, make sure the hostname is defined in that file with the correct IP address, and retry *lt_msg_enter_clt().* Use the ping utility to verify the correctness of an IP address.

LTEINVALIDIP **-2005**

Cause	The IP address is invalid, or the host machine is off-line, or is not available for access.
Action	Use the ping utility to verify the correctness or availability of an IP address.

LTEIPFAIL **-2006**

Cause	The IP address is invalid, or the host machine is off-line, or it is not available to access.
Action	Use the ping utility to verify the correctness or availability of an IP

address.

LTENOSOCK **-2007**

Cause The *lt_msg_enter_clt()* routine must be called first before any other API can be used.

Action Call *lt_msg_enter_clt()* first.

LTERECVFAIL **-2008**

Cause Receive failed. Make sure the data buffer is appropriately defined and that the length matches the defined data buffer size.

Action Correct the problem, retry *lt_msg_recv.*

LTESENDFAIL **-2009**

Cause Send failed. Make sure the data buffer is appropriately defined and that the length matches the defined data buffer size.

Action Correct the problem, retry *lt_msg_send.*

LTESTARTFAIL **-2010**

Cause	LeeTech cannot start the specified program, due to: 1. Loader error, such as unresolved external, cannot access predefined XL files that are associated with the program, 2. Invalid program file format or attributes, 3. Program file access security violation, or 4. A backup process is locking program file.
Action	Correct the problem, then retry *lt_msg_enter_clt().*

LTEDRFAIL **-2011**

Cause	The connection was dropped abnormally during data transmission.
Action	Re-initiate the connection by calling *lt_msg_enter_clt().*

IBM/CICS Error Messages

CICS_ERR_INVALID_HANDLE **-1**

Cause	HANDLE buffer contains invalid value; 1) The *ltCICSenter* has not been called yet, or 2) *ltCICSleave* needs to be called before it is possible to use the same HANDLE to initiate another *ltCICSenter call*.
Action	Call *ltCICSenter.*

CICS_ERR_INVALID_HOSTNAME **-2**

Cause HOSTNAME is more than 16 characters long.
Action Correct the value and retry *ltCICSenter*.

CICS_ERR_INVALID_SERVICE **-3**

Cause SERVICE is more than 16 characters long.
Action Correct the value and retry *ltCICSenter*.

CICS_ERR_INVALID_USERID **-4**

Cause CICS USERID is more than 8 characters long.
Action Correct the value and retry *ltCICSsecurity*.

CICS_ERR_INVALID_PASSWD **-5**

Cause PASSWORD is more than 8 characters long.
Action Correct the value and retry *ltCICSsecurity*.

CICS_ERR_INVALID_INFO **-6**

Cause INFO is more than 32 characters long.
Action Correct the value and retry *ltCICSenter*.

CICS_ERR_INVALID_TXID **-7**

Invalid CICS tran_code.

Cause
1. The tran_code must be 4 characters exactly for *ltCICSenter* and *ltCICSexec*, or
2. The tran_code is an empty string, then the default tran_code set by *ltCICSenter* will be used for *ltCICSexec*.

Action Correct the value and retry the call.

CICS_ERR_INVALID_REQLEN **-8**

Cause Invalid request buffer length. It must be between 1 through 30,000.

Action Correct the value and retry the *ltCICSexec*.

CICS_ERR_INVALID_REPLEN **-9**

Cause Invalid reply buffer length. It must be between 1 through 30,000.

Action Correct the value and retry the *ltCICSexec*.

CICS_ERR_INVALID_OPT **-10**

Cause Invalid option value.

Action Correct the value and retry *ltCICSsetopt* or *ltCICSgetopt.*

CICS_ERR_INVALID_TERM **-11**

Cause CICS TERM_ID must be 4 characters long.

Action Correct the value and retry the call.

CICS_ERR_INVALID_CONTROL **-12**

Cause Invalid client/server control key which is set by LeeTech Middleware as a security token. The control key is 9 characters long.

Action There might be a security breach. Inform a security administrator immediately.

CICS_ERR_CONNECT_FAIL **-101**

Cause Invalid hostname, the specified hostname/IP_address does not exist in the local HOSTS file, or the hostname has not been defined in the network domain name directory. Invalid service (port_id), the specified service (port_id) doesn't exist in the local SERVICES file, or the CICS listener that listens to the specified service (port_id) is not up and running.

Action Correct the problem and

retry the call.

CICS_ERR_SEND_TX_FAIL **-102**

Cause Cannot start the CICS transaction from the server. The CICS region may be down, the CICS listener is not up and running, or the network is down.

Action Contact technical support, fix the problem and retry the call.

CICS_ERR_WRITE_REQ_FAIL **-103**

Cause Cannot send data to the IBM server. The CICS region may be down, the CICS listener is not up and running, the server program was aborted or the network is down.

Action Contact technical support, fix the problem and retry the call.

CICS_ERR_READ_RESP_FAIL **-104**

Cause No response from the IBM server. The CICS region may be down, the CICS listener is not up and running, the server program was aborted or the network is down. Request for server data length failed.

Action Contact technical

support, fix the problem and retry the call.

CICS_ERR_READ_REP_FAIL **-105**

Cause No response from the IBM server. The CICS region may be down, the CICS listener is not up and running, the server program was aborted, or the network is down. Request for server data failed.

Action Contact technical support, fix the problem and retry the call.

CICS_ERR_REMOTE_FAIL **-201**

Cause The server program was aborted abnormally. The default tran_code does not exist or is not capable to perform some of the CICS system level routines. User log TSQ is full.

Action Call *ltCICSgetopt* for detailed CICS error code.

3. LeeTech AIM/SDK (SQL Development Kits)

The LeeTech AIM/SDK is an SQL-based development tool. It contains a set of client/server libraries that give the ability to develop an entire application from a client. The client can be a PC, an HP 9000, an HP 3000, a Sun or an IBM RS/6000. The LeeTech AIM/SDK provides seamless access to most of the SQL databases (ALLBASE/SQL, IMAGE/SQL, ORACLE, Informix, DB2/2, SQL Server and Access) residing on different servers (HP 9000, HP 3000, NT, Sun and IBM RS/6000).

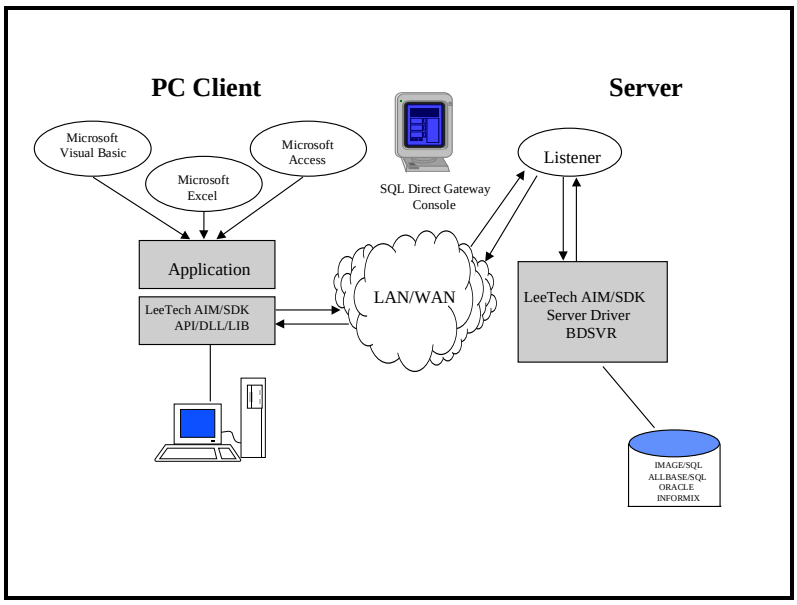

Figure 3-1. LeeTech AIM/SDK PC Client to Server Connectivity

GUI applications can be developed quickly and easily with the LeeTech AIM/SDK framework. LeeTech AIM/SDK interfaces with most PC front-end tools: Visual Basic, PowerBuilder, Gupta, Access, Excel, Lotus 1-2-3, etc.

Overview

In a client/server architecture, a client computer issues a request for data. In the case of LeeTech AIM/SDK, the request for data is in the form of SQL, to a server via a network. After the server receives the data request, it retrieves the requested data from the server database, or any other necessary data sources, and sends that data back to the requesting client.

The LeeTech AIM/SDK has both a client API and a server driver program. The API and server driver programs reside on top of the LeeTech AIM/CSF and provide additional features to simplify application development. The API accepts most of the SQL statements to query, update, delete and insert data.

The LeeTech AIM/SDK provides the PC client with an Application Programming Interface (API) in a Dynamic Link Library (DLL). It also has a Link Library (LIB) that the application can call to handle message formatting, connection and transmission. This is done via the network to the server. For clients other than the PC, the respective native library format is provided for linking to the application.

On the server side, LeeTech AIM/SDK uses the listener program to handle all incoming connection requests. LeeTech AIM also starts the LeeTech AIM/SDK SQL driver program that, in turn, processes the data request and passes the formatted data back to the requesting client. On the client side, the LeeTech AIM/SDK API receives the formatted data and turns it over to an application for processing.

Features and Benefits

1. The LeeTech AIM/SDK API can be used to directly implement the application. As a result, development of an application can occur on multiple platforms, using different programming languages with consistent APIs
2. No configuration is required to run an application, simplifying release distribution.
3. The LeeTech AIM/SDK API provides great flexibility for data formatting with any favorite tool. It is easy to manipulate and allows

data to be sent across different databases, platforms, tools and applications, without concern about data conversion.

4. A database and platform specific SQL driver program is provided with the LeeTech AIM/SDK. The driver program is the architect for performance at both the data fetching and the transmission levels. LeeTech AIM/SDK supports true SQL multiple-cursors and standard SQL transactions.

5. LeeTech AIM/SDK does more than 2-tier application development. All the APIs are also supported on the server side to accomplish a true multiple-tier client/server application framework. In fact, LeeTech AIM/SDK is the only SQL-based architecture that allows implementation of the OLTP application.

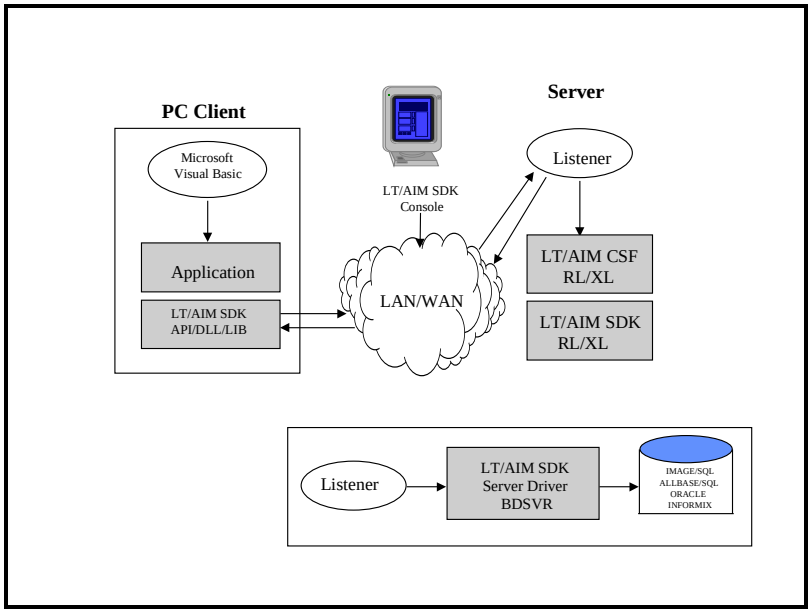

Figure 3-2. LeeTech AIM/SDK PC Client to Server Framework

API Routines

The LeeTech AIM/SDK has API routines for developing GUI applications and managing processes, users, network performance and host server load analysis on the client/server environment.

Table 3-1. API Routine Descriptions

API Routine	Description
Column Attributes *ltDBattr*	Executes an SQL SELECT statement and returns qualified rows.
Initiate Transactions *ltDBbegintx*	Begins a transaction.
Conclude Transactions *ltDBcommittx*	Completes a transaction and updates the database.
Connect to the Server Database *ltDBenter*	Establishes a network connection from the PC client to the server.
Execute SQL Statements *ltDBexec*	Executes all SQL statements, except the select statement.
Retrieve Column Value *ltDBgetcol*	Gets the column value from the returned *ltDBselect* rowbuf, returns the actual column value length and advances the rowbuf internal pointer to the next column.
Error Messages *ltDBgetopt*	Gets SQL standard error messages generated by the previous LeeTech AIM/SDK's call, or returns the current CURSOR_ID from the previous *ltDBselect* call.
Terminate Connections *ltDBleave*	Terminates the network connection from the PC client to the server established by calling *ltDBenter*.
Cancel Transaction *ltDBrollbacktx*	Cancels a transaction.
Retrieve Data from Remote Database *ltDBselect*	Executes a SQL SELECT statement and returns qualified rows.
Set Variables and Options *ltDBsetopt*	Sets LeeTech AIM/SDK's environment control variables and SQL execution options.

Table 3-1. API Routine Descriptions (continued)

API Routine	Description
Execute a User-Defined Function Remotely *ltDBusertx*	Remotely invokes user-defined server routines and communicates with single requests and replies.

Note: In the following sections, the terms "character array" and "character string" are used. A "character array refers to a variable of a fixed-size character buffer. A "character string" refers to a character array with a NULL ('%0') terminator at the end.

Connecting to the Database Server (ltDBenter)

Description The *ltDBenter* routine establishes a network connection from the client to the server. It starts the LeeTech AIM/SDK server program:

- MPE/iX - DBSVR.PUB.LEETECH
- UNIX - /opt/leetech/lbin/dbsvr.allbase, *or* /opt/leetech/lbin/dbsvr.informix,or /opt/leetech/lbin/dbsvr.oracle

It switches the server's logon from the Listener's default domain into the specified domain (*logon* parameter), and then connects to the specified database (*dbname* parameter).

Syntax	**Error = ltDBenter(handle, host, services, type, info, logon, dbname, opt)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Returned; character array (256 characters); passing by address/reference

It is a global variable that will be used for subsequent SDK calls. The *ltDBleave* routine releases the established connection and frees the *handle*. It can be called *ltDBenter* multiple times in the program as long as it does not exceed the network concurrent session limit

host Input (required); character string; passing by address/reference

It contains the server host machine name or its IP address. To use a logical host name, the host name must be predefined in the *hosts* file on the local computer (See Installation - *Hosts* file) or obtained from the IP address on the DNS server.

services Input (required); character string; passing by address/reference

It contains the TCP/IP port number or the alias name of the PORT ID, the port being listened to on the server. To use a logical service name, it must be predefined in the *services* file on the local computer (See Installation - *services* file).

type Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value

It contains the type of server and database management system.

Table 3-2. ltDBenter Server Type and Value

Type	Value
DB_TYPE_MPE_ALLBASE	257
DB_TYPE_MPE_IMAGESQL	257
DB_TYPE_HPUX_ALLBASE	513
DB_TYPE_HPUX_INFORMIX	514
DB_TYPE_HPUX_ORACLE	515

info Input (required); character string (maximum of 35 characters); passing by address/reference

It is used to uniquely identify a specific connection, which the system administrator can easily ascertain through the LeeTech AIM universal console. The following is an example:

Table 3-3. Sample Connection Identification

Character	Content
01-08	Application Module - s.a. G/L, A/P
09-09	one blank
10-17	User's ID - s.a. Employee ID
18-18	one blank
19-35	User's Phone Extension

logon Input (required); character string; passing by address/reference

It is used by LeeTech AIM/SDK's server program for server logon domain switching. A blank/empty string representing no logon domain switching is required; the default Listener's logon domain will be used. The logon domain will determine the user authority for specified databases.

HP MPE/iX format:

Table 3-4. Logon Information

Character	Content
01-08	User
09-16	User password
17-24	Account
25-32	Account password
33-40	Group
41-48	Group password

UNIX format: all values are case sensitive. Since the first character in the logon string is used as the string delimiter,

the first and last characters of the logon string must be identical. These strings are required, however, they are not part of the logon data.

$logon$password$

$ is the string delimiter from the above example, any character may be used as the delimiter.

dbname Input (required); character string; passing by address/reference

It is the database name that is to be connected to the LeeTech AIM/SDK server program. For IMAGE/SQL, this is the DBEnvironment (SQL), not the name of the IMAGE database.

opt Input (required); character string; passing by address/reference

It is reserved for future usage.

Example

Visual Basic

- **From file "declare.bas".**

```
'OS/Database type constant
Global Const DB_TYPE_MPEXL    = &H100
Global Const DB_TYPE_HPUX     = &H200
Global Const DB_TYPE_ALLBASE  = &H1

'Option constant
Global Const DB_OPT_SQLXERR  = 1
Global Const DB_OPT_BUFFER   = 2
 Declare Function ltDBenter% Lib "ltws.dll" (ByVal hDB$, ByVal Host$, _
                                             ByVal Service$, ByVal
                                             DBType%, _
                                             ByVal Info$, ByVal Logon$,
                                             _
                                             ByVal DBName$, ByVal opt$)
Declare Function ltDBgetopt% Lib "ltws.dll"(ByVal hDB$, ByVal Opt%, _
                                             ByVal Buffer$, ByVal
                                             Buf_Len%)

Global hDB As String * 256
```

- **ltDBenter**

```
Sub Begin_Program
    Dim info     As String * 35
    Dim dbname   As String
    Dim logon    As String
    Dim User     As String * 8
    Dim UserPwd  As String * 8
    Dim Acct     As String * 8
    Dim AcctPwd  As String * 8
    Dim Group    As String * 8
    Dim GroupPwd As String * 8

    info     = "POLICY 55285 408-253-2822"

DBtype%  = DB_TYPE_MPEXL Or DB_TYPE_ALLBASE 'MPEXL and ALLBASE/SQL

    User     = UCase$(txtUser.Text)
    UserPwd  = UCase$(txtUserPwd.Text)
    Acct     = UCase$(txtAcct.Text)
    AcctPwd  = UCase$(txtAcctPwd.Text)
    Group    = UCase$(txtGroup.Text)
    GroupPwd = UCase$(txtGroupPwd.Text)
    logon    = User & UserPwd & Acct & AcctPwd & Group & GroupPwd
    dbname   = txtDBName.Text
    hDB      = String$(256, 0)

    error_no% = ltDBenter(hDB, Trim$(NodeName.Text),
Trim$(AppName.Text), _
                          DBtype%, info, logon, dbname, opt$)
    If (error_no% <> 0) Then
       Screen.MousePointer = DEFAULT
       Msg$ = "[ERROR-" + Str$(error_no%) + "] Connection Failure"
       MsgBox Msg$, MB_OK, "LeeTech AIM/SDK"
       Exit Sub
    End If
End Sub
```

Initiating Transactions (ltDBbegintx)

Description The *ltDBbegintx* routine begins a transaction. All SQL actions within the same transaction are treated as one logical action. They are either all committed by *ltDBcommittx*, or all canceled by *ltDBrollbacktx*.

Locking Isolation Levels: To set the transaction's locking strategy *ltDBsetopt* is called with the *DB_OPT_ISOLATION_LEVEL* option. This step must be done before *ltDBbegintx* is called. Otherwise, the highest locking isolation repeatable read (RR) will be used by default. LeeTech AIM/SDK has 4 levels of locking isolations for ALLBASE/SQL, which are RR (repeatable

read), CS (cursor stability), RC (read committed) and RU (read uncommitted).

Explicit Transaction: We recommend using the explicit transaction, composed of *ltDBbegintx*, and either *ltDBcommittx* or *ltDBrollbacktx* to implement the client application. If this explicit method is not used, the implicit transaction rules are followed.

Implicit Transaction: The first *ltDBselect* initiates a transaction and remains active until it is closed by calling *ltDBselect* with the *DB_SELECT_CLOSE* option, or it closes by itself. LeeTech AIM/SDK's server driver maintains an internal buffer, containing rows that satisfy the SELECT statement. If there are no qualified rows, the cursor is automatically closed by the LeeTech AIM/SDK driver, even if the client program has not completely fetched all the rows from the server's internal buffer. This reduces the default transaction locking strategy (RR) to its minimum impact on the database.

Any *ltDBexec* calls made during the opened implicit *ltDBselect* transaction remain uncommitted until the *ltDBselect* routine is closed. Each independent *ltDBexec* call is treated as a standalone transaction, and is committed immediately after the SQL statement is executed.

Syntax	**Error = ltDBbegintx(handle, opt)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Input (required); returned by ItDBenter ( ); character array (256 characters); passing by address/reference

The parameter must be the character array returned by the *ltDBenter()* routine when the connection is established.

opt Reserved; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

Example

Visual Basic

- **ltDBsetopt - set isolation level**

ltDBbegintx - begin transaction

```
Sub Begin_Tranx
    opt$ = "RU"
    error_no% = ltDBsetopt(hDB, DB_OPT_ISOLATION_LEVEL, opt$,
Len(opt$))
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Isolation Level
Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
        Exit Sub
    End If

    error_no% = ltDBbegintx(hDB, Nullopt%)
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Begin Transaction
Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
        Exit Sub
    End If
 End Sub
```

- **ltDBcommittx - end transaction**

```
Sub End_Tranx

    error_no% = ltDBcommittx(hDB, Nullopt%)
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "]  Commit Work Failure"
        MsgBox Msg$, , "LeeTech AIM - LeeTech AIM/SDK"
        Exit Sub
    End If

 End Sub
```

Setting Variables and Options (ltDBsetopt)

Description The *ltDBsetopt* routine sets the LeeTech AIM/SDK environment control variables and the SQL execution options. All subsequent calls will be immediately affected by any variable or option changes.

Syntax	**error = ltDBsetopt(handle, opt, value, value_len)**

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Input; character array (256 characters); returned by *ltDBenter* passing by address/reference

opt Input (required); integer (2 byte for 16 bit OS; 4-byte for 32 bit OS); passing by value

It contains the following options:

Table 3-5. ltDBsetopt Options

OPT_DESCRIPTION	OPT
DB_OPT_BUFFER: **This option is specific for C language.** This option sets the default buffer used by subsequent *ltDBselect* calls. In the *ltDBselect* routine, the preset buffer will be used if the returned rowbuf is passed with a NULL value. This makes the memory allocation easier to maintain and speeds up the run-time execution *Example in C:* ..... error = **ltDBsetopt**(HANDLE,	2

Table 3-5. ltDBsetopt Options (continued)

OPT_DESCRIPTION	OPT
DB_OPT_BUFFER, **Gbuffer**, **Gbuffer_length);** ..... error = **ltDBselect** (HANDLE, DB_SELECT_OPEN, SQLStatement, &ncols, &nrows, **NULL**, NULL, &more); ..... The *Gbuffer* and its length (*Gbuffer_length*) will be used by the *ltDBselect* for storing the returned rows.	
DB_OPT_LEN_IN_ASCII: **The *ltDBgetcol* routine will be disabled with this option.** It converts the row length (if the DB_OPT_PREFIX_ROWLEN option is used) and the column length from 2-byte integer into a 5-byte numeric format (ASCII). If the value is a negative value, such as NULL value column, the first character will contain the negative sign "-"; otherwise; the value will be padded with a leading "0". The purpose of this option is a *DDE text data passing* among Visual Basic programs or programs developed by other tools supporting DDE.	3
DB_OPT_PREFIX_ROWLEN: This option will insert a row_length at the beginning of each row in the returned data buffer. This provides a quick way to move from one row to another, rather than scanning all columns.	4

Table 3-5. ltDBsetopt Options (continued)

OPT_DESCRIPTION	OPT
The row_length doesn't include itself; for example, in a 2-byte integer row_length option (the DB_OPT_LEN_IN_ASCII	
LeeTech AIM/SDK to return rows in a fix_length format. A fixed length of data will be always returned for the same column. Neither row_length nor column_length is included in the data buffer. The fixed column information (column name, attribute, NULL indicator, beginning position and offset) can be accessed anytime via *ltDBattr* routine. Note 1: A NULL value will be converted to a blank/empty character string, if the column is a CHAR type. A NULL value will be converted to 0, if the column is an INTEGER/SMALLINT/REAL type. Note 2: For VARCHAR type column, the maximum length is used. Note 3: For a DECIMAL type column, a sign and decimal point is used whether the scale is 0 or not. For example, the actual length for DECIMAL(3,0) is 5 ("+000." for value 0). Note 4: This is designed specifically for the IMAGE/SQL user. It manipulates the fixed length record at the client, and provides the foundation to allow the user to call on all Visual Basic DYNASET related routines (call Customer Support for more information).	5
DB_OPT_ISOLATION_LEVEL **The default isolation level is RR.** This option is used to set the locking isolation level for either an explicit transaction (use *ltDBbegintx* to begin a transaction) or an implicit transaction (use *ltDBselect* and *ltDBexec* directly without using *ltDBbegintx*). LeeTech AIM/SDK supports 4 levels of isolation: RR-	6

Table 3-5. ltDBsetopt Options (continued)

OPT_DESCRIPTION	OPT
repeatable read, CS-cursor stability, RC-read committed and RU-read uncommitted (refer to ALLBASE/SQL manual for details). The *value* for this option must be **2 characters long**, "RR," "CS," "RC" or "RU." The length of the *value* must be **2**.	
DB_OPT_CURSOR_ID This option is used to switch from the current cursor to a previously opened cursor. A cursor is opened with the *ltDBselect* routine and the CURSOR_ID is retrieved from *ltDBgetopt.*	7
DB_OPT_AUTO_CLOSE This option determines whether or not the LeeTech AIM/SDK server driver automatically closes the cursor as soon as all rows are fetched into the server's internal buffer. If *AUTO_CLOSE* is chosen, since the cursor has already been closed, the *ltDBgetopt* for the *CURSOR_ID* call may fail. If the application needs multiple cursors, it is recommended that it is set *DB_OPT_AUTO_CLOSE* = "N." This ensures that the cursor will be always open until *ltDBselect* is called with the *DB_SELECT_CLOSE* option to explicitly close it.	8
DB_OPT_SECURE This option encrypts the transaction for additional security. The values for this option can be) to disable encryption of 1 to enable encryption.	9

Value Input (required); character string; passing by address/reference

It contains the actual settings for the specific option.

Table 3-6. ltDBsetopt Settings

OPT	VALUE
DB_OPT_BUFFER	Data buffer variable; it must be a global variable and the spaces must be pre-allocated.
DB_OPT_LEN_IN_ASCII	"Y" or "N" (default)
DB_OPT_PREFIX_ROWLEN	"Y" or "N" (default)
DB_OPT_FIX_FORMAT	"Y" or "N" (default)
DB_OPT_ISOLATION_LEVEL	2 character string; it must be one of the following 4 values: "RR," "CS," "RC" or "RU."
DB_OPT_CURSOR_ID	8 character string.
DB_OPT_AUTO_CLOSE	"Y" or "N" (default)

value_len Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value. It is the length of the *value* parameter.

Table 3-7. ltDBsetopt Value Length

OPT	VALUE LENGTH
DB_OPT_BUFFER	The size of data buffer variable; it must be a global variable and the spaces must be pre-allocated.
DB_OPT_LEN_IN_ASCII	1
DB_OPT_PREFIX_ROWLEN	1
DB_OPT_FIX_FORMAT	1
DB_OPT_ISOLATION_LEVEL	2
DB_OPT_CURSOR_ID	8
DB_OPT_AUTO_CLOSE	1

Examples

C

- **DB_OPT_BUFFER**

```
#define DB_MAX_MSG_LEN  30000
#define DB_SELECT_OPEN  1
#define DB_OPT_BUFFER   2
#define OK              0

int  rc;
char buffer[DB_MAX_MSG_LEN];
char hDB[256];
char *stmt;
int  ncols;
int  nrows;
int  more;

rc = ltDBsetopt(hDB, DB_OPT_BUFFER, buffer, DB_MAX_MSG_LEN);
if (rc != OK)
    goto leave;
rc = ltDBselect(hDB, DB_SELECT_OPEN, stmt, &ncols, &nrows, NULL, _
                NULL, &more);
if (rc != OK)
    goto leave;

leave:
    rc = ltDBleave(hDB);
    if (rc != OK)
      exit(NULL);
```

Visual Basic

- **From file “declare.bas”.**

```
Global Const DB_OPT_BUFFER           = 2
Global Const DB_OPT_LEN_IN_ASCII     = 3
Global Const DB_OPT_PREFIX_ROWLEN    = 4
Global Const DB_OPT_FIX_FORMAT       = 5
Global Const DB_OPT_ISOLATION_LEVEL  = 6
Global Const DB_OPT_CURSOR_ID        = 7
Global Const DB_OPT_AUTO_CLOSE       = 8

Global Const DB_CURSOR_ID_LEN        = 8

Global hDB  As String * 256

Declare Function ltDBsetopt% Lib "ltws.dll" (ByVal hDB$, ByVal opt%, _
                                             ByVal ptr$, ByVal length%)
```

- **DB_OPT_LEN_IN_ASCII**

```
error_no% = ltDBsetopt(hDB, DB_OPT_LEN_IN_ASCII, "Y", 1)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set ASCII Option Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If
```

- **DB_OPT_PREFIX_ROWLEN**

```
error_no% = ltDBsetopt(hDB, DB_OPT_PREFIX_ROWLEN, "Y", 1)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Prefix RowLen Option
Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If
```

- **DB_OPT_FIX_FORMAT**

```
error_no% = ltDBsetopt(hDB, DB_OPT_FIX_FORMAT, "Y", 1)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Fix Format Option
Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If
```

Executing SQL Statements

Description The *ltDBexec* routine executes all SQL statements except the SELECT statement. Since *ltDBexec* does not return any data, use *ltDBselect* to execute an SQL statement and return qualified rows. It supports both SQL Database Definition Language (DDL - CREATE TABLE, DROP INDEX, ... etc.) and SQL Data Manipulation Language (DML - INSERT, UPDATE and DELETE).

Syntax	**error = ltDBexec(handle, stmt, nrows, tid, tid_len)**

Parameters

error Returned value; integer (2-bytefor 16 bit OS, 4 byte for 32-bit OS)

0 = OK

handle Input (required); returned by *ltDBenter;* character array (256 characters); passing by address/reference

stmt Input (required); character string; passing by address/reference

It contains the SQL statement.

nrows Returned (required); long (4-byte); passing by address/reference

It contains the number of rows being successfully update or deleted.

tid Returned (required); character array (128 characters); passing by address/reference

It contains the unique tuple identifier (*tid*) for the INSERTED row.

tid_len Input (required); integer (2-byte); passing by address/reference

It is the size of *tid*. Use 128.

Example

Visual Basic

- **From file "declare.bas".**

```
Global hDB  As String * 256
Declare Function ltDBexec% Lib "ltws.dll" (ByVal hDB$, ByVal Stmt$, _
                                           NumOfRow%, ByVal Tid$,
Tid_Len%)
Declare Function ltDBgetopt% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%, _
                                            ByVal Buffer$, ByVal
Buf_Len%)
```

- **ltDBexec – DDL**

```
error_no% = ExecSQL(hDB, "CREATE TABLE TDB1 (C1 INTEGER);")
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Create Table Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If

error_no% = ExecSQL(hDB, "DROP TABLE TDB1;")
```

```
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Drop Table Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If
```

- **ltDBexec - DML**

```
error_no% = ExecSQL(hDB, "INSERT INTO TDB1 VALUES (100);")
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Insert Table Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If

error_no% = ExecSQL(hDB, "DELETE FROM TDB1 WHERE C1 = 100;")
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Delete Table Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    Exit Sub
End If
```

- **ExecSQL function**

```
Function ExecSQL (hDB As String, Stmt As String) As Integer

    Dim Tid As String * 128

    error_no% = ltDBexec(hDB, Stmt, nrow%, Tid, Len(Tid))
    If (error_no% <> 0) Then
        GetSQLXERR(hDB)
    End if
    ExecSQL = error_no%

End Function
```

- **ltDBgetopt - get SQL Error Message**

```
Sub GetSQLXERR (hDB As String)

    Dim opt_value As String * 256
    error_no% = ltDBgetopt(hDB, DB_OPT_SQLXERR, opt_value,
Len(opt_value))
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] ltDBgetopt Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
```

```
    Else
        MsgBox (opt_value), , "LeeTech AIM/SDK"
    End If
End Sub
```

Retrieving Data from a Remote Database (ltDBselect)

Description

The *ltDBselect* routine executes an SQL SELECT statement and returns qualified rows. The options that may be used are, open a cursor, fetch rows, close a cursor or use a combination of these three options. This routine also allows skipping or ignoring of rows of data for better control over the selected data. To use this function, the row variable must be set to the number of rows to be skipped.

Multiple Cursors

The LeeTech AIM/SDK supports multiple concurrent cursors. In the client application, more than one cursor can be opened by 1) specifying the option *DB_SELECT_OPEN* for the first *ltDBselect* call, and 2) using the combined option of *DB_SELECT_OPEN* | *DB_SELECT_NEW* for subsequent *ltDBselect* calls.

ltDBgetopt may be used with the *DB_OPT_CURSOR_ID* option to get the current CURSOR_ID and save it to a local variable. Then use *ltDBsetopt* with the *DB_OPT_CURSOR_ID* option to switch from the current cursor to a previously opened cursor by using the saved CURSOR_ID and continue fetching rows from the previously opened cursor (which is actually the current cursor).

Maximum number of concurrent cursors

ORACLE:	16 maximum concurrent cursors.
INFORMIX:	16 maximum concurrent cursors.
ALLBASE/SQL:	16 maximum concurrent cursors.
IMAGE/SQL:	16 maximum concurrent cursors.

Number of rows per transmission

The number of rows per transmission depends upon the size of the *rowbuf.* The LeeTech AIM/SDK server tries to fill the client buffer with as many rows as it can. The number of returned rows (*nrows*) and number of columns per row (*ncols*) are also returned; a row or a column can be easily gotten from the *rowbuf*. A *more* indicator shows whether or not the end of the cursor has been reached.

Syntax	**error = ltDBselect(handle, opt, stmt, ncols, nrows, rowbuf, rowbuf_len, more)**

Parameters

error Returned value; integer (2-byte for 16 bit OS, 4 byte for 32-bit OS)

0 = OK

handle Input (required); returned by *ltDBenter;* character array (256 characters); passing by address/reference

opt Input (required); integer (2-byte for 16 bit OS, 4 byte for 32-bit OS); passing by value

It contains the following options:

Table 3-8. ltDBselect Options

OPT_DESCRIPTION	OPT
DB_SELECT_OPEN This option will open a cursor based upon the SQL SELECT statement that is specified in the *stmt* parameter. If there is no active transaction, it initiates an implicit transaction. An active transaction can be started by an explicit *ltDBbegintx* call, or by the very first *ltDBselect* call, or by a *ltDBselect* call immediately following a completed transaction.	1

Table 3-8. ltDBselect Options (continued)

OPT_DESCRIPTION	OPT
This option does not return any data to the data buffer. The default locking isolation level for an implicit transaction is RR. An implicit transaction remains active until all cursors are closed, or ***when the very first cursor is closed***. Before a cursor can be closed in a multiple-cursor transaction, *ltDBsetopt* must be called and the current cursor set to the target cursor. After this has been done, call *ltDBselect* with the *DB_SELECT_CLOSE* option to close the cursor. *The very first cursor must be kept track of when an implicit transaction is used.* If the first cursor is closed in a multiple-cursor, implicit transaction, all other cursors are also closed when the implicit transaction is closed. Therefore, ***explicit transactions are recommended*** rather than implicit transactions. Please refer to: *ltDBsetopt, ltDBgetopt, ltDBbegintx, ltDBcommittx* or *ltDBrollbacktx.*	1
DB_SELECT_FETCH This option fetches rows from the current cursor. LeeTech AIM/SDK server tries to fit as many rows as it can into the data buffer specified in the client program. If there are more rows that can fit, the *more* flag is turned on (value = 1); otherwise, the *more* flag will be off (value = 0).	2
DB_SELECT_CLOSE This option closes the cursor and optionally issues a "commit work." If this cursor is the very first cursor or the only cursor of the implicit transaction, the active implicit transaction is terminated as well.	4

Table 3-8. ltDBselect Options (continued)

OPT_DESCRIPTION	OPT
DB_SELECT_OPEN and DB_SELECT_FETCH This option opens a cursor and fetches the first batch of rows.	3
DB_SELECT_FETCH and DB_SELECT_CLOSE This option fetches a set of rows, then closes the cursor. The rows fetched are not necessarily the last set; it depends upon the current cursor location.	6
DB_SELECT_OPEN and DB_SELECT_FETCH and DB_SELECT_CLOSE This option opens a cursor, fetches the first batch of rows, then closes the cursor.	7
DB_SELECT_NEW The option opens up a new cursor, keeping all previously opened cursors. The current CURSOR_ID should be saved before this option is used in the *ltDBselect*. Once the new cursor is opened, switch back to the previously opened cursor is not an option without using its CURSOR_ID. Get the CURSOR_ID by calling *ltDBgetopt* with the option of *DB_OPT_CURSOR_ID*. Switch back to an old cursor by calling *ltDBsetopt* with the option of *DB_OPT_CURSOR_ID* and the saved CURSOR_ID. This option must be used in conjunction with the **DB_SELECT_OPEN** option. A call to **DB_SELECT_OPEN** without **DB_SELECT_NEW** option will trigger LeeTech AIM/SDK server to close both the current cursor and	8

Table 3-8. ltDBselect Options (continued)

OPT_DESCRIPTION	OPT
the implicit transaction. This occurs if the current cursor is the very first cursor or the only cursor of the implicit transaction.	
DB_SELECT_SKIP This option ignores (or skips) the rows of data specified by the **nrows** parameter and fetches the rows of greater than **nrows**. This option must be used in conjunction with the **DB_SELECT_FETCH** option.	16

stmt Input (required); character string; passing by address/reference
It contains the SQL SELECT statement.

ncols Returned (required); integer (2-byte for 16 bit OS; 4-byte for 32 bit OS); passing by address/reference
It contains the number of columns per row.

nrows Returned (required); integer (2-byte for 16 bit OS; 4-byte for 32 bit OS); passing by address/reference
It contains the number of rows in the *rowbuf*.
Input (required); It contains the number of rows to skip if DB_SELECT_SKIP is specified.

rowbuf Returned (required); character array (maximum of 30,000 characters); passing by address/reference
It contains the qualified rows for the SQL SELECT statement specified in *stmt*. LeeTech AIM/SDK converts all column values into **ASCII CHARACTER** format, regardless of their database defined types.

The returned data structure is variable. It depends upon the format options setting via *ltDBsetopt* before *ltDBselect* is executed.

FORMAT 1:

Default Format:

1. No *ltDBsetopt* calls prior to an *ltDBselect* call.

2. Each returned column data consists of a length and a value.

3. The length is a 2-byte integer containing the actual number of characters of the column value; it does not include the length of itself (2-byte). A 0 means no data; a value>0 means the actual number of characters of the column value. A -1 means no data and the column contains a NULL value.

FORMAT 2:

DB_OPT_PREFIX_ROWLEN:

1. **ltDBsetopt(~,DB_OPT_PREFIX_ROWLEN,"Y", 1)**

2. Each row begins with a row length. The row length is a 2-byte integer containing the actual number of characters in the row. It includes all column lengths (if applicable) and column values of the row; but it does not include itself (2-byte). A 0 means no data; a value>0 means the actual number of characters of the row.

FORMAT 3:

DB_OPT_LEN_IN_ASCII:

1. **ltDBsetopt(~,DB_OPT_LEN_IN_ASCII,"Y", 1)**

2. Each column consists of a length and a value.

3. The length is a 5-character numeric string containing the actual number of characters in the column value; it does not include the length of itself (5 characters). A "00000" means no data; a value>"00000" means the actual number of characters of the column value; and a "-0001" means no data and the column contains a NULL value.

FORMAT 4:

DB_OPT_LEN_IN_ASCII:
DB_OPT_PREFIX_ROWLEN:

1. **ltDBsetopt(~,DB_OPT_LEN_IN_ASCII,"Y",1)**
   ltDBsetopt(~,DB_OPT_PREFIX_RWOLEN,"Y",1)

2. A row length leads each row. The row length is a 5-character numeric string containing the actual number of characters in the row. It includes all column lengths (if applicable) and column values of the row; but it does not include itself (5 characters). A "00000" means no data, and a value>"00000" means the actual number of characters of the row.

FORMAT 5:

DB_OPT_FIX_FORMAT:

1. **ltDBsetopt(~,DB_OPT_FIX_FORMAT,"Y",1)**

2. There is no column length or row length in the returned rowbuf. All rows are the same length. Each column has the same beginning position and offset in all rows. The *ltDBattr( )* routine returns the row format.

rowbuf_len Input (required); integer (2-byte for a 16 bit OS; 4-byte for a 32 bit OS); passing by value

more Returned (required); integer (2-byte for a 16 bit OS; 4-byte for a 32 bit OS); passing by address/reference

Example

Returned Data Format

- **From file "declare.bas".**

```
Global Const DB_OPT_LEN_IN_ASCII = 3
Global Const DB_OPT_PREFIX_ROWLEN= 4
Global Const DB_OPT_FIX_FORMAT           = 5
Global Const DB_OPT_ISOLATION_LEVEL      = 6
Global Const DB_OPT_CURSOR_ID            = 7
Global Const DB_OPT_AUTO_CLOSE           = 8

Global Const DB_SELECT_OPEN              = &H1
Global Const DB_SELECT_FETCH             = &H2
```

```
Global Const DB_SELECT_CLOSE              = &H4
Global Const DB_SELECT_NEW                = &H8

Global Const DB_CURSOR_ID_LEN             = 8
Global Const DB_ERR_NULL_CURSOR_ID        = 3516

Global hDB As String * 256
Global Const DB_MAX_MSG_LEN               = 8192
Global GBuffer As String * DB_MAX_MSG_LEN

Declare Function ltDBselect% Lib "ltws.dll" (ByVal hDB$, ByVal
SelectOpt%, _
                        ByVal Stmt$, NumOfCol%, NumOfRow%, ByVal
RowBuf$, _
                        ByVal RowBufLen%, More%)
Declare Function  ltDBsetopt% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%, _
                        ByVal Value$, ByVal Value_Len%)
Declare Function  ltDBgetopt% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%, _
                       ByVal Buffer$, ByVal Buf_Len%)
Declare Function  ltDBgetcol% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%, _
                       ByVal ColBuf$, ByVal ColBuf_Len%, RLen%, ByVal
RowBuf$)
Declare Function  ltDBattr% Lib "ltws.dll" (ByVal hDB$, ByVal Stmt$, _
                        NumOfCol%, ByVal Attr$, ByVal Buf_Len%)
Declare Function  ltDBbegintx% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%)
Declare Function  ltDBcommittx% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%)
Declare Function  ltDBleave% Lib "ltws.dll" (ByVal hDB$)
```

- **Get Rows and Columns - default format**

```
Dim stmt As String

error_no% = ltDBselect(hDB, DB_SELECT_OPEN, stmt, ncol%, nrow%, _
                    Gbuffer, DB_MAX_MSG_LEN, more%)
If  (error_no% <> 0) Then
     Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Database Failure"
     MsgBox Msg$, , "LeeTech AIM/SDK"
     GetSQLXERR(hDB)
     GoTo MyLeave
End If

Dim attr As String * 3000

error_no% = ltDBattr(hDB, stmt, ncol%, attr, Len(attr))
If  (error_no% <> 0) Then
     Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Attribute Failure"
     MsgBox Msg$, , "LeeTech AIM/SDK"
     GetSQLXERR(hDB)
     GoTo MyLeave
End If
```

```
temp$ = attr

For i = 1 To ncol% Step 1
    'Retrieve Column Name
    pos% = InStr(1, temp$, Chr(0))
    ColName$ = Mid$(temp$, 1, pos% - 1)
    temp$ = Right$(temp$, Len(temp$) - pos%)
    'Bypass data type
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)
```

```
    'Bypass NULL
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)
Next i

Dim ColBuf As String * 128

Do
    error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%, _
                           Gbuffer, DB_MAX_MSG_LEN, more%)
    If  (error_no% <> 0) Then
         Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
         MsgBox Msg$, , "LeeTech AIM/SDK"
         GetSQLXERR(hDB)
         GoTo MyLeave
    End If

    For row_index = 1 To nrow% Step 1
      For col_index = 1 To ncol% Step 1
        ColBuf = String$(Len(ColBuf), 0)
         error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN,
_
                                rlen%, Gbuffer)
         If  (error_no% <> 0) Then
             Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column
Failure"
             MsgBox Msg$, , "LeeTech AIM/SDK"
             GoTo MyLeave
         End If
        Next col_index
      Next row_index
Loop Until more% = 0

MyLeave:
    error_no% = ltDBleave(hDB)
    If (error_no% <> 0) Then
        Screen.MousePointer = DEFAULT
        Msg$ = "[ERROR-" + Str$(error_no%) + "]"
        MsgBox Msg$, MB_OK, "LeeTech AIM/SDK"
    End If
End
```

- **DB_OPT_FIX_FORMAT**

  quick way for row and column fetching

```
error_no% = ltDBsetopt(hDB, DB_OPT_FIX_FORMAT, "Y", 1)
If  (error_no% <> 0) Then
     Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Fix Format Option
Failure"
     MsgBox Msg$, , "LeeTech AIM/SDK"
     GoTo MyLeave
End If

Dim Offset As Integer
Dim BeginPos As Integer
Dim RowLength As Integer
Dim stmt As String

error_no% = ltDBselect(hDB, DB_SELECT_OPEN, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more%)
If  (error_no% <> 0) Then
     Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Database Failure"
```

```
     MsgBox Msg$, , "LeeTech AIM/SDK"
     GetSQLXERR(hDB)
     GoTo MyLeave
End If

ReDim Offset(1 To ncol%)
ReDim BeginPos(1 To ncol%)

temp$ = attr

For i = 1 To ncol% Step 1
    'Retrieve Column Name
    pos% = InStr(1, temp$, Chr(0))
    ColName$ = Mid$(temp$, 1, pos% - 1)
    temp$ = Right$(temp$, Len(temp$) - pos%)
    'Bypass Data Type
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Bypass NULL Indicator
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Retrieve Beginning Position
    pos% = InStr(1, temp$, Chr(0))
    BeginPos(i) = Val(Mid$(temp$, 1, pos% - 1))
    temp$ = Right$(temp$, Len(temp$) - pos%)
    'Retrieve Offset
     pos% = InStr(1, temp$, Chr(0))
     Offset(i) = Val(Mid$(temp$, 1, pos% - 1))
     temp$ = Right$(temp$, Len(temp$) - pos%)
Next i
RowLength = BeginPos(i - 1) + Offset(i - 1)

Do
    error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%, _
                           Gbuffer, DB_MAX_MSG_LEN, more%)
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
        GetSQLXERR(hDB)
        GoTo MyLeave
    End If

    temp$ = Gbuffer

    For row_index = 1 To nrow% Step 1
      For col_index = 1 To ncol% Step 1
          LT_DATA$ = Mid$(temp$, BeginPos(col_index) + 1,
Offset(col_index))
      Next col_index
      temp$ = Right$(temp$, Len(temp$) - RowLength)
    Next row_index
Loop Until more% = 0

MyLeave:
    error_no% = ltDBleave(hDB)
    If (error_no% <> 0) Then
        Screen.MousePointer = DEFAULT
        Msg$ = "[ERROR-" + Str$(error_no%) + "]"
        MsgBox Msg$, MB_OK, "LeeTech AIM/SDK"
    End If
    End
```

Implicit Transaction:

- **DB_OPT_ISOLATION_LEVEL**
  DB_OPT_AUTO_CLOSE

```
opt$ = "RU"
error_no% = ltDBsetopt(hDB, DB_OPT_ISOLATION_LEVEL, opt$, Len(opt$))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Isolation Level
Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

opt$ = "N"
error_no% = ltDBsetopt(hDB, DB_OPT_AUTO_CLOSE, opt$, Len(opt$))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Auto Close Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
```

- **Open Cursor #1**
  Get #1 CURSOR_ID
  Fetch Rows from Cursor #1

```
Dim stmt As String
Dim ColBuf As String * 128
Dim CursorID1 As String * DB_CURSOR_ID_LEN

error_no% = ltDBselect(hDB, DB_SELECT_OPEN, stmt, ncol%, nrow%,
Gbuffer, _
                              DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Cursor #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If
error_no% = ltDBgetopt(hDB, DB_OPT_CURSOR_ID, CursorID1,
DB_CURSOR_ID_LEN)
If (error_no% <> 0 And error_no% <> DB_ERR_NULL_CURSOR_ID) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Cursor ID #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                         DB_MAX_MSG_LEN, more1%)
If (error_no% <> 0) Then

    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If
```

```
For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN,
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"
          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
    Next col_index
Next row_index
```

- **Open Cursor #2**
  Get #2 CURSOR_ID
  Fetch Rows from Cursor #2

```
Dim CursorID2 As String * DB_CURSOR_ID_LEN
error_no% = ltDBselect(hDB, DB_SELECT_OPEN Or DB_SELECT_NEW, stmt,
ncol%, _
                       nrow%, Gbuffer,      DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Cursor #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

error_no% = ltDBgetopt(hDB, DB_OPT_CURSOR_ID, CursorID2,
DB_CURSOR_ID_LEN)
If (error_no% <> 0 And error_no% <> DB_ERR_NULL_CURSOR_ID) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Cursor ID #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more2%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN, _
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"
          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
    Next col_index
Next row_index
```

- **Switch to Cursor #1**
  Fetch Rows from Cursor #1

```
error_no% = ltDBsetopt(hDB, DB_OPT_CURSOR_ID, CursorID1,
DB_CURSOR_ID_LEN)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Cursor ID #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more1%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN, _
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"
          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
    Next col_index
Next row_index
```

- **Close Cursor #1**
  Terminate whole transaction

```
error_no% = ltDBselect(hDB, DB_SELECT_CLOSE, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Close Cursor #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

MyLeave:
    error_no% = ltDBleave(hDB)
    If (error_no% <> 0) Then
        Screen.MousePointer = DEFAULT
        Msg$ = "[ERROR-" + Str$(error_no%) + "]"
        MsgBox Msg$, MB_OK, "LeeTech AIM/SDK"
    End If
    End
```

Explicit Transaction

- **DB_OPT_ISOLATION_LEVEL**
  DB_OPT_AUTO_CLOSE

```
opt$ = "RU"
error_no% = ltDBsetopt(hDB, DB_OPT_ISOLATION_LEVEL, opt$, Len(opt$))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Isolation Level
Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
opt$ = "N"
error_no% = ltDBsetopt(hDB, DB_OPT_AUTO_CLOSE, opt$, Len(opt$))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Auto Close Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
```

- **Begin Transaction**

```
error_no% = ltDBbegintx(hDB, Nullopt%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Begin Transaction Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
```

- **Open Cursor #1**
  Get #1 CURSOR_ID
  Fetch Rows from Cursor #1

```
Dim stmt As String
Dim ColBuf As String * 128
Dim CursorID1 As String * DB_CURSOR_ID_LEN

error_no% = ltDBselect(hDB, DB_SELECT_OPEN, stmt, ncol%, nrow%,
Gbuffer, _
                    DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Cursor #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

error_no% = ltDBgetopt(hDB, DB_OPT_CURSOR_ID, CursorID1,
DB_CURSOR_ID_LEN)
If (error_no% <> 0 And error_no% <> DB_ERR_NULL_CURSOR_ID) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Cursor ID #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                    DB_MAX_MSG_LEN, more1%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
```

```
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN, _
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"

          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
    Next col_index
Next row_index
```

- **Open Cursor #2**
  Get #2 CURSOR_ID
  Fetch Rows from Cursor #2

```
Dim CursorID2 As String * DB_CURSOR_ID_LEN
error_no% = ltDBselect(hDB, DB_SELECT_OPEN Or DB_SELECT_NEW, stmt,
ncol%, _
                       nrow%, Gbuffer, DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Cursor #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

error_no% = ltDBgetopt(hDB, DB_OPT_CURSOR_ID, CursorID2,
DB_CURSOR_ID_LEN)
If (error_no% <> 0 And error_no% <> DB_ERR_NULL_CURSOR_ID) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Cursor ID #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more2%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN, _
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"
          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
```

```
    Next col_index
Next row_index
```

- **Switch to Cursor #1**
  Fetch Rows from Cursor #1

```
error_no% = ltDBsetopt(hDB, DB_OPT_CURSOR_ID, CursorID1,
DB_CURSOR_ID_LEN)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Cursor ID #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

error_no% = ltDBselect(hDB, DB_SELECT_FETCH, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more1%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Fetch Data Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If

For row_index = 1 To nrow% Step 1
    For col_index = 1 To ncol% Step 1
      ColBuf = String$(Len(ColBuf), 0)
      error_no% = ltDBgetcol(hDB, Nullopt%, ColBuf, DB_MAX_MSG_LEN, _
                             rlen%, Gbuffer)
      If (error_no% <> 0) Then
          Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Column Failure"
          MsgBox Msg$, , "LeeTech AIM/SDK"
          GoTo MyLeave
      End If
    Next col_index
Next row_index
```

- **Close Cursor #1**

```
error_no% = ltDBselect(hDB, DB_SELECT_CLOSE, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN,     more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Close Cursor #1 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
```

```
    GoTo MyLeave
End If
```

- **Switch to Cursor #2**

```
error_no% = ltDBsetopt(hDB, DB_OPT_CURSOR_ID, CursorID2,
DB_CURSOR_ID_LEN)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Cursor ID #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If
```

- **Close Cursor #2**

```
error_no% = ltDBselect(hDB, DB_SELECT_CLOSE, stmt, ncol%, nrow%,
Gbuffer, _
                       DB_MAX_MSG_LEN, more%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Close Cursor #2 Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    GoTo MyLeave
End If
```

- **End Transaction**

```
error_no% = ltDBcommittx(hDB, Nullopt%)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Commit Work Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GoTo MyLeave
End If

MyLeave:
    error_no% = ltDBleave(hDB)
    If (error_no% <> 0) Then
        Screen.MousePointer = DEFAULT
        Msg$ = "[ERROR-" + Str$(error_no%) + "]"
        MsgBox Msg$, MB_OK, "LeeTech AIM/SDK"
    End If
    End
```

Column Attributes (ltDBattr)

Description The *ltDBattr* routine returns the column attributes for a specific SQL SELECT statement. The attributes can then be used with the *ltDBselect* routine for data retrieval. The column attributes are used not only for information, but also quick column value retrieving from a ***fixed length row***. Setting the option DB_OPT_FIX_FORMAT to “Y” in ltDBsetopt call triggers the fixed length row fetching.

Column attributes: each column attribute record is composed of 3 items for ***non fixed length row*** fetching: column name, column type, and NULL indicator. Each record also contains 5 items for ***fixed length row*** fetching: column name, column type, NULL indicator, beginning position and column length.

Syntax	**error = ltDBattr(handle, stmt, ncols, attrbuf, attrbuf_len)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Input (required); returned by *ltDBenter;* character array (256 characters); passing by address/reference

stmt Input (required); character string; passing by address/reference

The character string contains an SQL SELECT statement. SQL Direct SDK always checks the statement syntax first; if the specified SQL SELECT statement has an SQL syntax error, there will be no returned column attributes.

In order to get the fixed format column attributes (including *Beginning Position* and O*ffset*), call the *ltDBsetopt* routine to set DB_OPT_FIX_FORMAT to "Y," before *ltDBattr* is called. The *ltDBsetopt* routine is detailed in this chapter.

ncols Returned (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by address/reference
ncols contains the number of columns for the specified SQL statement.

attrbuf Returned (required); character array (the maximum size is 30,000 characters); passing by address/reference

This parameter contains the column attribute table for the specified SQL SELECT statement. Enough space must be reserved to store the returned column attribute table. The format for each column attribute is:

All attribute items are in ASCII format, which are characters. Convert the column Beginning Position and Length into INTEGER variables before doing any mathematical calculations.

All attribute items end with the NULL character, 0. In C, the last character is a '\0'; in Visual Basic, it is a 'CHR$(0)'; and in COBOL, it is a 'VALUE %0'.

The standard formula to estimate the size of attrbuf:
56 characters * column count (ncols)

attrbuf_len Input (required); integer (2-byte for 16-bit OS, 4-byte for 32-bit OS); passing by value. This parameter is the size of the *attrbuf*. It cannot be greater than 30,000.

Example

Visual Basic

- **From file "declare.bas".**

```
Global Const DB_OPT_FIX_FORMAT = 5
Global hDB  As String * 256
Declare Function ltDBsetopt% Lib "ltws.dll" _
                 (ByVal hDB$, ByVal opt%, ByVal ptr$, ByVal length%)
Declare Function ltDBattr%   Lib "ltws.dll" _
                 (ByVal hDB$, ByVal stmt$, NumOfCol%, ByVal attr$, _
                  ByVal Buf_Len%)
Declare Function ltDBgetopt% Lib "ltws.dll" _
                 (ByVal hDB$, ByVal Opt%, ByVal Buffer$, ByVal
Buf_Len%)
```

- **ltDBattr - default format**

```
Dim stmt As String
Dim attr As String * 3000
```

```
error_no% = ltDBattr(hDB, stmt, ncol%, attr, Len(attr))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Attribute Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
    Exit Sub
End If

temp$ = attr

For i = 1 To ncol% Step 1
    'Retrieve Column Name
    pos% = InStr(1, temp$, Chr(0))
    ColName$ = Mid$(temp$, 1, pos% - 1)
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Bypass data type
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Bypass NULL indicator
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)
Next i
```

- **ltDBsetopt and ltDBattr - fixed format**

```
error_no% = ltDBsetopt(hDB, DB_OPT_FIX_FORMAT, "Y", 1)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Fix Format Option
Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK”
    Exit Sub
End If

Dim stmt As String
Dim attr As String * 3000
Dim BeginPos As Integer
Dim Offset As Integer

error_no% = ltDBattr(hDB, stmt, ncol%, attr, Len(attr))
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Attribute Failure"
    MsgBox Msg$, , "LeeTech AIM/SDK"
    GetSQLXERR(hDB)
```

```
    Exit Sub
End If

ReDim Offset(1 To ncol%)
ReDim BeginPos(1 To ncol%)

temp$ = attr
For i = 1 To ncol% Step 1
    'Retrieve Column Name
    pos% = InStr(1, temp$, Chr(0))
    ColName$ = Mid$(temp$, 1, pos% - 1)
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Bypass Data Type
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Bypass NULL Indicator
    pos% = InStr(1, temp$, Chr(0))
    temp$ = Right$(temp$, Len(temp$) - pos%)

    'Retrieve Beginning Position
    pos% = InStr(1, temp$, Chr(0))
    BeginPos(i) = Val(Mid$(temp$, 1, pos% - 1))
    temp$ = Right$(temp$, Len(temp$) - pos%)
    'Retrieve Offset
    pos% = InStr(1, temp$, Chr(0))
    Offset(i) = Val(Mid$(temp$, 1, pos% - 1))
    temp$ = Right$(temp$, Len(temp$) - pos%)
Next i
```

Retrieving Column Value (ltDBgetcol)

Description

The *ltDBgetcol* routine gets the column value from the returned *ltDBselect rowbuf*, returns the actual column value length, and advances the *rowbuf* internal pointer to the next column.

This routine is **disabled,** if the ***DB_OPT_FIX_FORMAT*** is set to "Y" (via *ltDBsetopt* routine).

Refer to *ltDBselect* section for detail *rowbuf* layout (FIX and VARIABLE formats).

Syntax	**error = ltDBgetcol(handle, opt, colbuf, colbuf_len, colbuf_rlen, rowbuf)**

Parameters

error	Returned value; integer (2-byte for 16 bit OS, 4-byte for 32 bit OS) 0 = OK
handle	Input (required); returned by *ltDBenter;* character string (256 characters); passing by address/reference
opt	Input (required); integer (2-byte for 16 bit OS, 4-byte for 32-bit OS); passing by value Reserved for the future; set to 0
colbuf	Returned (required); character array; passing by address/reference The column value in ASCII format
colbuf_len	Input (required); integer (2-byte); passing by value It is the size of the *colbuf*.
colbuf_rlen	Returned (required); integer (2-byte); passing by address/reference. It contains the actual size of the column value.
rowbuf	Input (required); character string; passing by address/reference It is the returned buffer from *ltDBselect that* contains multiple rows satisfying the SQL SELECT statement selection criteria.

Getting Error Messages (ltDBgetopt)

Description	The *ltDBgetopt* routine gets SQL standard error messages generated by the previous LeeTech AIM/SDK call, or

returns the current *CURSOR_ID* from the previous *ltDBselect* call.

Set the option ***DB_OPT_AUTO_CLOSE*** to "N" before *ltDBselect* is called to ensure that *ltDBgetopt* always returns the *CURSOR_ID* from the previous *ltDBselect* call. (Please refer to *ltDBselect* section for more detail).

Syntax	**error = ltDBgetopt(handle, opt, value, value_len)**

Parameters

error Returned value; integer (2-byte)
0 = OK

handle Input (required); returned by *ltDBenter;* character array (256 characters); passing by address/reference

opt Input (required); integer (2-byte for 16 bit OS, 4-byte for 32 bit OS); passing by value.

It contains the following options:

Table 3-9. ltDBgetopt Options

OPT_DESCRIPTION	OPT
DB_OPT_SQLXERR: This option gets the SQL standard error messages and returns them into the designated value buffer. The maximum error message buffer size is 3096 characters.	1
DB_OPT_CURSOR_ID: This option gets the current CURSOR-ID in ASCII format with 8 fixed characters.	7

value Returned (required); character string; passing by address/reference

It contains the actual setting for the specific option.

Table 3-10. ltDBgetopt Actual Value Settings

OPT	VALUE
DB_OPT_SQLXERR	Data buffer variable; it contains the SQL standard error messages in ASCII format. The maximum size is 3096 characters.
DB_OPT_CURSOR_ID	Data buffer variable; it must be 8 characters long, and the *value_len* must be equal to 8.

value_len Input (required); integer (2-byte); passing by value.

It is the size of the *value*.

Example

Visual Basic

- **From file "declare.bas".**

```
Global Const DB_OPT_SQLXERR        = 1
Global Const DB_OPT_CURSOR_ID      = 7
Global Const DB_OPT_AUTO_CLOSE     = 8
Global Const DB_CURSOR_ID_LEN      = 8
Global Const DB_ERR_NULL_CURSOR_ID = 3516
Global Const DB_SELECT_OPEN        = &H1
Global hDB As String * 256
Global Const DB_MAX_MSG_LEN = 8192
Global GBuffer As String * DB_MAX_MSG_LEN

Declare Function ltDBgetopt% Lib "ltws.dll" (ByVal hDB$, ByVal Opt%, _
                                             ByVal Buffer$, ByVal
Buf_Len%)
```

- **ltDBgetopt - get SQL Error Message**

```
Sub GetSQLXERR (hDB As String)

    Dim opt_value As String * 256

    error_no% = ltDBgetopt(hDB, DB_OPT_SQLXERR, opt_value,
Len(opt_value))
    If (error_no% <> 0) Then
```

```
        Msg$ = "[ERROR-" + Str$(error_no%) + "] ltDBgetopt Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
    Else
        MsgBox (opt_value), , "LeeTech AIM/SDK"
    End If

End Sub
```

- **ltDBgetopt - get CURSOR-ID**

```
Sub Open_Cursor

    opt$ = "N"
    error_no% = ltDBsetopt(hDB, DB_OPT_AUTO_CLOSE, opt$, Len(opt$))
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Set Auto Close Failure"
        MsgBox Msg$, , "LeeTech AIM/SDK"
        Exit Sub
    End If

    Dim stmt As String

    error_no% = ltDBselect(hDB, DB_SELECT_OPEN, stmt, ncol%, nrow%,
Gbuffer _
                        DB_MAX_MSG_LEN, MORE%)
    If (error_no% <> 0) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Open Cursor Failure"
        MsgBox Msg$, , "SQL Direct  Gateway"
        Get SQLWERR(hDB)
        Exit Sub
    End If

    error_no% = ltDBgetopt(hDB, DB_CURSOR_ID, CursorID,
DB_CURSOR_ID_LEN)
    If (error_no% <> And error_no% <> DB_ERR_NULL_CURSOR_ID) Then
        Msg$ = "[ERROR-" + Str$(error_no%) + "] Get Cursor ID Failure"
        MsgBox Msg$, , "SQL Direct  Gateway"
        Exit Sub
    End If

End Sub
```

Concluding Transactions (ltDBcommittx)

Description The *ltDBcommittx* routine completes a transaction and updates the database. All SQL actions for a transaction are permanently committed to the database. Any database locks are also released. This works the same for both explicit and implicit transactions.

Syntax	**error = ltDBcommittx(handle, opt)**

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Input (required); returned by the ltDBenter ( ); character array (256 characters); passing by address/reference

opt Reserved; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

Canceling Transactions (ltDBrollbacktx)

Description The *ltDBrollbacktx* routine cancels a transaction. All SQL actions for a transaction are canceled and there will be no updates to the database. Any database locks are released with this routine. It works the same for both explicit and implicit transactions.

Syntax	**error = ltDBrollbacktx(handle, opt)**

Parameters

error Returned value; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

0 = OK

handle Input (required); by the ltDBenter ( ); returned; character array (256 characters); passing by address/reference

opt Reserved; integer (2-byte for 16-bit OS, 4-byte for 32-bit OS)

Terminating Connections (ltDBleave)

Description The *ltDBleave* routine terminates the network connection from the client to the server that was established by the *ltDBenter* routine. It releases the connected database and ends the LeeTech AIM/SDK server program.

Syntax	error = ltDBleave(handle)

Parameters

error Returned value; integer (2-byte)
0 = OK

handle Input (required); returned by ItDBenter; character array (256 characters); passing by address/reference

In multiple *ltDBenter* calls programs, it is the user's responsibility to close the matching one by using the correct *handle*.

Executing a User-Defined Function Remotely (ltDBusertx)

Description This routine remotely invokes user-defined server routine and communicates with single request and reply. The user-defined routines can be developed in any 3GL or 4GL, they either directly link with LeeTech/SDK provided main driver program or are dynamically loaded from XL during execution time. The user-defined routine are completely independent to SQL database, they may access any database (such as TURBOIMAGE), KSAM file, flat file, ... etc.

Syntax	error = ltDBusertx(handle, txid, request, request_len, reply, reply_len)

Parameters

error Returned value; integer (2-byte)

0 = OK

handle Returned by ltDBenter(); character string (256 characters); passing by address/reference

txid input (required); integer (2-byte); passing by value

A user-defined transaction id which is mapped to a server subroutine.

request input (required); character string; passing by address/reference

It contains the user-defined request data.

request_len input (required); integer (2-byte); passing by value

It is the size of the *request*.

reply returned (required); character string; passing by address/reference

It contains the server reply data.

reply_len input (required); integer (2-byte); passing by address/reference

It is the size of the *reply*.

Example

Visual Basic

- **From file "declare.bas".**

```
Global Const DBSVR_TXID_ECHO = 10001
Global Const DBSVR_TXID_TIME = 10002
Global hDB  As String * 256

Declare Function ltDBusertx% Lib "ltws.dll" (ByVal hDB$, ByVal txid%,
                                             ByVal req$, ByVal
                                             Req_Len%, ByVal Reply$,
                                             Reply_Len%)
```

- **Trigger an User-Defined Transaction**

```
error_no% = ExecUSER(hDB, DBSVR_TXID_ECHO, "123456789")
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Echo Failure"
    MsgBox Msg$, , "LeeTech/SDK"
    Exit Sub
End If

error_no% = ExecUSER(hDB, DBSVR_TXID_TIME, req$)
If (error_no% <> 0) Then
    Msg$ = "[ERROR-" + Str$(error_no%) + "] Return Time Failure"
```

```
    MsgBox Msg$, , "LeeTech/SDK"
    Exit Sub
End If

Function ExecUSER (hDB As String, txid As Integer, req As String) As
        Integer

    Dim rep As String * 128

    error_no% = ltDBusertx(hDB, txid, req, Len(req), rep, Len(rep))
    If (error_no% <> 0) Then
       ExecUSER = error_no%
       Exit Function
    End If

    MsgBox rep
    ExecUSER = error_no%

End Function
```

Note: New routines can be created by the user, but this function comes with preset routines: Echo & Time, Echo

Client API Error Messages

DB_ERR_INVALID_TYPE **1001**

Cause	The operating system and database type specified in the ltDBenter() is invalid.
Action	Correct it and retry ltDBenter().

DB_ERR_INVALID_OPT **1002**

Cause	The option, option value buffer or option value buffer length specified in ltDBgetopt(), ltDBsetopt() are invalid. The ltDBgetcol() does not support FIX_FORMAT row data.
Action	Correct it and retry the call.

DB_ERR_INFO_TOO_LARGE **1003**

Cause	The *info* string specified in ltDBenter() is bigger than 35 characters.
Action	Make it shorter and retry the call.

DB_ERR_INVALID_LOGON **1004**

Cause	The MPE/iX *logon* string in ltDBenter() must be either 48 characters long exact or an empty string (length = 0). The empty logon string means no OS switch logon will be performed, the server process will inherit listener's logon.
Action	Correct it and retry the call.

DB_ERR_DBNAME_TOO_LARGE **1005**

Cause	The database name in ltDBenter() is too long. The maximum database name is 128 characters.
Action	Correct it and retry the call.

DB_ERR_BUFFER_TOO_LARGE **1501**

Cause	The maximum data buffer size that can be specified in ltDBattr(), ltDBgetcol(), ltDBselect() or ltDBsetopt() is 30000.
Action	Correct it and retry the call.

DB_ERR_BUFFER_NOT_DEFINED **1502**

Cause	The *data buffer* used in ltDBattr(), ltDBgetcol() or

ltDBselect() has not been defined.

Action Declare or define the buffer and retry the call.

DB_ERR_BUFFER_TOO_SMALL 1503

Cause The specified buffer is too small to hold the returned data in ltDBattr(), ltDBexec() or ltDBselect().

Action Make it bigger enough and retry the call.

DB_ERR_XCONNECT_FAIL 2001

Cause Network connect failed. (1) An invalid host name was specified or the specified host name was not defined properly in the local *hosts* file; (2) An invalid port_id was specified, or the specified port_id was not defined properly in the local *services* file; (3) the server listener which listens to the specified port_id is not up and running or the server is simply not up.

Action Correct it and retry the ltDBenter().

DB_ERR_SENDMSG1_FAIL 2002

Cause Network connection was dropped while sending data to the server. The server program (DBSVR) may be terminated abnormally or the network has been shut down. Logon to the server, run LANUTIL program

to verify the existence or the status of the corresponding server program (DBSVR).

Action — Call ltDBleave() to clean-up first, then reconnect to the server by calling ltDBenter().

DB_ERR_SENDMSG2_FAIL **2003**

Cause — Network connection was dropped while sending data to the server. The server program (DBSVR) may be terminated abnormally or the network has been shut down. Logon to the server, run LANUTIL program to verify the existence or the status of the corresponding server program (DBSVR).

Action — Call ltDBleave() to clean-up first, then reconnect to the server by calling ltDBenter().

DB_ERR_RECVMSG1_FAIL **2004**

Cause — Network connection was dropped while sending data to the server. The server program (DBSVR) may be terminated abnormally or the network has been shut down. Logon to the server, run LANUTIL program to verify the existence or the status of the corresponding server program (DBSVR).

Action — Call ltDBleave() to clean-up first, then reconnect to the server by calling ltDBenter().

DB_ERR_RECVMSG2_FAIL **2005**

	Cause	Network connection was dropped while sending data to the server. The server program (DBSVR) may be terminated abnormally or the network has been shut down. Logon to the server, run LANUTIL program to verify the existence or the status of the corresponding server program (DBSVR).
	Action	Call ltDBleave() to clean-up first, then reconnect to the server by calling ltDBenter().

DB_ERR_SENDMSG_TOO_LARGE **2006**

	Cause	The maximum size of send data buffer is 30000. This is general message for 1) the SQL statement buffer contains more than 30000 characters, 2) the data buffer contains more than 30000 characters or 3) the buffer length contains value which is greater than 30000.
	Action	Make sure the buffer or buffer length value is appropriate set and retry the call.

DB_ERR_INVALID_TXID **2501**

	Cause	The server program version is older than the client DLL version is.
	Action	Make sure the client/server environment is in-sync.

DB_ERR_BIND_FAIL **3001**

Cause	Make sure the LOOPBACK is up and running.
Action	On MPE/iX, use NETCONTROL STATUS to verify whether the LOOPBACK is running or not, and use NETCONTROL NET=LOOP;START to start it up.

DB_ERR_BOUND_ALREADY **3002**

Cause	LeeTech/SDK internal error.
Action	Please contact LeeTech.

DB_ERR_LOGON_FAIL **3003**

Cause	The logon string contains invalid user or passwords. For MPE/iX, all passwords must be in upper cases.
Action	Make sure valid logon and password are used. Call ltDBleave() to clean-up first, then retry ltDBenter().

DB_ERR_CONNECT_FAIL **3501**

Cause	Connect to database failed.
Action	Make sure the existence of the database and make sure that the specified logon pertains the authority to access the specified database. Call ltDBgetopt() for detail SQL error message.

DB_ERR_CONNECTED_ALREADY **3502**

Cause	LeeTech/SDK internal error.

Action	Please contact LeeTech.

DB_ERR_NOT_CONNECTED **3503**

Cause	Call ltDBenter() to connect to the database first.
Action	Call ltDBenter() to connect to the database that is to be accessed.

DB_ERR_SQLX_NOT_ACTIVATED **3504**

Cause	Call ltDBenter() to connect to the database first.
Action	Call ltDBenter() to connect to the database that is to be accessed.

DB_ERR_BEGIN_WORK_FAIL **3505**

Cause	The logon may not have the authority to perform this command.
Action	Call the DBA to be granted the appropriate access authority. Call ltDBgetopt() for detail SQL error message.

DB_ERR_ATTR_FAIL **3508**

Cause	The logon may not have the authority to perform this command.
Action	Call the DBA to grant appropriate access authority. Call ltDBgetopt() for detail SQL error message.

DB_ERR_SELECT_OPEN_FAIL **3509**

Cause	Open cursor failed. The SQL statement specified in ltDBselect() is invalid.
Action	Call ltDBgetopt() for detail SQL error message.

DB_ERR_SELECT_FETCH_FAIL **3510**

Cause	Fetch data failed.
Action	Call ltDBgetopt() for detail SQL error message.

DB_ERR_SELECT_NOT_ACTIVATED **3512**

Cause	The cursor must be opened before data can be fetched from it.
Action	Call ltDBselect() with DB_SELECT_OPEN option to open the cursor first.

DB_ERR_EXEC_IMMEDIATE_FAIL **3513**

Cause	Invalid SQL statement specified in ltDBexec().
Action	Call ltDBgetopt() for detail SQL error message.

DB_ERR_TXN_STARTED_ALREADY **3514**

Cause	Calling ltDBbegintx() to initiate another transaction when another transaction is still acting is not allowed. Multiple transactions are not supported.
Action	Call ltDBcommittx() or ltDBrollbacktx() to end the acting transaction then call ltDBbegintx(). Call

		ltDBgetopt() for detail SQL error message.

DB_ERR_TXN_NOT_STARTED **3515**

Cause	There is no acting transaction to be either committed or rollbacked.
Action	Ignore it or find out why the previous transaction was closed. Call ltDBgetopt() for detail SQL error message.

DB_ERR_NULL_CURSOR_ID **3516**

Cause	There is no open cursor.
Action	N/A

DB_ERR_OUT_OF_CURSOR_ID **3517**

Cause	The maximum number of concurrent cursors are 16.
Action	Make sure this limit is not exceeded.

DB_ERR_INVALID_CURSOR_ID **3518**

Cause The cursor id is invalid. The specified cursor has been closed. Remember, the ltDBcommittx() and ltDBrollbacktx() close all cursors as well.

4. Overview

StandBy is a new innovative feature of the LeeTech AIM product family. It provides the user with the capability to pre-load or pre-start the server programs, which will tremendously reduce the overhead of traditional client/server process creation, database connection and initialization.

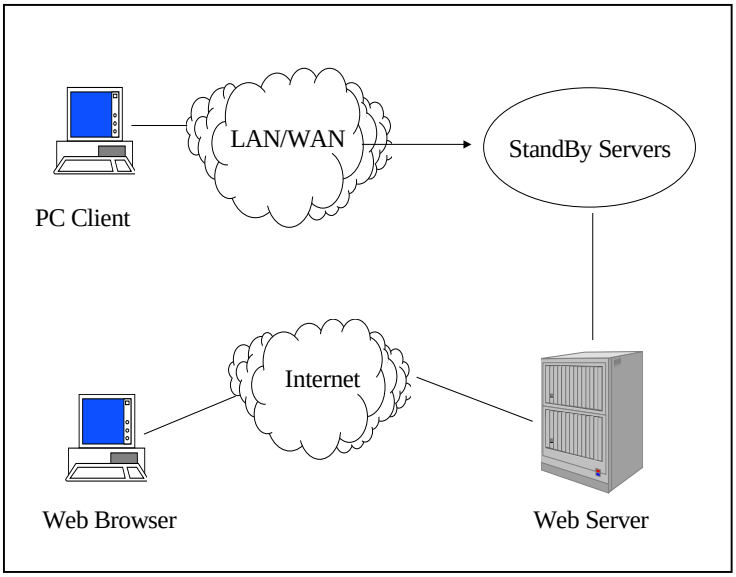

Figure 4-1. Overview of LeeTech AIM/StandBy

Benefits

There are many benefits to applying StandBy technology into variant client/server applications that require high performance and short turn-around time.

Web Type Transaction

Based on the characteristics of the web, the data transaction type is short and high volume. In order to furnish a web page, multiple transactions may be fired to retrieve data. The hit rate may vary tremendously during a day's operation. StandBy is the ultimate selection for this varied application.

Multiple-Tier Client/Server Application

For a large scale of client/server environments, the system may become more complex than originally planned. The number of users may increase unexpectedly. Multiple-Tier client/server architecture provides the capability of partitioning the business logic and partitioning the physical database among the distributed hosts. StandBy provides the seamless connection between host after host. It gives the system engineers the freedom to design a sophisticated system without worrying about the performance.

Multithread Feature with Conventional Single Thread Programming

With StandBy, the system is holding an arsenal of server programs to serve the client's demands. There is no need to complicate the programming in the way of multithread style. Especially when there are tons of lines of existing code. There is no need to worry about the global variable, shared memory or locking semaphore.

System Architecture

One LeeTech AIM listener can monitor multiple StandBy queues. Each StandBy queue is a dynamic pool of pre-loaded server programs waiting for client's requests. The system administrator can configure and launch multiple listeners for load balance purpose.

When the LeeTech AIM listener is launched, it reads the StandBy information from the configuration file. Then the listener fires the StandBy server programs. The requests from the clients still go to the listener first. The listener will do the load balance to dispatch the request to any available server programs. If no sever program is available at that point, the listener will queue the request until there is a free resource.

How to Turn on the StandBy

StandBy Server Program

Modify the server program in a continuous loop to receive the request from the clients. Recall the basic LT/CSF server program coding, lt_msg_enter(), lt_msg_recv(), lt_msg_send(), data transaction back and forth, lt_msg_leave() and then exit the program. Instead of terminating the

process, the flow goes back to a loop of calling lt_msg_enter() again and stay there waiting for the next request.

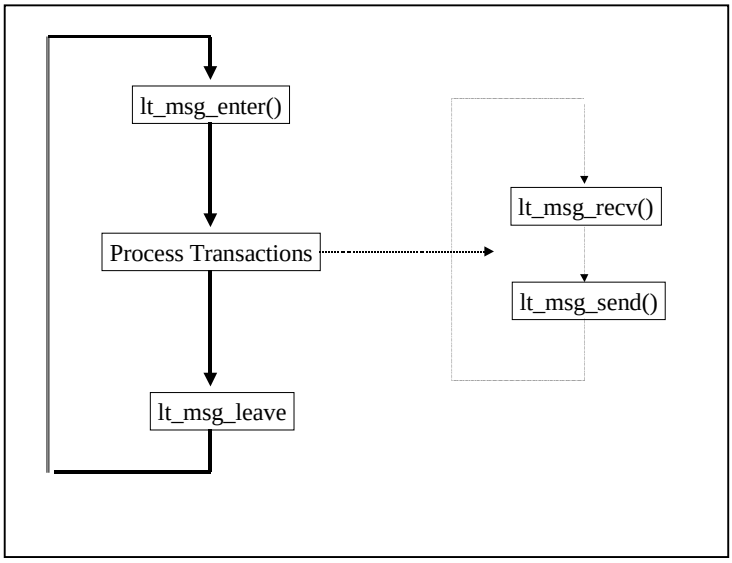

Figure 4-2. Flow of StandBy Server Program

StandBy Listener

To turn on the StandBy feature of the ordinary LeeTech AIM listener should be quite simple. In UNIX and HP MPE/iX server platform, instead of specifying the port number after the listener command, the user only needs to append the StandBy configuration file name leading with a '^' after the listener command.

Example

UNIX

```
/opt/leetech/lbin/listner ^/opt/leetech/lbin/myconfig
```

MPE/iX

```
LISTNER.PUB.LEETECH ^MYCONFIG.PUB.LEETECH
```

The configuration file is a plain text file with the following tags and format.

```
[ GLOBAL ]
SERVICE =
HOME =
DBGOUT =
DBGMASK = 60000000 APP1 + APP2
SERVER =
PARM =
STANDBY = MYQ1 MYQ2

# comment
[MYQ1]
SERVER = APPSVR.PUB.DEMO
MIN = 10
MAX = 100

[MYQ2]
SERVER = /LeeTech/Demo/appsvr
MIN = 20
MAX = 40
```

Keyword

Service The TCP/IP service port that the listener is waiting on.

Home The home directory the running listener process. It is used in most UNIX system where the current working directory of the daemon process is changed to root "/" as system default.

DBGOut The path of log file name to store the debugging or log messages.

DBGMask A binary number in octuple format to hold 32 bit log indicators. There are total 31 log levels that user can check at run time. The order is from left to right. For example, level 1 is noted as 40000000, and level 31 is noted as 00000001. Use logic OR to combine multiple level log.

Server Default server program name if the client makes a connect request without does not specifying a server program name.

Parm The Parm parameter for HP MPE/iX server program only.

StandBy List of StandBy queue's name, separated by space

[Q Name]: StandBy queue section

Server	The server program name to be pre-started
Min	Initial and minimum server processes to start
Max	Maximum server processes that the listener can launch for this queue.

StandBy for Windows NT Server

For the Windows NT server platform users, LeeTech AIM provides a graphic user interface program, LeeTech AIM Service Manager, to configure and maintain the listeners and StandBy queues.

For more information on LeeTech AIM Service Manager see Appendix B.

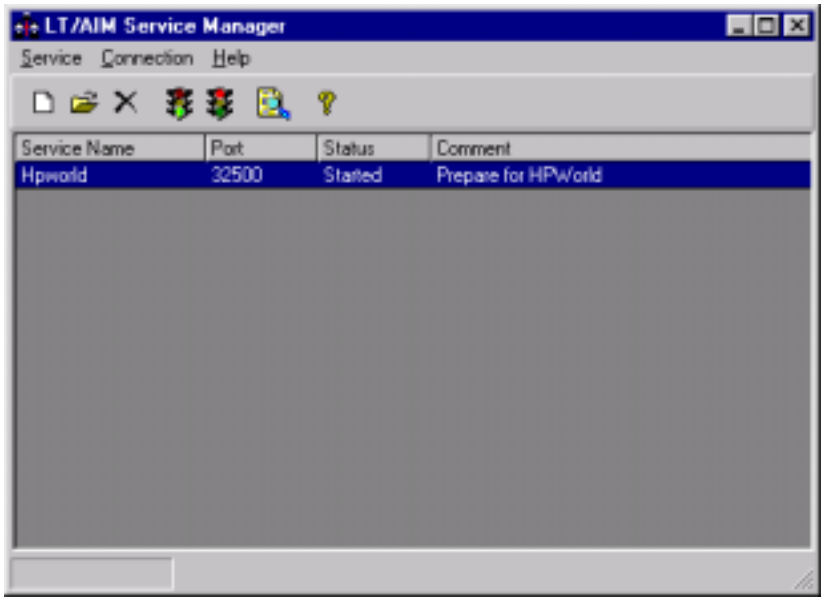

Figure 4-3. Opening Screen of LT/AIM Service Manager

How to Configure a StandBy Queue

1. From the opening screen, click on the “New Service” icon in the toolbar, or click on “Service,” and click on “New” in the menu bar.

   The following screen will appear:

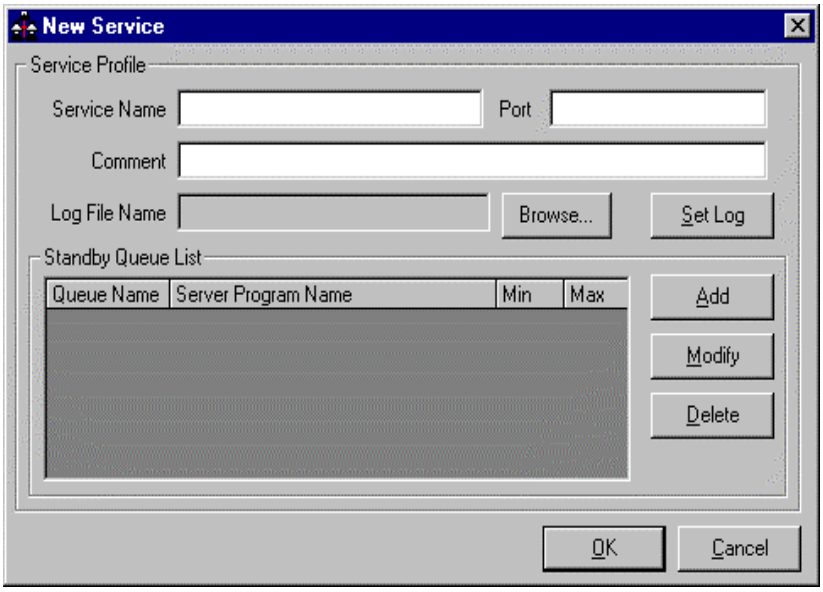

Figure 4-4. LeeTech AIM NT Service Manager “New Service” Screen

2. The user must fill out the Service Profile section of the “New Service” window, prior to filling out the “StandBy queue list” section.

3. In the “New Service” screen, click on the “add” button.

4. Complete the following fields and press OK.

Queue Name	This can be any name without a space.
Server Program Name	The server program file name.
Min	This is the minimum or initial number of server processes that the listener will launch at any given time.

Max This is the maximum number of server processes that the listener will open at any given time.

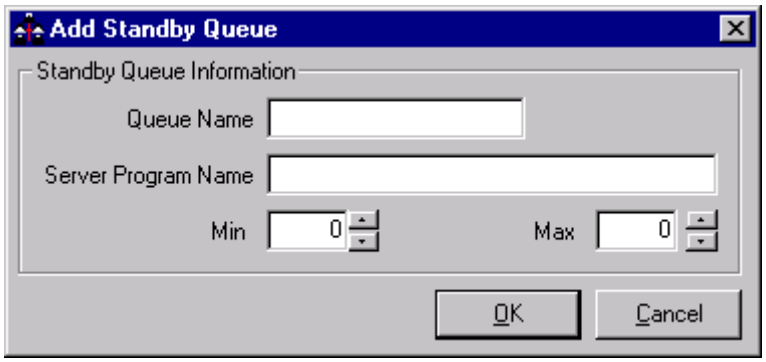

Figure 4-5. Add StandBy Queue Window

How to Modify a StandBy Queue

Stop the service that is to be modified by highlighting the service to be modified, and clicking on the "Stop" icon in the tool bar.

Click on the "modify" button from StandBy Queue List.

The screen which will appear looks exactly like the "Add StandBy Queue" window, except it says "Modify" instead of "Add".

Complete the modifications, and click the OK button.

How to Delete a StandBy Queue

1. Stop the program that is to be deleted, by highlighting the desired process, and clicking on the "Stop" icon in the tool bar, or click on "Service" and "Delete" in the menu bar.

2. Click on the "Delete" button from StandBy Queue List. This will delete the desired process.

Sample Server Programs

COBOL

```
4.
5.       STANDBY-LOOP.
6.          MOVE "STANDBY server"  TO RUN-ID.
7.          STRING RUN-ID          DELIMITED BY "  "
8.                 NULL-TERMINATOR DELIMITED BY SIZE
9.                 INTO LT-DATA-BUFFER.
10.         MOVE 0 TO LT-ERROR.
11.         CALL "lt_msg_enter" USING LT-DATA-BUFFER
12.                                   LT-ERROR.
13.         IF LT-ERROR NOT = 0
14.            IF LT-ERROR = 9999
15.               MOVE LOW-VALUES TO LT-DEBUG
16.               STRING "*** ERROR - Too many connections."
17.                                             DELIMITED BY SIZE
18.                  LT-NEXT-LINE               DELIMITED BY SIZE
19.                  LT-NULL-TERMINATOR         DELIMITED BY SIZE
20.                  INTO LT-DEBUG
21.               CALL "ERROR" USING LT-DEBUG.
22.            ELSE IF LT-ERROR = 9004
23.               “Listener shutdowns the server program”
24.               GO TO END-LOOP.
25.         ELSE
26.               MOVE LOW-VALUES TO LT-DEBUG
27.               STRING "*** ERROR - connection failed."
28.                                             DELIMITED BY SIZE
29.                  LT-NEXT-LINE               DELIMITED BY SIZE
30.                  LT-NULL-TERMINATOR         DELIMITED BY SIZE
31.                  INTO LT-DEBUG
32.               CALL "ERROR" USING LT-DEBUG
33.               GO TO END-LOOP.
34.         END-IF
35.
36.            “Do some data transactions “
37.            …
38.            MOVE LOW-VALUES TO LT-DATA-BUFFER.
39.            MOVE 20         TO LT-DATA-LENGTH.
40.            MOVE 0          TO LT-ERROR.
41.            CALL "lt_msg_recv" USING  LT-DATA-BUFFER
42.                                      \LT-DATA-LENGTH\
43.                                       LT-ERROR.
44.            …
45.
46.            CALL "lt_msg_leave".
47.            GO TO STANDBY-LOOP.
48.
49.      END-LOOP.
50.               STOP RUN.
```

C

```
main()
{
        int     rc;
        char    databuf[2000];
        int     buflen = 2000;

        While(1)
```

```
        {
        lt_msg_enter("StandBy server", &rc);
        if( rc != 0 )
        {
                if( rc == 9999 )
                        error("Too many connections\n");
                else if( rc == 9004 )
                        /* Listener shutdown */
                else
                        error("lt_msg_enter() failed\n");
                break;
        }

                lt_msg_recv(databuf, buflen, &rc);

                /* process the transaction request */
                …
                …

                lt_msg_leave();
        }

        exit(0);
}
```

A. Installation

LeeTech AIM is supplied in two parts, a client and a server component. The client software is available on 3.5 inch high density floppy disks or CD-ROMs suitable for installation on a standard PC. The server software is on a DAT tape. The client and server programs are installed separately. The server software follows generally accepted conventions for server installations (such as, UNIX, IBM/CICS, HP MPE/iX). The client software follows conventions for product installations in Windows. Installation programs are provided for convenience.

LeeTech AIM is to be installed on a local area network. It is based on the TCP/IP protocol and uses Windows socket (winsock.dll).

Installing the Server

Installing on MPE/iX:

1. Log on as system manager:
   :MANAGER.SYS.
2. Restore the tape as follows:

   :File SV;Dev=Dat or Device Number
   :Restore *SV;@.@.LEETECH;CREATE
3. After the tape has been restored, execute the command file:
   :LTINST.PUB.LEETECH.

   This sets the proper security for the account and groups. This command will also prompt for permission to stream some standard Listener jobs used to demonstrate the product.
4. Modify the SERVICES.PUB.LEETECH file to include these lines:

   CSF 32606/tcp csf
   DBA 32600/tcp dba

   These are the port numbers that the demo programs use.

LeeTech account structure:

1. Account access security required:
   (R,X:ANY;W,A,L:AC)
2. Account capability required:
   AM,AL,GL,ND,SF,BA,IA,MR,DS,PH,PM
3. PUB group access security required:
   (R,X:ANY;W,A,L,S:AC)
4. PUB group capability required:
   BA,IA,MR,DS,PH,PM

Starting the MPE/iX Server:

The listener is a server daemon that should be launched before any client-server connection. To start the listener, refer to the listener appendix.

Installing on HP-UX:

1. "Root" authority is necessary to restore the server files with correct attributes.
2. Mount the AIM tape and place the drive online.
3. Restore the AIM files INSTALL and LEETECH.0:
   $tar xvf /dev/rmt/0m (device name)
4. Run INSTALL. It will create the following directories if they do not already exist:

   $/opt/leetech/bin
   $/opt/leetech/demo
   $/opt/leetech/lbin
   $/opt/leetech/lib
   $/opt/leetech/share
   $/opt/leetech/share/include
   $/opt/leetech/newconfig
   $/etc/opt/leetech
   $/var/opt/leetech
5. Delete the files INSTALL and LEETECH.0 upon successful completion of the AIM installation:
   :rm INSTALL LEETECH.0

   The installed files are:

$/opt/leetech/bin/lanutil
$/opt/leetech/bin/ltupgrad
$/opt/leetech/bin/ltvalida
$/opt/leetech/newconfig/CSF
$/opt/leetech/demo/CSFserverC
$/opt/leetech/demo/CSFserverC.c
$/opt/leetech/demo/CSFserverCOBOL.cbl
$/opt/leetech/demo/CUSTDTA
$/opt/leetech/demo/MAKE.csf.demo
$/opt/leetech/lbin/lanmon
$/opt/leetech/lbin/listner
$/opt/leetech/lib/libcsf.a
$/etc/opt/ leetech /CNTLCSF.PUB.STARVSN

Starting the HP-UX Server:

The listener is a server daemon that should be launched before any client-server connection. To start the listener, refer to appendix B.

Installing on IBM/CICS:

LeeTech for the IBM/CICS is supplied in two parts. The client software is available on 3.5 inch high-density floppy disks or CD-ROMs suitable for installation on a standard PC. The IBM/CICS server program source codes are provided in standard COBOL format files. The LeeTech client and server are installed separately. The server installation simply copies server program source codes from a 3.5 inch high-density floppy disk or CD-ROM to the designated IBM mainframe server CICS region via the standard ***ftp*** utility. The client installation follows conventions for product installations in PC Windows. Installation programs are provided for convenience.

LeeTech is to be installed on a local area network. It is based on TCP/IP protocol and uses Windows socket (winsock.dll).

1. Load the source program EAZCICSE and LTPGM01.
2. Compile and bind the program, EAZCICSE: the security exit routine for IBM/CICS Listener.
3. Compile and bind the program, LTPGM01: the generic TCP/IP socket server program for IBM/CICS Listener.
4. Setup a reserved TXID (transaction code) for LeeTech, such as LT01, which must be setup with appropriate capabilities to allow LeeTech to

perform TSQ operations. The reserved TXID will be used by default. if no personal TXID is chosen for the application transaction. It is used when the PC client issues *ltCICSenter* and *ltCICSexec* calls.

5. Setup a reserved TERMID (terminal ID) for LeeTech, such as TCP01. It must be given appropriate capabilities to allow verification of CICS security (SIGNON/SIGNOFF). It is used when the PC client issues a *ltCICSsecurity* call.
6. Start the CICS region with IBM/CICS TCP/IP Listener.

Installing on NT Server:

1. Insert the media into the appropriate drive for installation.
2. Click on the SETUP.EXE command, and follow the directions on the screen
3. When setup is completed, click on the NT Service Manager icon, and the opening screen should appear.
4. To work with NT Service Manager, refer to the Service Manager section.

Installing the Client

LeeTech AIM contains a DLL and a set of sample .EXE programs. The client should be installed on each PC that will access servers.

System Requirements:

IBM or compatible PC
80386 or above
Hard drive with at least 3 megabytes free
VGA or SVGA monitor
Mouse
Windows 95/98/NT

1. If not already running, load Windows.
2. Insert the AIM client media into the appropriate drive for installation.

3. Window 3.1 procedure: choose **File**, **Run** from the Program Manager main menu.
4. Windows 95/98/NT procedure: click on SETUP.EXE and follow the instructions on the screen.
5. Type **Setup** in the command line text box.

Configuring the LAN

Please consult a network system administrator for this section.

SERVICES file

1. Find the SERVICES file on the PC, and insert the line

   CSF 32606/tcp csf
   DBA 32600/tcp dba

CSF and DBA are the application names or port names issued by the sample program. The "32606/tcp" portion of the entry is the port ID, which must be identical to the server's SERVICES file. It is the common name used to link the PC client program and the host server program. The "csf" portion of the entry is another alias of CSF and 32606/tcp. The "dba" portion of the entry is another alias of DBA and 32600/tcp.

2. The SERVICES file on the PC is located in:

   Win 95/98 \windows\
   Win NT \winnt\system32\drivers\etc\

HOSTS file

3. Find the HOSTS file on the PC and insert the host IP Address and its alias (node) name. Use PING.EXE to verify that the host IP address is correct.
4. The HOSTS file on the PC is located in:

   Win 95/98 \windows\
   Win NT \winnt\system32\drivers\etc\

5. The socket driver must exist. Make sure the winsock.dll file is accessible. It must either be in the \windows directory or in a directory listed by the path variable.

B. LeeTech AIM Listener

MPE/iX Listener - LISTNER.PUB.LEETECH

To start a listener: STREAM JLTLSTN.PUB.LEETECH

LISTNER: Listener program.
LTLSTNS: The command file to start the listener program.
JLTLSTN: The job stream file to activate the listener command file.

The listener program (LISTNER) accesses 2 files, HOSTS and SERVICES, which must be setup correctly before streaming the listener job (JLTLSTN).

The HOSTS.NET.SYS file name is used by default. It is possible to create a personal HOSTS file and use a file equation to redirect HOSTS.NET.SYS to the new HOSTS file. The HOSTS file contains those node names that the application can access programmatically. For LeeTech, a "127.0.0.1 localhost loopback" line is required in the HOSTS file.

The SERVICES.NET.SYS file name is used by default. It is possible to create a personal SERVICES file and use a file equation to redirect SERVICES.NET.SYS to the new SERVICES file. The SERVICES file contains service names and associated port_ids, which must be a unique value to the specific HP 3000 machine and is used to establish the initial network connection between CLIENT and SERVER. Different HP 3000 machines can use the same service name and port_id. On the network, the machine node name and port_id are treated as one combined identifier. To avoid any conflict with the HP's reserved port_ids, it is recommended that port_ids greater than 32000 be used. The range for valid port_ids is from 1025 to 32767.

The listener must be started before any client connections are attempted. Multiple listeners may be started to ease the load of a single listener. Each listener must be "listening" to a different service/PORT_ID.

UNIX Listener - /opt/leetech /lbin/listner

Start the listener from the root user login. To start a listener: /opt/leetech/lbin/listner LeeTech "port_name"

LISTNER: - /opt/leetech/lbin/listner

HOSTS: - /etc/hosts
SERVICES: - /etc/services

Windows NT Listener

LeeTech AIM Service Manager will allow the user to install multiple LeeTech AIM listener programs from a GUI. It will also allow the user to have a standby queue of services, modify existing services, and delete unwanted services. The following figure is the opening screen of LeeTech AIM service manager when the user runs the Service Manager program.

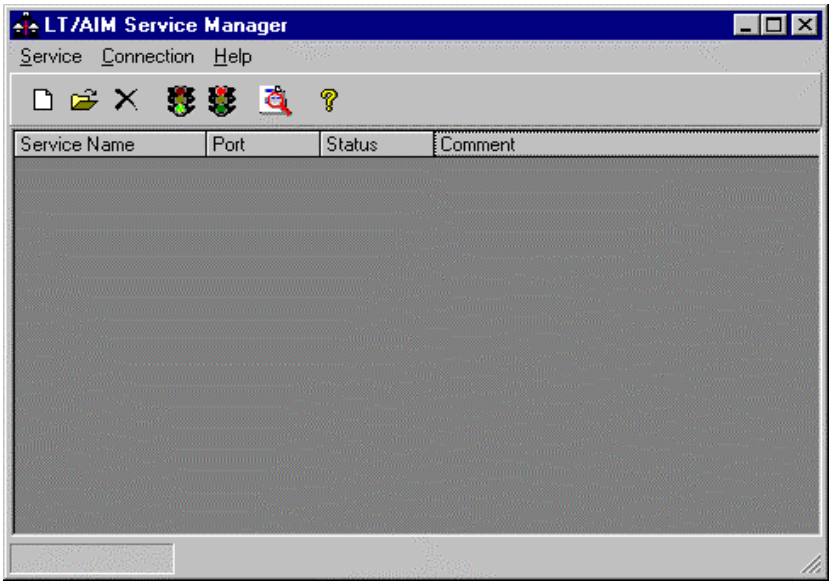

Figure B-1. Opening Screen of LeeTech AIM Service Manager

Parameters

Service Name The name of the listener.

Port The TCP/IP Port where the listener is waiting when the client program wants to connect to the server program.

Status This tells the user whether the listener is active or inactive.

Comment This could be any comment about a specific listener program.

Creating a New Service

Either click on service and then click on new, or use the toolbar icon for "New Service." The following screen will appear:

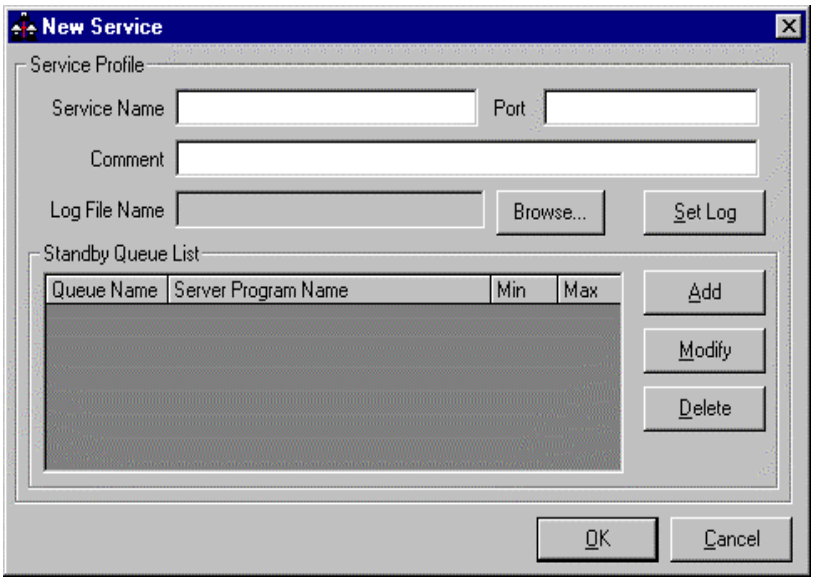

Figure B-2. NT Service Manager New Service Screen

Parameters

Service Name The "Service Name" is the name with which the user labels the LeeTech AIM listener for each new service.

Port The TCP/IP port is where the listener is waiting when the client program wants to connect to the server program.

Comment The "Comment" section is beneficial because it allows the user to put any comment in that will assist the user for quick referencing a specific function or explanation of the listener brought up on the screen.

Log File Name This is the file name to store the system log and debug trace.

Standby Queue List This is the list of queues that are monitored by the selected listener.

Parameters

Queue Name This can be any name without a space.

Server Program Name This can be any executable server program file name.

Min This is the minimum or initial number of server processes that the listener will open at any given time.

Max This is the maximum number of server processes that the listener will open at any given time.

Add This button opens the Add Standby Queue window. This is where the user sets the Queue Name, Server Program Name and Min/Max values.

Modify This button allows the user to modify the standby queue.

Delete This allows the user to delete any of the standby queues.

Using the Toolbar Functions

The Toolbar functions will allow the user to quickly perform one of the many functions available in the Service Manager.

New Service

To create a new service click on the *new service* icon in the tool bar at the top of the LT/AIM Service Manager screen. The New Service screen will appear.

Modifying and Deleting Service

To modify a service, highlight the “Service Name,” and click on the *modify* icon (the folder) in the tool bar. This will bring up the Modify Service screen. To modify a service, simply input the new information into the proper fields.

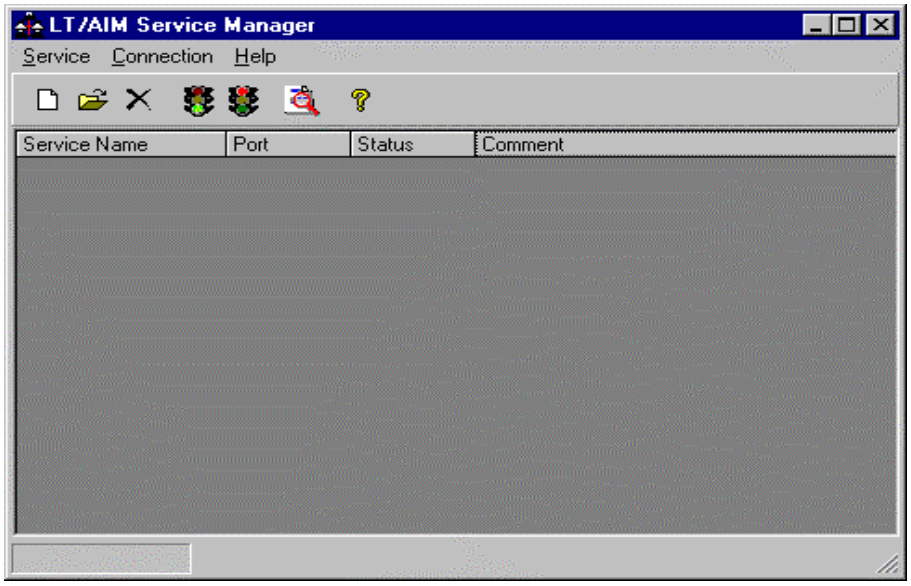

Figure B-3. NT Screen for Deleting/Modifying Service

To delete a service, highlight the “Service Name,” and click on the *delete* icon (the “X”) in the tool bar. This will delete the service highlighted.

Starting and Stopping Service

To start a Service, highlight the “Service Name” to be started and click on the *start service* icon (the green light) in the tool bar. The highlighted service name will start.

Figure B-4. NT Screen for Starting/Stopping Service

To stop a service, highlight the service name to be stopped and click on the *stop service* icon (the red light) in the tool bar. The highlighted service will cease.

Listing Connections to Listener

To list connections, highlight the "Service Name" desired and click on the *list connections* icon (the magnifying glass). A list of server programs will appear in a data box.

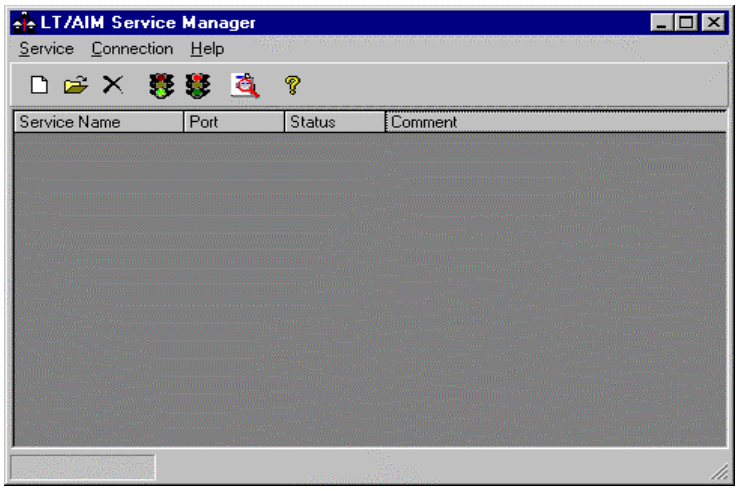

Figure B-5. NT Screen for Checking and Listing Connections to the Listener

C. LANUTIL

LeeTech AIM provides a utility program, LANUTIL, to help the system administrator monitoring and managing the LeeTech AIM client-server connections.

For HP MPE/iX users, run the program LANUTIL.PUB.LEETECH. For UNIX users, run /opt/leetech/bin/lanutil. It is a command mode program. The screen first shows up like:

```
HOST:[127.0.0.1]  APPLICATION:[32611]

Commands: LIST    - shows all connected users.
          KILL id - kills the specified user.
          STOPALL - terminates listener and all users.
          HOST id - sets to new host node name.
          APPL id - sets to new application name.
          SETQ qname #servers
                  - sets # of standby servers for a queue
          EXIT    - ends the LANUTIL.
LANUTIL>>
```

Keywords

HOST The IP address of the host that LANUTIL connects to. The default is 127.0.0.0, which is the system LOOPBACK interface. In order to make LeeTech AIM working, the LOOPBACK interface should be configured first. The user can type LANUTIL command, **HOST** *"remote host name" or "remote host IP address,"* to switch the LANUTIL management channel from host to host.

APPLICATION The TCP/IP port number the listener is waiting on for the client's request. When a listener starts, it binds itself to a specific TCP/IP port. Multiple listeners may be running on the same host at the same time. The user can type LANUTIL command APPL "port number" or "port ID," to switch the LANUTIL management channel from listener to listener.

SETQ At run time, the user can reset the number of StandBy server programs in a StandBy Queue. The number may

be less than the preset MIN value of the queue, but no more than the preset MAX value.

Following is a typical LANUTIL connection list. If the server program is configured in StandBy mode, the Queue name will shows before the server program name.

```
LANUTIL>>LIST

 ID  Pin/Type              Start/Program/Login
==== ======== ==========================================
   1 161      97/10/01 11:14:05 [Q1]APPCARE.PUB.CARE
     TCP/IP   APPCARE
   2 106      97/10/01 11:14:05 [Q1]APPCARE.PUB.CARE
     TCP/IP   APPCARE
   3 152      97/10/01 11:14:04 [Q1]APPCARE.PUB.CARE
     TCP/IP   APPCARE
   4 91       97/10/01 11:14:04 [Q1]APPCARE.PUB.CARE
     TCP/IP   APPCARE
   5 40       97/10/01 11:14:04 [Q1]APPCARE.PUB.CARE
     TCP/IP   APPCARE

HOST:[127.0.0.1]  APPLICATION:[32611]

Commands: LIST    - shows all connected users.
          KILL id - kills the specified user.
          STOPALL - terminates listener and all users.
          HOST id - sets to new host node name.
          APPL id - sets to new application name.
          SETQ qname #servers
                  - sets # of standby servers for a queue
          EXIT    - ends the LANUTIL.
LANUTIL>>
```

For NT server users, the LANUTIL has been integrated with LeeTech AIM NT Service Manager. The connections can be easily managed by clicking List icon from toolbar and doing the same tasks as the command mode LANUTIL.

D. Customer Support

LeeTech Software Inc. is dedicated to providing the highest quality software products available. In addition, LeeTech believes that high quality, knowledgeable support is an essential element for full utilization of the supplied products. LeeTech has thus structured a complete support package that is available with all supplied products.

All support contracts include telephone or message based support. LeeTech's staff is qualified to assist not only with product use, but also with any database problems customers might have. If problems prove extensive and outside the scope of the maintenance contract, LeeTech can still assist with low cost consulting.

In addition to specific service requests, LeeTech provides users with periodic newsletters, software bulletins and other materials designed to help ensure effective use of LeeTech products. If you with to be placed on the mailing list simply send your request to LeeTech.

LeeTech customer service can be contacted by any of the following.

Mail

Please send all inquiries or problems to:

LeeTech Software Inc.
2085 Hamilton Ave., Suite 200
San Jose, CA 95125
U.S.A.

Telephone Support

All LeeTech customers with a current support services contract can contact LeeTech at the following numbers.

U.S.A.	(800) 995-1987
All Other Countries	(408) 253-1987

Fax-Back Service

LeeTech maintains a 24-hour Fax number for customer input. Questions regarding support services receive immediate attention at the beginning of the following business day. Responses are returned by Fax or by the method designated on the submittal.

Fax Number (408) 558-0802

Electronic Messaging

Questions or input can be made via electronic mail.

- Support needs can be mailed to:

support@leetech.com

- Information inquiries can be mailed to:

info@leetech.com

- All product and sales needs can be mailed to:

sales@leetech.com

INDEX