ALTIBASE Application Development

Log Analyzer User's Manual

release 5.3.3

ALTIBASE Application Development Log Analyzer User's Manual

Release 5.3.3

Altibase Corporation

10F, Daerung PostTower II, 182-13,

Guro-dong Guro-gu Seoul, 152-847, Korea

Telephone: +82-2-2082-1000 Fax: 82-2-2082-1099

E-mail: support@altibase.com www: http://www.altibase.com

Contents

Preface

Regarding This Manual

This manual describes the concept of Altibase Log Analyzer and its use.

Intended Audience

The manual has been prepared for the following Altibase users:

- Database Administrator
- Performance Administrator
- Database User
- Application Development
- Technical Assistant Team

Before reading this manual, understanding of following background knowledge is recommended.

- Basic knowledge regarding operation of a computer, operating system and operating system utility.
- Experience in use of a relational database or understanding of the database concept
- Experience in administration of database server, operating system or network
- Experience in computer programming

Software Environment

The manual has been prepared by assuming that Altibase Version 5.3.1 is used as a database server.

Organization

The manual consists of the following sections:

- Chapter 1 Introduction
- Chapter 2 XLog Sender
- Chapter 3 XLog Analysis
- Chapter 4 Log Analysis API
- Appendix-A Error Codes
- Appendix-B Sample Codes

Conventions

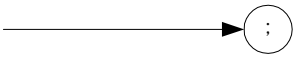
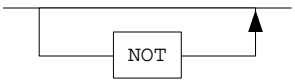
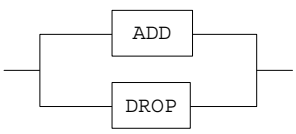
This section describes the conventions used throughout the manual. If you get familiarized with these rules, you will be able to find necessary information more easily from this manual or other manuals in the documentation set.

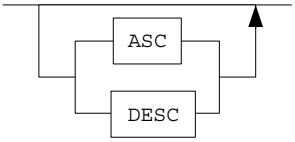
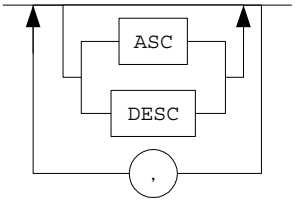
The rules used throughout the manual are as follows:

- Syntax Diagram
- Sample Code Rules

Syntax Diagram

In order to describe the syntax of a statement, this manual uses a diagram that consists of the following components:

Component	Meaning
	A statement begins. A syntactic element that is not a complete statement begins with an arrow.
	A statement continues in the next line. A syntactic element that is not a complete statement ends with this symbol.
	A statement continues from the previous line. A syntactic element that is not a complete statement begins with this symbol.
	A statement ends.
	A required item
	An optional item
	A required item with options. Only one item must be specified.

Component	Meaning
	An optional item with options.
	An optional item. More than one item is allowed. Each repetition must be preceded by a comma.

Sample Code Rules

Code examples demonstrate the use of SQL, Stored Procedure, iSQL or other command line syntax.

The following table describes the print conventions used in code examples.

Rule	Meaning	Example
[ ]	Indicates an optional item.	VARCHAR [(size)] [[FIXED |] VARIABLE]
{ }	Indicates a required item. At least one item must be selected.	{ ENABLE | DISABLE | COMPILE }
|	Separates arguments for an optional or required item.	{ ENABLE | DISABLE | COMPILE }[ENABLE | DISABLE | COMPILE]
...	Repeats the previous arguments.Indicates an omission in an example code.	SQL> SELECT ename FROM employee; ENAME ---------- SWNO HJNO HSCHOI ... 20 rows selected.
Other Symbols	The symbols other than those shown above.	EXEC :p1 := 1; acc NUMBER(11,2);

Rule	Meaning	Example
Italic	A variable that should be specified by a user. A placeholder to which a specific value should be supplied.	SELECT * FROM table_name; CONNECT userID/password;
Lower Case	Program elements provided by a user. Ex. table name, column name, file-name, etc.	SELECT ename FROM employee;
Upper Case	Elements provided by the system or keywords appearing in a statement.	DESC SYSTEM_.SYS_INDICES_;

References

For more information, please refer to the following documentation:

- Altibase Administration Starting User's Manual
- Altibase Administration Administrator's Manual
- Altibase Administration Replication User's Manual
- Altibase Application Development SQL User's Manual
- Altibase Application Development ODBC User's Manual
- Altibase Application Development Spatial SQL User's Manual
- Altibase Application Development Application Program Interface User's Manual
- Altibase Tools iSQL User's Manual
- Altibase Message Error Message Reference

Online Manual

Korean and English versions of on-line manuals (PDF or HTML) are available from Altibase Download Center (http://atc.altibase.com/).

Altibase Welcomes Your Opinions!

Please send us your comments and suggestions regarding this manual. Your comments and suggestions are important, and they may be used to improve future versions of the manual. When you send your feedback, please make sure to include the following information:

- The name and version of the manual in use
- Your comments or suggestions regarding the manual
- Your name, address, and phone number

Please send your e-mail to the following address:

support@altibase.com

This address is intended to report any errors or omissions discovered in the manual. When you need an immediate assistance regarding technical issues, please contact Altibase Customer Support Center.

We always appreciate your comments and suggestions.

1 Introduction

This section describes the concept of Altibase Log Analyzer and its basic use.

Altibase Log Analyzer

Altibase Log Analyzer is a set of modules and API in Altibase DBMS that provides a history of DML-related transactions based on active logs in Altibase DBMS.

Altibase Log Analyzer can be used for:

1. linking of Altibase DBMS and other DBMS.
2. detecting and handling of changes within Altibase DBMS from the outside of Altibase DBMS.

Terms & Concepts

XLog

XLog is a logical log that is converted from a physical log.

It stores the DML(Insert/Update/Delete) transaction history.

XLog Sender

XLog Sender is a module that creates XLog by analyzing active logs and passes it to XLog Collector.

XLog Sender performs a handshake and XLog transmission actively.

XLog Collector

XLog Collector is a module that receives meta data and XLog from XLog Sender.

XLog Collector contains meta data, XLog Queue, Trasaction Table and XLog Pool. It is called via Log Analysis API.

Log Analysis API

Log Analysis API provides XLog and the meta data that can be used to interpret the XLog.

Handshake

A handshake is a process in which XLog Collector checks protocol version and meta data before it sends/receives XLog.

XLog Queue

XLog Queue is a place where available XLog is stored before a user can get it.

XLog Pool

XLog Pool is a place where memory to allocate for XLog is stored.

XLog Pool is used to reuse memory to allocate for XLog and to prevent excessive use of memory.

Transaction Table

Transaction Table is a place where the transaction status and additional information are stored.

Restart SN

Restart SN is the SN for an active log that will be read when XLog Sender is restarted.

SN

SN (Sequence Number) is a serial number of a log record in an active log.

Replication

Replication is a module that synchronizes data between different Altibase DBMS's in real-time. Please refer to the Altibase Administration Replication User's Manual to get more information.

Replication SYNC

Replication SYNC is a function that sends all records in the replication tables of a local server to a remote server.

It is used to synchronize replication tables before the replication module is started based on active logs. Please refer to the Altibase Administration Replication User's Manual to get more information.

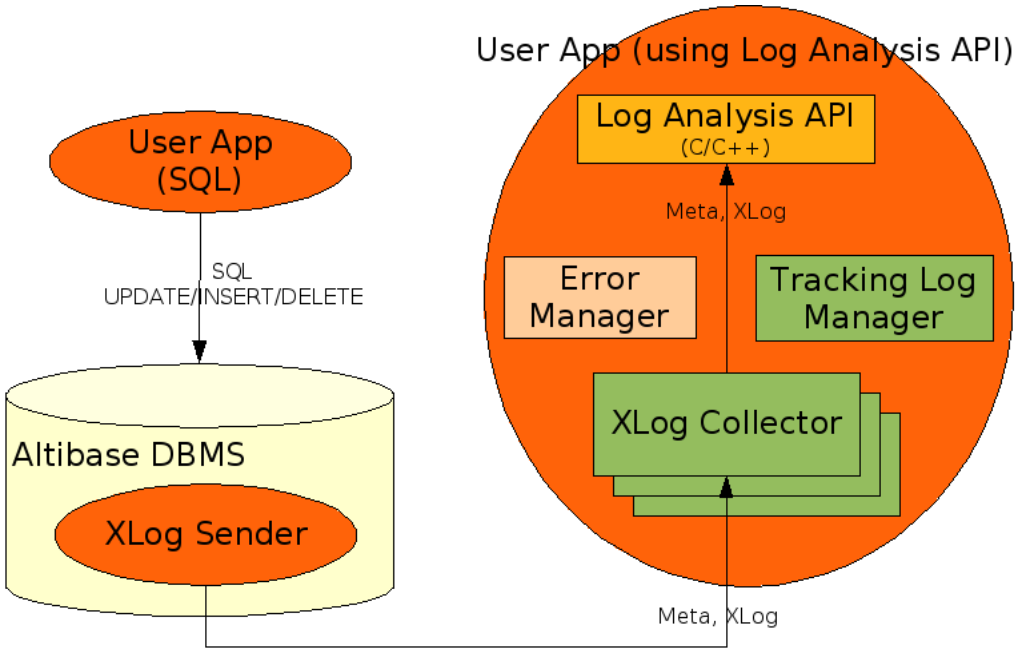
How Altibase Log Analyzer Works

XLog Sender exists within Altibase DBMS. It creates XLog with active logs, and sends the XLog and its meta data to XLog Collector. XLog Collector exists within user's application. It provides a user with XLog and its meta data via Log Analysis API.

If it fails to invoke Log Analysis API, a proper action should be taken based on a cause for the error. The latest error information is stored in Error Manager. Log Manager is provided to trace the error. Log Manager records brief trace and error information.

The entire structure is shown in the following illustration:

Figure 1-1 The Structure of Altibase Log Analyzer



With Log Analysis API, a user can obtain XLog and the meta data from XLog Collector. The movement of the meta data and XLog within XLog Collector is as follows:

The meta data is received by XLog Sender during handshaking, and it is valid until the next handshaking.

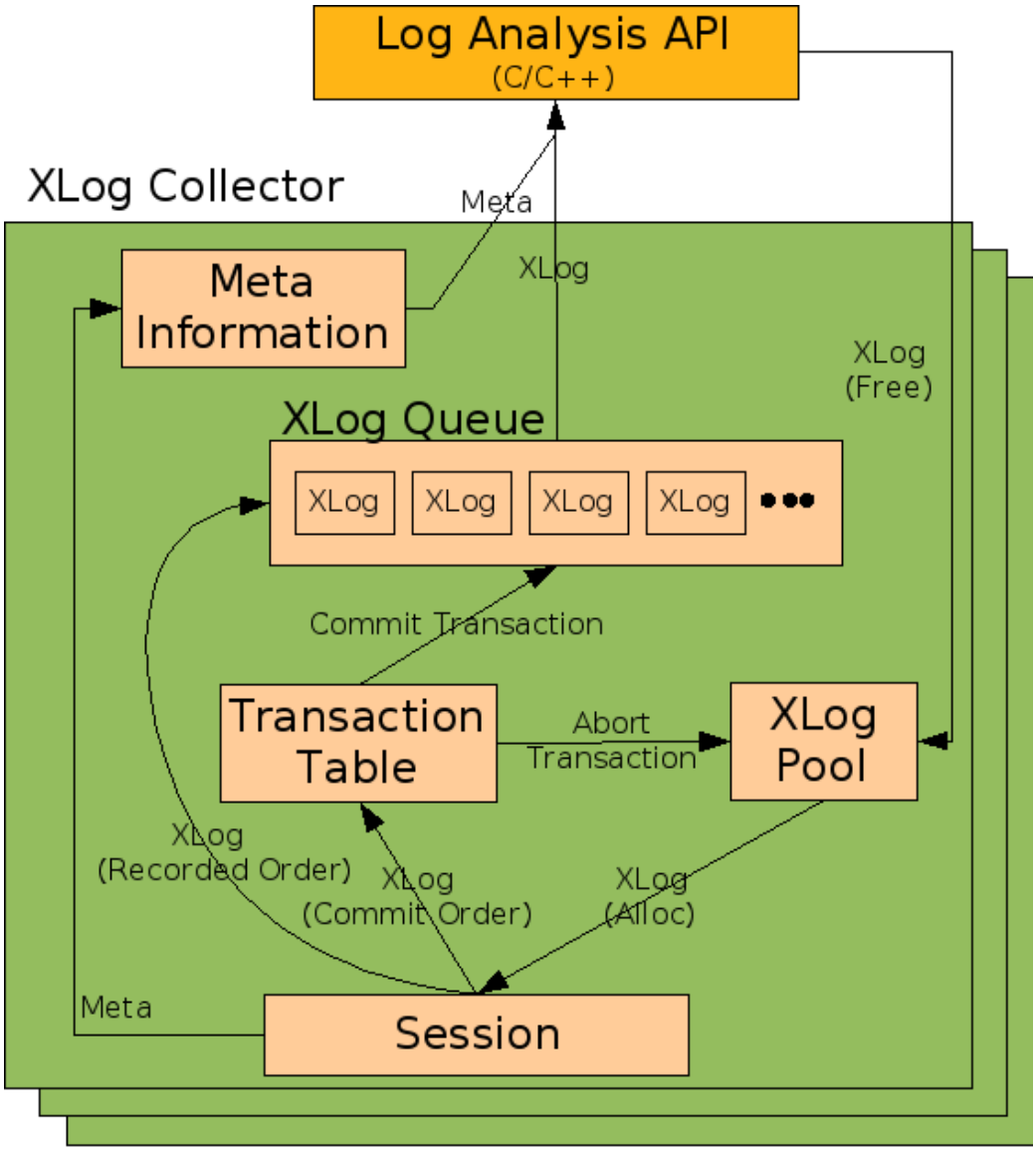
XLog is circulated in the following order:

1. Memory for XLog is obtained from XLog Pool.
2. XLog Sender receives the XLog configuration data and creates XLog.
3. The XLog is added to XLog Queue and a user.takes it from there.

   To obtain the XLog for a transaction in the order of commit, the XLog is temporalily stored in Transaction Table before it is added to XLog Queue.
4. The used XLog is returned to XLog Pool.

The following illustration shows the movement of the meta data and XLog within XLog Collector:

Figure 1-2 The Structure of XLog Collector



Features

Log Sender uses the Replication module

XLog Sender is managed with the almost same SQL as in the Replicaton, and it is applied with properties in Replication.

For more information on Replication, please refer to the Altibase Administration Replication User's Manual.

Transaction XLog can be obtained in the order of commit

When XLog Collector is created, it can be set such that transaction XLog can be obtained in the order of commit. If this is the case, the following restrictions apply:

- The corresponding transaction XLog can be obtained after COMMIT XLog is received.

  When a user obtains XLog, the savepoint-related XLog is not neccessary and thus it is not provided.

- A user may not obtain XLog for the Abort transaction.

TCP and UNIX Domain are supported for XLog transmission.

- UNIX Domain can be used only when XLog Sender and XLog Collector are in the same machine and the OS is UNIX or LINUX.
- Only one socket type can be used for a XLog Sender.

Conversion to the ODBC C value is supported.

The internal data of Altibase can be converted to the ODBC C value.

Restrictions

Because XLog Sender uses the Replication module, the following restrictions apply:

- A user must have the SYSDBA system privilege.
- Analysis targets are in table-base.
- A table to analyze must have a primary key.
- The primary key of the table to analyze cannot be modified.
- However, INSERT or DELETE can be performed on the primary key.
- DDL cannot be performed on the table to analyze.
- Up to 32 XLog Senders and Replication Senders can be created together in a single Altibase DBMS.
- The protocol version of Log Analysis API should be the same with the protocol version of Replication.

  If there are more than one XLog Collector in a single process, the protocol version of Replication in Altibase DBMS should be the same with the protocol version of Replication.

However, unlike Replication, it:

- does not examine the foreign key column in a table.
- supports Lazy Mode only.

- does not support Replication SYNC.

For more information on Replication, please refer to the Altibase Administration Replication User's Manual.

How to Use Log Analysis API

This section explains how to use Log Analysis API that is included in user's application.

For information on using XLog Sender, please refer to 'Chapter 2 XLog Sender.'

Required Files

Table 1-1 Required Files

Type	Filename	Description
Header	alaAPI.h	The file is required to use Log Analysis API and it contains alaTypes.h.
	alaTypes.h	It defines data types that will be used by Log Analysis API.
Library	libala_sl.x	A shared library of Log Analysis API.
	libala.x	A static library of Log Analysis API.

The followings should be considered when a source code is being created and compiled:

- User's source file should contain the 'alaAPI.h' file.

  If it is going to be used in Windows, it should contain the 'windows.h' before the 'alaAPI.h' file. If _WINDOWS_ is not defined in the 'windows.h' file, it should be defined by a user.

- For compiling, a shared library or static library should be linked.

- The extension of a library varies depending on a platform.

Data Type

The basic data types used by Log Analysis API are as follows:

Table 1-2 Basic Data Types

Type	Data Type	Description
Boolean	ALA_BOOL	ALA_TRUE : true ALA_FALSE : false
Return Code	ALA_RC	ALA_SUCCESS : success ALA_FAILURE : fail
Character	SChar	Signed Character (8 bits)
	UChar	Unsigned Character (8 bits)

Type	Data Type	Description
Integer	SShort	Signed Small Integer (16 bits)
	UShort	Unsigned Small Integer (16 bits)
	SInt	Signed Integer (32 bits)
	UInt	Unsigned Integer (32 bits)
	SLong	Signed Big Integer (64 bits)
	ULong	Unsigned Big Integer (64 bits)

Error Handling

All Log Analysis API's receive Error Manager as an argument. If invoking Log Analysis API results in ALA_FAILURE, a cause for the error should be checked for and handled. The error information that is provided to a user consists of Error Code, Error Level and Error Message.

```
typedef struct ALA_ErrorMgr
{
 UInt mErrorCode; /* CODE */
 SChar mErrorState[6]; /* STATE */
 SChar mErrorMessage[ALA_MAX_ERROR_MSG_LEN+256];
} ALA_ErrorMgr;
```

The followings should be considered when the Error Manager is used:

- A host (a process or thread) that invokes Log Analysis API creates and stores the Error Manager.
- The Error Manager keeps information on the last error only.
- Error handling functions cannot use a NULL as the Error Manager's argument.
- If a function other than the error handling functions uses the NULL as the Error Manager's argument in Log Analysis API, an error is not recorded by Log Manager when it occurs.
- Since an error code contains internal data, it must be obtained with ALA_GetErrorCode().

ALA_ErrorLevel can be obtained with ALA_GetErrorLevel() and it represents the error level.

```
typedef enum
{
    ALA_ERROR_FATAL = 0, /* Need to Destroy */
    ALA_ERROR_ABORT, /* Need to Handshake */
    ALA_ERROR_INFO /* Information */
} ALA_ErrorLevel;
```

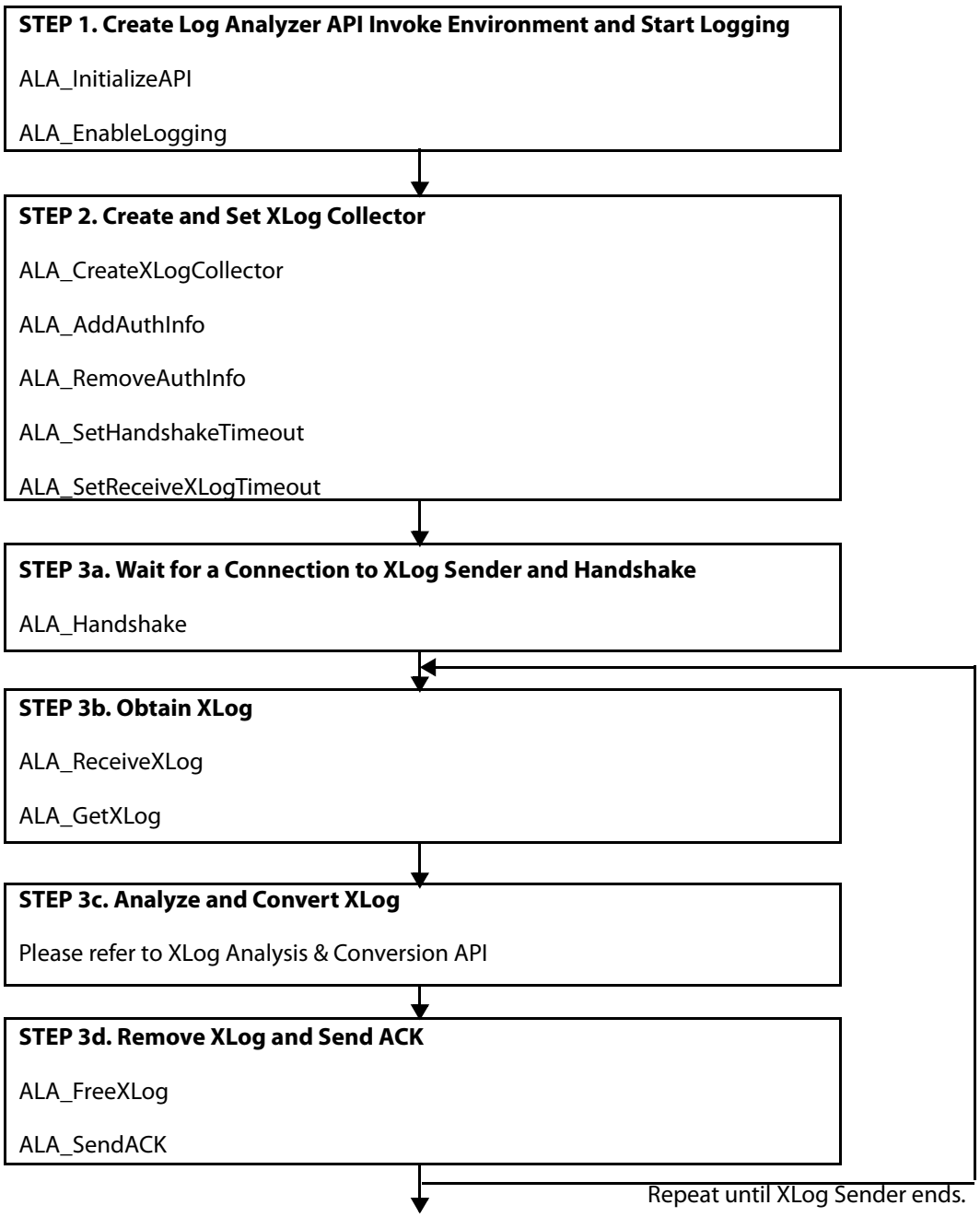
Actions to be taken for each error level are as follows:

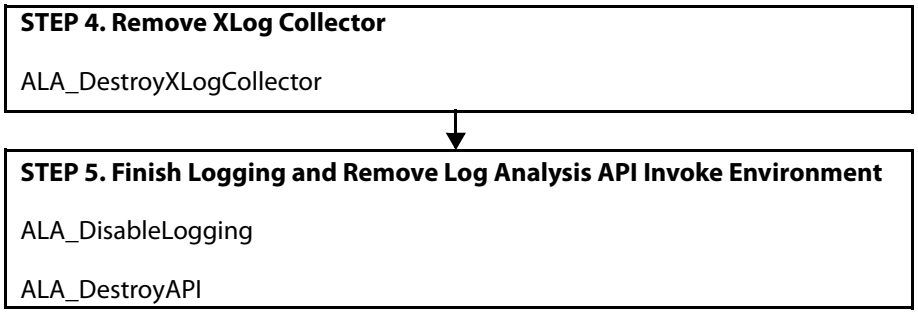
- Since ALA_ERROR_FATAL is a fatal error, ALA_DestroyXLogCollector() should be invoked to end the corresponding XLog Collector.

- Since ALA_ERROR_ABORT indicates that XLog Collector is in an abnormal status, ALA_Handshake() should be invoked to perform handshaking again for the corresponding XLog Collector.
- ALA_ERROR_INFO indicates that invoking Log Analysis API has failed. An appropriate action should be taken based on the error code.

Error that have already occurred can be checked via Log Manager. For information on using Log Manager, please refer to the sections that describe ALA_EnableLogging() and ALA_DisableLogging().

Basic Use





When Log Analysis API is used, the followings should be considered.

- XLog Collector Monitoring is used in STEP 3a - 3d.
- Error Handling API is used in all steps.
- If there are more than one XLog Collector, STEP 2 - 4 should be performed for each XLog Collector.
- ALA_SendACK() does not have to be called every time. For information on when to send ACK, please refer to ALA_CreateXLogCollector() and ALA_SendACK().
- If XLog is to be applied to DBMS via ODBC in STEP 3c, the Autocommit option should be turned off.
- XLog Sender should be started after STEP 3a.
- In STEP 3b, ALA_ReceiveXLog() and ALA_GetXLog() can be invoked by different threads from each other.
- Once ALA_FreeXLog is used, the corresponding XLog and data cannot be used anymore.

Summary of Log Analysis API

Log Analysis API Environment Management

Type	Log Analysis API	Description
Create & Remove Log Analysis API Environment	ALA_InitializeAPI	Create environment in which Log Analysis API can be invoked.
	ALA_DestroyAPI	Remove environment in which Log Analysis API can be invoked.
Logging	ALA_EnableLogging	Enable logging for problem tracking.
	ALA_DisableLogging	Disable logging.

XLog Collector-related API

Type	Log Analysis API	Description
Create & Prepare XLog Collector	ALA_CreateXLogCollector	Create XLog Collector corresponding to XLog Sender.
	ALA_AddAuthInfo	Add authentication information for XLog Sender.
	ALA_RemoveAuthInfo	Remove authentication information for XLog Sender.
	ALA_SetHandshakeTimeout	Specify the handshake timeout.
	ALA_SetReceiveXLogTimeout	Specify the XLog receiving timeout.
Receive Meta Data and XLog	ALA_Handshake	Wait for a connection to XLog Sender and perform handshaking.
	ALA_ReceiveXLog	After receiving XLog, add it to XLog Queue.
	ALA_GetXLog	Obtain XLog from XLog Queue.
	ALA_SendACK	Send ACK to XLog Sender.
	ALA_FreeXLog	Return XLog to XLog Pool.
Remove XLog Collector	ALA_DestroyXLogCollector	Remove XLog Collector.

Type	Log Analysis API	Description
Monitor XLog Collector	ALA_GetXLogCollectorStatus	Obtain the status of XLog Collector.

XLog Analysis & Conversion API

Type	Log Analysis API	Description
XLog	ALA_GetXLogHeader	Obtain the XLog header from XLog.
	ALA_GetXLogPrimaryKey	Obtain the XLog header from XLog.
	ALA_GetXLogColumn	Obtain the XLog column from XLog.
	ALA_GetXLogSavepoint	Obtain the XLog savepoint from XLog.
	ALA_GetXLogLOB	Obtain the XLog LOB from XLog.
Meta data	ALA_GetProtocolVersion	Obtain the protocol version of Log Analysis API.
	ALA_GetReplicationInfo	Obtain the Replication information.
	ALA_GetTableInfo	Retrieve the table information by table OID.
	ALA_GetTableInfoByName	Retrieve the table information by user name and table name.
	ALA_GetColumnInfo	Retrieve the column information in a table by Column ID.
	ALA_GetIndexInfo	Retrieve the index information in a table by Index ID.
Altibase Internal Data Type	ALA_GetInternalNumericInfo	Obtain the sign and exponent of FLOAT and NUMERIC.
	ALA_GetAltibaseText	Convert the internal data of Altibase to a string.
	ALA_GetAltibaseSQL	Convert transaction XLog to an Altibase SQL string.
ODBC C Conversion	ALA_GetODBCCValue	Convert the internal data of Altibase to the ODBC C value.

Error Handling API

Type	Log Analysis API	Description
Error Handling	ALA_ClearErrorMgr	Initialize Error Manager.
	ALA_GetErrorCode	Obtain an error code.
	ALA_GetErrorLevel	Obtain the error level.
	ALA_GetErrorMessage	Obtain a specific error message.

2 XLog Sender

This section provides description on using the XLog Sender component of Altibase Log Analyzer.

XLog Sender combines log records in XLog and passes it to XLog Collector. XLog Sender is a module within Altibase DBMS and it is managed via the SQL interface similar to that for replication.

XLog Sender SQL

Create XLog Sender

Syntax

```
CREATE REPLICATION replication_name FOR ANALYSIS
        WITH {{'remote_host_ip', remote_host_ port_no}
        ...
        | UNIX_DOMAIN}
    FROM user_name.table_name TO user_name.table_name
    [, FROM user_name.table_name TO user_name.table_name]
    ...
    ;
```

Description

Creates XLog Sender.

- Lazy Mode is forced.
- 'UNIX_DOMAIN' in the WITH clause specifies the UNIX Domain Connection.
- In the FROM clause, a table with a foreign key can be specified.

The remaining parts are the same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

The UNIX Domain Connectivity can only be used in UNIX and LINUX.

Example

XLog Sender Name : log_analysis

XLog Collector Info : TCP(IP : 127.0.0.1, PORT : 35300)

IP of server running XLog Collector

Port number specified in XLog Collector

Table for Analysis : sys.t1

```
iSQL> CREATE REPLICATION log_analysis FOR ANALYSIS
            WITH '127.0.0.1', 35300
            FROM sys.t1 TO sys.t1;
```

Remove XLog Sender

Syntax

```
DROP REPLICATION replication_name;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Example

XLog Sender Name : log_analysis

```
iSQL> DROP REPLICATION log_analysis;
```

Start XLog Sender

Syntax

```
ALTER REPLICATION replication_name {START|QUICKSTART};
```

Description

Starts XLog Sender.

- For UNIX Domain, a socket filename is automatically created.
- Socket Filename : $ALTIBASE_HOME/trc/rp-replication_name
- Unlike in Replication, it is not registered in the Heart-Beat thread.

The remaining parts are the same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

XLog Collector should be available for connection before this task can be performed.

When a UNIX Domain is used, the same environment variable '$ALTIBASE_HOME' should be specified in XLog Collector.

Since the maximum allowable length of a socket filename varies depending on an operating system, the maximum allowable length should be checked so that it is not exceeded.

Example

XLog Sender Name : log_analysis

Start Point for Log Analysis : The point at which the last analysis ended.

```
iSQL> ALTER REPLICATION log_analysis START;
```

End XLog Sender

Syntax

```
ALTER REPLICATION replication_name STOP;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Example

XLog Sender Name : log_analysis

```
iSQL> ALTER REPLICATION log_analysis STOP;
```

Add a Table for Analysis

Syntax

```
ALTER REPLICATION replication_name ADD TABLE
    FROM user_name.table_name TO user_name.table_name;
```

Description

Adds a table for analysis.

In the FROM clause, a table with a foreign key can be specified.

The remaining parts are the same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Example

XLog Sender Name : log_analysis

Table for Analysis : sys.t2

```
iSQL> ALTER REPLICATION log_analysis ADD TABLE
 FROM sys.t2 TO sys.t2;
```

Remove a Table for Analysis

Syntax

```
ALTER REPLICATION replication_name DROP TABLE
    FROM user_name.table_name TO user_name.table_name;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Example

XLog Sender Name : log_analysis

Table for Analysis : sys.t2

```
iSQL> ALTER REPLICATION log_analysis DROP TABLE
 FROM sys.t2 TO sys.t2;
```

Add a Host

Syntax

```
ALTER REPLICATION replication_name
    ADD HOST 'remote_host_ip', remote_port_no;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

Only TCP information can be added.

A host cannot be added when a UNIX Domain is already specifed as a connection type.

Example

XLog Sender Name : log_analysis

XLog Collector Info : TCP(IP : 127.0.0.1, PORT : 30301)

```
iSQL> ALTER REPLICATION log_analysis
 ADD HOST '127.0.0.1', 30301;
```

Remove a Host

Syntax

```
ALTER REPLICATION replication_name
    DROP HOST 'remote_host_ip', remote_port_no;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

Only the TCP information can be removed.

Example

XLog Sender Name : log_analysis

XLog Collector Info : TCP(IP : 127.0.0.1, PORT : 30301)

```
iSQL> ALTER REPLICATION log_analysis
 DROP HOST '127.0.0.1', 30301;
```

Set a Host

Syntax

```
ALTER REPLICATION replication_name
    SET HOST 'remote_host_ip', remote_port_no;
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

It is applied when XLog Sender is started.

Only the TCP information can be specified.

Example

XLog Sender Name : log_analysis

XLog Collector Info : TCP(IP : 127.0.0.1, PORT : 30301)

```
iSQL> ALTER REPLICATION log_analysis
 SET HOST '127.0.0.1', 30301;
```

Flush XLog

Syntax

```
ALTER REPLICATION replication_name FLUSH [ALL]
    [WAIT timeout_sec];
```

Description

The same as in Replication.

For more information, please refer to the Altibase Administration Replication User's Manual.

Note

If XLog Collector does not send ACK, a timeout may occur.

Example

XLog Sender Name : log_analysis

Reference Point for Flush : The time point at which it is performed.

Timeout : 10 seconds

```
iSQL> ALTER REPLICATION log_analysis FLUSH WAIT 10;
```

Meta Table

The same meta table as in Replication is used.

SYSTEM_ SYS_REPLICATIONS_

It has information on the settings and status of XLog Sender.

Column name	Type	Description
REPLICATION_NAME	VARCHAR(40)	Replication Name
LAST_USED_HOST_NO	INTEGER	The most recently used remote server
HOST_COUNT	INTEGER	The number of remote servers
IS_STARTED	INTEGER	Whether replication has been started or not.
XSN	BIGINT	The Transmission Start SN for a sender
ITEM_COUNT	INTEGER	The number of tables to replicate
CONFLICT_RESOLUTION	INTEGER	The conflict resolution for replication
REPL_MODE	INTEGER	The default replication mode
ROLE	INTEGER	Role
OPTIONS	INTEGER	Flag for Extra Features of Replication
INVALID_RECOVERY	INTEGER	Whether to recover replication

If the value for the role column is 1, it indicates XLog Sender.

For more information, please refer to the Altibase Administration Administrator's Manual.

SYSTEM_.SYS_REPL_HOSTS_

It has information on a target to which XLog Sender will connect.

Column name	Type	Description
HOST_NO	INTEGER	Serial Number
REPLICATION_NAME	VARCHAR(40)	Replication Name
HOST_IP	VARCHAR(40)	IP address for a remote server
PORT_NO	INTEGER	Replication port number for a remote server

For a UNIX domain, the value for the HOST_IP column is 'UNIX_DOMAIN' and the value for the PORT_NO column is the value for the HOST_NO column.

For more information, please refer to the Altibase Administration Administrator's Manual.

SYSTEM_.SYS_REPL_ITEMS_

It has information on a table for analysis.

Column name	Type	Description
REPLICATION_NAME	VARCHAR(40)	Replication Name
TABLE_OID	BIGINT	Table Object Identifier
LOCAL_USER_NAME	VARCHAR(40)	Username for a local server
LOCAL_TABLE_NAME	VARCHAR(40)	Table name for a local server
LOCAL_PARTITION_NAME	VARCHAR(40)	Partition Name in Local Server
REMOTE_USER_NAME	VARCHAR(40)	Username for a remote server
REMOTE_TABLE_NAME	VARCHAR(40)	Table name for a remote server
REMOTE_PARTITION_NAME	VARCHAR(40)	Partition Name in Remote Server
IS_PARTITION	CHAR(1)	Whether to Partition
INVALID_MAX_SN	BIGINT	The highest SN of logs to skip.
CONDITION	VAR-CHAR(1000)	Replication Conditional Clause

For a UNIX domain, the value for the HOST_IP column is 'UNIX_DOMAIN' and the value for the PORT_NO column is the value for the HOST_NO column.

For more information, please refer to the Altibase Administration Administrator's Manual.

Performance View

It provides the same performance view as in Replication.

V$REPEXEC

It has information on an administrator.

Column name	Type	Description
PORT	INTEGER	The number of port being used.
MAX_SENDER_COUNT	INTEGER	The maximum number of senders
MAX_RECEIVER_COUNT	INTEGER	The maximum number of receivers

For more information, please refer to the Altibase Administration Administrator's Manual.

V$REPSENDER

It has information on the status of XLog Sender.

Column name	Type	Description
REP_NAME	VARCHAR(42)	Name
START_FLAG	BIGINT	Start flag
NET_ERROR_FLAG	BIGINT	Error status flag
XSN	BIGINT	SN of shipping log
STATUS	BIGINT	Current status
SENDER_IP	VARCHAR(50)	Sender's IP
PEER_IP	VARCHAR(50)	Remote IP
SENDER_PORT	INTEGER	Sender's port
PEER_PORT	INTEGER	Remote port
XSN	BIGINT	The SN for the transmission log
COMMIT_XSN	BIGINT	The SN for the commit log
READ_LOG_COUNT	BIGINT	The number of logs read
SEND_LOG_COUNT	BIGINT	The number of replication logs read
REPL_MODE	VARCHAR(7)	Current Replication Mode

Column name	Type	Description
ACT_REPL_MODE	VARCHAR(7)	Replication runs in lazy mode if its process exceeds the value of property.

For a UNIX domain, the value for the SENDER_IP and PEER_IP columns is 'UNIX_DOMAIN' and the value for the SENDER_PORT and PEER_PORT columns is 0.

For more information, please refer to the Altibase Administration Administrator's Manual.

V$REPSENDER_TRANSTBL

It has information on the transaction table in XLog Sender.

Column name	Type	Description
REP_NAME	VARCHAR(40)	Name
LOCAL_TID	INTEGER	The local transaction ID
REMOTE_TID	INTEGER	Currently not used.
BEGIN_FLAG	INTEGER	Whether BEGIN for a transaction has been sent or not.
BEGIN_SN	BIGINT	The initial log SN for a transaction

For more information, please refer to the Altibase Administration Administrator's Manual.

V$REPGAP

It has information on the log analysis progress.

Column name	Type	Description
REP_NAME	VARCHAR(40)	Name
REP_LAST_SN	BIGINT	The serial number for the last log
REP_SN	BIGINT	The serial number for the transmission log
REP_GAP	BIGINT	Interval
READ_LFG_ID	INTEGER	Group of Log Files Read Currently
READ_FILE_NO	INTEGER	The Number of Log Files Read Currently
READ_OFFSET	INTEGER	Location Read Currently

For more information, please refer to the Altibase Administration Administrator's Manual.

3 XLog Analysis

This chapter describes XLog, the meta data and Altibase internal data types that are required for XLog analysis.

XLog and the meta data can be obtained via Log Analysis API.

XLog

This section describes the types and components of XLog.

A user should invoke ALA_GetXLog() to obtain XLog.

Types of XLog

```
typedef enum
{
 XLOG_TYPE_BEGIN = 1,                /* Transaction Begin */
 XLOG_TYPE_COMMIT = 2,               /* Transaction Commit */
 XLOG_TYPE_ABORT = 3,                /* Transaction Rollback */
 XLOG_TYPE_INSERT = 4,               /* DML: Insert */
 XLOG_TYPE_UPDATE = 5,               /* DML: Update */
 XLOG_TYPE_DELETE = 6,               /* DML: Delete */
 XLOG_TYPE_SP_SET = 8,               /* Savepoint Set */
 XLOG_TYPE_SP_ABORT = 9,             /* Abort to savepoint */
 XLOG_TYPE_LOB_CURSOR_OPEN = 14,     /* LOB Cursor open */
 XLOG_TYPE_LOB_CURSOR_CLOSE = 15,    /* LOB Cursor close */
 XLOG_TYPE_LOB_PREPARE4WRITE = 16,   /* LOB Prepare for write */
 XLOG_TYPE_LOB_PARTIAL_WRITE = 17,   /* LOB Partial write */
 XLOG_TYPE_LOB_FINISH2WRITE = 18,    /* LOB Finish to write */
 XLOG_TYPE_KEEP_ALIVE = 19,          /* Keep Alive */
 XLOG_TYPE_REPL_STOP = 21            /* Replication Stop */
} ALA_XLogType;
```

XLog can be classified into 13 types of the transaction XLog and two types of the control XLog.

The transaction XLog begins with XLOG_TYPE_BEGIN and ends with XLOG_TYPE_COMMIT or XLOG_TYPE_ABORT.

Since LOB is a large data, updating LOB type of data can be made of more than one XLog. In this case, the LOB XLog is received in the following structure:

```
XLOG_TYPE_LOB_CURSOR_OPEN
{
XLOG_TYPE_LOB_PREPARE4WRITE
 {
XLOG_TYPE_LOB_PARTIAL_WRITE
 ...
 }
XLOG_TYPE_LOB_FINISH2WRITE
 ...
}
XLOG_TYPE_LOB_CURSOR_CLOSE
```

The XLog related the control includes KEEP_ALIVE and REPL_STOP.

KEEP_ALIVE is the XLog that XLog Sender sends to check if the network is still connected when it has no XLog to send.

REPL_STOP indicates that XLog Sender ends normally. Once ALA_SendACK() is invoked, the network is disconnected.

Configuration of XLog

```
typedef UInt ALA_TID;       /* Transaction ID */
typedef ULong ALA_SN;       /* Log Record SN */
typedef struct ALA_Value    /* Altibase Internal Data */
{
 UInt length;               /* Length of value */
 const void * value;
} ALA_Value;
```

Structure Member	Description
length	The length of the internal data value of Altibase.
Value	The internal data value of Altibase

```
typedef struct ALA_XLogHeader            /* XLog Header */
{
 ALA_XLogType mType;                     /* XLog Type */
 ALA_TID mTID;                           /* Transaction ID */
 ALA_SN mSN;                             /* SN */
 ALA_SN mSyncSN;                         /* Reserved */
 ULong mTableOID;                        /* Table OID */
} ALA_XLogHeader;
typedef struct ALA_XLogPrimaryKey        /* Primary Key */
{
 UInt mPKColCnt;                         /* Primary Key Column Count */
 ALA_Value *mPKColArray;                 /* Priamry Key Column Value Array */
} ALA_XLogPrimaryKey;
typedef struct ALA_XLogColumn            /* Column */
{
 UInt mColCnt;                           /* Column Count */
 UInt *mCIDArray;                        /* Column ID Array */
 ALA_Value *mBColArray;                 /* Before Image Column Value Array */
 ALA_Value *mAColArray;                 /* After Image Column Value Array */
} ALA_XLogColumn;
typedef struct ALA_XLogSavepoint         /* Savepoint */
{
 UInt mSPNameLen;                        /* Savepoint Name Length */
 SChar *mSPName;                         /* Savepoint Name */
} ALA_XLogSavepoint;
typedef struct ALA_XLogLOB               /* LOB */
{
 ULong mLobLocator;                      /* LOB Locator of Altibase */
 UInt mLobColumnID;
 UInt mLobOffset;
 UInt mLobOldSize;
 UInt mLobNewSize;
 UInt mLobPieceLen;
 UChar *mLobPiece;
} ALA_XLogLOB;
typedef struct ALA_XLog                  /* XLog */
{
 ALA_XLogHeader mHeader;
 ALA_XLogPrimaryKey mPrimaryKey;
 ALA_XLogColumn mColumn;
 ALA_XLogSavepoint mSavepoint;
 ALA_XLogLOB mLOB;
                                         /* Used internally */
```

```
 struct ALA_XLog *mPrev;
 struct ALA_XLog *mNext;
} ALA_XLog;
```

XLog consists of a header, primary key, column, savepoint and LOB.

Each of these components can be obtained directly from ALA_XLog or via Log Analysis API for XLog.

ALA_XLogPrimaryKey does not have the Primary Key Column ID Array. This can be obtained with mPKColumnArray[sIndex] -> mColumnID in the meta table. The meta table can be obtained via ALA_GetTableInfo() or ALA_GetTableInfoByName().

Configuration Based on XLog Type

The type of XLog can be identified with the mType member in ALA_XLogHeader.

BEGIN XLog

Header (mType, mTID, mSN, mSyncSN)

COMMIT XLog

Header (mType, mTID, mSN, mSyncSN)

ABORT XLog

Header (mType, mTID, mSN, mSyncSN)

INSERT XLog

Header (mType, mTID, mSN, mSyncSN, mTableOID)

Column (mColCnt, mCIDArray, mAColArray)

UPDATE XLog

Header (mType, mTID, mSN, mSyncSN, mTableOID)

Primary Key (mPKColCnt, mPKColArray)

Column (mColCnt, mCIDArray, mBColArray, mAColArray)

DELETE XLog

Header (mType, mTID, mSN, mSyncSN, mTableOID)

Primary Key (mPKColCnt, mPKColArray)

SP_SET XLog

Header (mType, mTID, mSN, mSyncSN)

Savepoint (mSPNameLen, mSPName)

- If mSPName begins with "$$IMPLICIT", it is the implict savepoint.
- If mSPName is "$$PSM_SVP", it is PSM Savepoint.

SP_ABORT XLog

Header (mType, mTID, mSN, mSyncSN)

Savepoint (mSPNameLen, mSPName)

- If mSPName begins with "$$IMPLICIT", it is the implict savepoint.
- If mSPName is "$$PSM_SVP", it is PSM Savepoint.

LOB_CURSOR_OPEN XLog

Header (mType, mTID, mSN, mSyncSN, mTableOID)

Primary Key (mPKColCnt, mPKColArray)

LOB (mLobLocator, mLobColumnID)

LOB_CURSOR_CLOSE XLog

Header (mType, mTID, mSN, mSyncSN)

LOB (mLobLocator)

LOB_PREPARE4WRITE XLog

Header (mType, mTID, mSN, mSyncSN)

LOB (mLobLocator, mLobOffset, mLobOldSize, mLobNewSize)

LOB_PARTIAL_WRITE XLog

Header (mType, mTID, mSN, mSyncSN)

LOB (mLobLocator, mLobOffset, mLobPieceLen, mLobPiece)

- mLobOffset is a relative position from the mLobOffset of LOB_PREPARE4WRITE XLog.

LOB_FINISH2WRITE XLog

Header (mType, mTID, mSN, mSyncSN)

LOB (mLobLocator)

KEEP_ALIVE XLog

Header (mType, mTID, mSN, mSyncSN)

REPL_STOP XLog

Header (mType, mTID, mSN, mSyncSN)

Meta Data

This section describes how to obtain the meta data that can be used to analyze XLog.

A user should invoke ALA_Handshake() before he/she can obtain the meta data.

Meta Data Structure

```
typedef struct ALA_ProtocolVersion
{
 UShort mMajor;                                /* Major Version */
 UShort mMinor;                                /* Minor Version */
 UShort mFix;                                  /* Fix Version */
} ALA_ProtocolVersion;
typedef struct ALA_Replication
{
 SChar mXLogSenderName[ALA_NAME_LEN];

/* XLog Sender Name */
 UInt mTableCount;                             /* Table Count */
 ALA_Table *mTableArray;                       /* Table Array */
} ALA_Replication;
typedef struct ALA_Table
{
 ULong mTableOID;                              /* Table OID */
 SChar mFromUserName[ALA_NAME_LEN];            /* (From) User Name */
 SChar mFromTableName[ALA_NAME_LEN];           /* (From) Table Name */
 SChar mToUserName[ALA_NAME_LEN];              /* (To) User Name */
 SChar mToTableName[ALA_NAME_LEN];             /* (To) Table Name */
 UInt mPKIndexID;                              /* Index ID of Primary Key */
 UInt mPKColumnCount;                          /* Primary Key Column Count */
 ALA_Column **mPKColumnArray;                  /* Primary Key Column Array */
 UInt mColumnCount;                            /* Column Count */
 ALA_Column *mColumnArray;                     /* Column Array */
 UInt mIndexCount;                             /* Index Count */
 ALA_Index *mIndexArray;                       /* Index Array */
} ALA_Table;
typedef struct ALA_Column
{
 UInt mColumnID;                               /* Column ID */
 SChar mColumnName[ALA_NAME_LEN];              /* Column Name */
 UInt mDataType;                               /* Column information Type */
 UInt mLanguageID;                             /* Column Language ID */
 UInt mSize;                                   /* Column Size */
 SInt mPrecision;                              /* Column Precision */
 SInt mScale;                                  /* Column Scale */
 ALA_BOOL mNotNull;                            /* Column Not Null? */
} ALA_Column;
typedef struct ALA_Index
{
 UInt mIndexID;                                /* Index ID */
 SChar mIndexName[ALA_NAME_LEN];               /* Index Name */
 ALA_BOOL mUnique;                             /* Index Unique? */
 UInt mColumnCount;                            /* Index Column Count */
 UInt *mColumnIDArray;                         /* Index Column ID Array */
} ALA_Index;
```

The meta data includes Protocol Version, Replication, Table, Column and Index.

The mPKColumnArray member of the ALA_Table structure is an array of the ALA_Column pointers.

ALTIBASE Internal Data Type

This section describes the format of the internal data of Altibase.

The column information (ALA_Column) can be obtained by invoking ALA_GetColumnInfo(), and the internal data (ALA_Value) can be obtained by using Log Analysis API for XLog.

The internal data value is the value member of the ALA_Value structure, and the length of the internal data value is the length member of the ALA_Value structure.

The mDataType value for ALA_Column can be used to determine the type of the internal data.

Table 3-1 ALTIBASE Internal Data Types

Category	Type of Internal Data	Constant
Number	FLOAT	6
	NUMERIC	2
	DOUBLE	8
	REAL	7
	BIGINT	(UInt)-5
	INTEGER	4
	SMALLINT	5
Date/Time	DATE	9
Character/Binary	CHAR	1
	VARCHAR	12
	NCHAR	(UInt)-8
	NVARCHAR	(UInt)-9
	BYTE	20001
	NIBBLE	20002
	BIT	(UInt)-7
	VARBIT	(UInt)-100
	BLOB	30
	CLOB	40
Spatial	GEOMETRY	10003

FLOAT, NUMERIC

Internal Structure

The internal data structure of FLOAT and NUMERIC are the same.

```
typedef struct mtdNumericType
{
 UChar length;           /* Length of (signExponent + mantissa) */
 UChar signExponent;     /* Sign and Exponent */
 UChar mantissa[1];      /* UChar Array (100 Base) */
} mtdNumericType;
```

ALA_GetInternalNumericInfo() can be used to obtain the sign and exponent.

Obtaining the sign from mtdNumericType

```
if(signExponent is 128 ~ 255)
{
 Sign = '+';
}
else /* if(signExponent is 0 ~ 127) */
{
 Sign = '-';
}
```

Obtaining the exponent from mtdNumericType

It is an exponent for a decimal number.

```
if(signExponent is 128 ~ 255)
{
 Exponent = ((SInt)(signExponent & 0x7F) - 64) * 2
 + ((mantissa[0] < 10) ? -1 : 0);
}
else                              /* if(signExponent is 0 ~ 127) */
{
 Exponent = (64 - (SInt)(signExponent & 0x7F)) * 2
 + ((mantissa[0] >= 90) ? -1 : 0);
}
```

Obtaining the mantissa string from mtdNumericType

Each UChar has a value between 0 and 99 (100-nary).

The result is a number between 0 and 1.

```
if(Sign is '+')
{
/* Example : 01 23 45 67 89 -> 0.123456789
/* 12 34 56 78 99 -> 0.1234567899
 */
 /* mantissa[0] */
 if(mantissa[0] < 10)
 {
 MantissaStr = mantissa[0];
 }
 else
```

```
 {
 MantissaStr = mantissa[0] / 10;
 MantissaStr = MantissaStr + mantissa[0] % 10;
 }
 /* mantissa[1] - mantissa[mLength - 1] */
 for(Index = 1; Index < mLength - 1; Index++)
 {
 MantissaStr = MantissaStr + mantissa[Index] / 10;
 MantissaStr = MantissaStr + mantissa[Index] % 10;
 }
}
else /* if(Sign is '-') */
{
 /* Example : 98 76 54 32 10 -> 0.123456789
 /* 09 87 65 43 21 -> 0.9012345678
 */
 /* mantissa[0] */
 if(mantissa[0] >= 90)
 {
 MantissaStr = MantissaStr + (99 - mantissa[0]);
 }
 else
 {
 MantissaStr = MantissaStr + (99 - mantissa[0]) / 10;
 MantissaStr = MantissaStr + (99 - mantissa[0]) % 10;
 }
 /* mantissa[1] - mantissa[mLength - 1] */
 for(Index = 1; Index < mLength - 1; Index++)
 {
 MantissaStr = MantissaStr + (99 - mantissa[Index]) / 10;
 MantissaStr = MantissaStr + (99 - mantissa[Index]) % 10;
 }
}
```

DOUBLE, REAL, BIGINT, INTEGER, SMALLINT

Internal Structure

Each type is mapping to a primitive data type.

```
typedef SDouble mtdDoubleType;                  /* DOUBLE */
typedef SFloat mtdRealType;                     /* REAL */
typedef SLong mtdBigintType;                    /* BIGINT */
typedef SInt mtdIntegerType;                    /* INTEGER */
typedef SShort mtdSmallintType;                 /* SMALLINT */
```

DATE

Internal Structure

There is only one internal data type available for time and date.

```
typedef struct mtdDateType
{
 SShort year;                           /* Year(16bit) */
 UShort mon_day_hour;                   /* Not Used(2bit), Month(4bit), */
                                        /* Day(5bit), Hour(5bit) */
 UInt min_sec_mic;                      /* Minute(6bit), Second(6bit), */
```

```
                                        /* MicroSec(20bit) */
} mtdDateType;
```

CHAR, VARCHAR, NCHAR, NVARCHAR, BYTE, NIBBLE, BIT, VARBIT, BLOB, CLOB

Internal Structure

Each data type has similar structure with one another.

```
typedef struct mtdCharType              /* CHAR, VARCHAR */
{
 UShort length;                         /* Length of value */
 UChar value[1];                        /* UChar Array */
} mtdCharType;
typedef struct mtdNcharType {           /*NCHAR, NVARCHAR */
 UShort length;                         /* Length of value */
 UChar value[1];                        /* UChar Array */
} mtdNcharType;
typedef struct mtdByteType              /* BYTE */
{
 UShort length;                         /* Length of value */
 UChar value[1];                        /* UChar Array */
} mtdByteType;
typedef struct mtdNibbleType            /* NIBBLE */
{
 UChar length;                          /* Length of Nibbles */
 UChar value[1];                        /* UChar Array */
} mtdNibbleType;
typedef struct mtdBitType               /* BIT, VARBIT */
{
 UInt length;                           /* Length of Bits */
 UChar value[1];                        /* UChar Array */
} mtdBitType;
typedef struct mtdLobType
{
 UInt length;                           /* Length of value */
 UChar value[1];                        /* UChar Array */
} mtdLobType;
typedef mtdLobType mtdBlobType;         /* BLOB */
typedef mtdLobType mtdClobType;         /* CLOB */
```

BLOB and CLOB are not supported by ALA_GetAltibaseText(), ALA_GetAltibaseSQL() and ALA_GetODBCCValue().

GEOMETRY

Internal Structure

For information on the data structure and handling for the GEOMETRY data, please refer to Altibase Application Development Spatial SQL User's Manual.

GEOMETRY is not supported by ALA_GetAltibaseText(), ALA_GetAltibaseSQL() and ALA_GetODBCCValue().

SAVEPOINT

If you specify SAVEPOINT, transactions executed until now are saved temporarily in their process.

SAVEPOINTs used in Altibase are classified as follows.

- Implicit Savepoint
- Explicit Savepoint
- PSM Savepoint

If you execute statements related to transactions, Implicit Savepoint is managed as the savepoint used internally and list. If you fail to do, only corresponding statements are used for partial rollback automatically.

Users can specify Explicit Savepoint, and this is managed as the list. Refer to SQL User's Manual for more details about Explicit Savvepoint.

PSM Savepoint is used internally when you execute PSM. Only PSM Savepoint used currently is managed. Refer to Stored Procedure User's Manual for more details about stored procedure.

Each SAVEPOINT is managed separately and users process it depending on situation.

Example

```
iSQL> CREATE TABLE T1 (I1 INTEGER PRIMARY KEY);
Create success.
iSQL> INSERT INTO T1 VALUES (2);
1 row inserted.
iSQL> CREATE OR REPLACE PROCEDURE PROC1
      2 AS
      3 BEGIN
      4     INSERT INTO T1 VALUES(1);
      5     SAVEPOINT EXPLICIT_SP;
      6     INSERT INTO T1 VALUES(2);
      7     INSERT INTO T1 VALUES(3);
      8 END;
      9 /
Create success.
iSQL> AUTOCOMMIT OFF;
Set autocommit off success.
iSQL> EXEC PROC1;
[ERR-11058 : The row already exists in a unique index.
0006 :     INSERT INTO T1 VALUES(2);
^                         ^    ]
iSQL> ROLLBACK TO SAVEPOINT EXPLICIT_SP;
Rollback success.
```

4 Log Analysis API

This chapter provides description on using the Log Analysis API component of Altibase Log Analyzer.

Log Analysis API is an API that is invoked by the user's application. It provides functions which receive XLog from XLog Sender and analyze it.

An argument which begins with aOut is an output argument.

The following sections describes functions that can be used in C/C++ languages.

ALA_InitializeAPI

Syntax

```
ALA_RC ALA_InitializeAPI(
    ALA_BOOL       aUseAltibaseODBCDriver,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aUseAltibaseODBCDriver	Whether to use the Altibase ODBC driver.
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Creates environment in which Log Analysis API can be invoked.

Note

ALA_ClearErrorMgr() should be called before any other Log Analyzer API is called.

If it fails, Log Analysis API cannot be used.

SQLAllocEnv() should be called before the ALTIBASE ODBC Driver is used in case of connecting thru ALTIBASE ODBC Driver

Related Function

ALA_DestroyAPI

Example

```
#include <sqlcli.h>
#include <alaAPI.h>
```

```
...

/* When the Altibase ODBC driver is not used */
void testAPIEnvironment1()
{
/* Create Log Analysis API Environment */
(void)ALA_InitializeAPI(ALA_FALSE, NULL);

/* Invoke Log Analysis API */
...

/* Remove Log Analysis API Environment */
(void)ALA_DestroyAPI(ALA_FALSE, NULL);
}

/* When the Altibase ODBC driver is used */
void testAPIEnvironment2(ALA_BOOL aUseAltibaseODBCDriver)
{
SQLHENV sEnv = NULL;

/* Create Altibase ODBC Environment */
(void)SQLAllocEnv(&sEnv);

/* Create Log Analysis API Environment */
(void)ALA_InitializeAPI(ALA_TRUE, NULL);

/* Invoke Altibase ODBC API and Log Analysis API */
...

/* Remove Log Analysis API Environment */
(void)ALA_DestroyAPI(ALA_TRUE, NULL);

/* Remove Altibase ODBC Environment */
(void)SQLFreeEnv(sEnv);
}
```

ALA_DestroyAPI

Syntax

```
ALA_RC ALA_DestroyAPI(
    ALA_BOOL       aUseAltibaseODBCDriver,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aUseAltibaseODBCDriver	Whether to use the Altibase ODBC driver.
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Removes environment in which Log Analysis API can be invoked.

Note

Regardless of the result, other Log Analysis API may not be used after invoking this function.

When the ALTIBASE ODBC Driver is used, the last SQLFreeEnv() should be used before it is invoked.

Related Function

ALA_InitializeAPI

Example

Please refer to ALA_InitializeAPI.

ALA_EnableLogging

Syntax

```
ALA_RC ALA_EnableLogging(
    const SChar  * aLogDirectory,
    const SChar  * aLogFileName,
    UInt           aFileSize,
    UInt           aMaxFileNumber,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aLogDirectory	Log Directory
aLogFileName	Log Filename
aFileSize	The size of a log file
aMaxFileNumber	The maximum number of previous log files
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Enables logging for problem tracking.

If it is not invoked, logging will not be performed.

If there is no log file, a new log file is created. If there is one, logs are appended to the existing log file.

If the size of a log file reaches aFileSize, the log file is renamed and a new log file is created. A log file is renamed by referring to 1~aMaxFileNumber in the header of the file. For example, if the header of the log file 'analysis.log' has number 1 and the size of the log file reaches aFileSize, its filename is changed from 'analysis.log' to 'analysis.log-1' and a new log file named 'analysis.log' is created.

The number of the log file header starting from 1 is incremented by 1. If it reaches aMaxFileNumber, incrementing starts over from 1. As a result, only the current log file in use and the last aMaxFile-Number log files are kept.

Note

Up to 1,024 combinations of log directory and log file names are allowed (including NULL).

It cannot be invoked when logging is already enabled.

Log Analysis API cannot be invoked while it is being invoked.

If the log file header is not normal, the log file is deleted and created again.

If aFileSize is 0, the log file can grow infinitely.

Related Function

ALA_DisableLogging

Example

```
#include <alaAPI.h>
...
void testLogging()
{
/* Create Log Analysis API Environment */
          (void)ALA_InitializeAPI(ALA_FALSE, NULL);


/* Enable logging
 * Log Directory                                  : The current directory
 * Log File Name                                  : analysis.log
 * The Size of Log File Size : 10 MB
 * The Max. Number of Previous Log Files : 10
 */
      (void)ALA_EnableLogging(".",
                              "analysis.log",
                               10 * 1024 * 1024,
                               10,
                               NULL);

      /* Invoke Log Analysis API */
         ...

      /* Disable logging */
         (void)ALA_DisableLogging(NULL);

      /* Remove Log Analysis API Environment */
         (void)ALA_DestroyAPI(ALA_FALSE, NULL);
}
```

ALA_DisableLogging

Syntax

```
ALA_RC ALA_DisableLogging(
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Disables logging.

Note

It cannot be invoked when logging is not enabled.

Log Analysis API cannot be invoked while it is being invoked.

Related Function

ALA_EnableLogging

Example

Please refer to ALA_EnableLogging.

ALA_CreateXLogCollector

Syntax

```
ALA_RC ALA_CreateXLogCollector(
    const SChar  * aXLogSenderName,
    const SChar  * aSocketInfo,
    SInt           aXLogPoolSize,
    ALA_BOOL       aUseCommittedTxBuffer,
    UInt           aACKPerXLogCount,
    ALA_Handle   * aOutHandle,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aXLogSenderName	The name of the corresponding XLog Sender (length: 1 - 40)
aSocketInfo	Socket Information (TCP, UNIX Domain)
aXLogPoolSize	The maximum size of XLog Pool (range: 1 -)
aUseCommittedTxBuffer	Whether to obtain Transaction XLog in the order of commit.
aACKPerXLogCount	The reference number of XLog for which ACK will be sent (range: 1 -)
aOutHandle	XLog Collector Handle
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Creates XLog Collector corresponding to XLog Sender.

The name of XLog Sender can be duplicated.

aSocketInfo is a string in the
“SOCKET=socket_type;PEER_IP=xlog_sender_ip;MY_PORT=listen_port” format.

- socket_type can only be specified to either of 'TCP' or 'UNIX.' If it is specified to 'UNIX,' the socket filename is automatically created in the '$ALTIBASE_HOME/trc/rp-replication_name' format.
- xlog_sender_ip is specified when socket_type is TCP. The valid length is between 1 – 39. This information is required for authentication of XLog Sender.
- listen_port is specified when socket_type is TCP. The valid range is between 1024 – 65535 (0xFFFF). This is a port number to which XLog Sender will connect. It is used by ALA_Handshake().

aACKPerXLogCount is applied when ALA_SendACK() is invoked.

Note

When the transaction XLog is obtained in the order of commit, the followings should be considered:

- Since XLog is accumulated in the transaction table until COMMIT XLog arrives, the size of XLog Pool should be set large enough. That is, it requires more memory space.
- XLog can be obtained after COMMIT XLog is received. That is, the time interval between the time that XLog arrives and the time that it is actually proccessed becomes larger. If a transaction is a bulk job, it is very likely to decrease performance.

If the reference number of XLog for which ACK will be sent out is set to a number bigger than 1, ACK may not be sent to XLog Sender even when ALA_SendACK() is invoked.

If the socket type is TCP, the listening port should not be a duplicate.

If the socket type is UNIX Domain, the same environment variable '$ALTIBASE_HOME' should be specified as in the Altibase DBMS to which XLog Sender belongs. Since the maximum allowable length of a socket filename varies depending on an operating system, the maximum allowable length should be checked so that it is not exceeded.

Related Function

ALA_AddAuthInfo

ALA_RemoveAuthInfo

ALA_DestroyXLogCollector

Example

```
#include <alaAPI.h>
     …
     void testXLogCollectorTCP()
     {
ALA_Handle sHandle;

/* Create XLog Collector that uses TCP
 * XLog Sender Name : log_analysis
 * XLog Sender Authentication Information : IP=127.0.0.1
 * Listening Port                                          : 30300
```

```
 * The max. size of XLog Pool                                    : 10000
 * Obtain transaction XLog in the order of commit                : Disabled
 * The reference number of XLog for which ACK will be sent out : 100
 */
          (void)ALA_CreateXLogCollector("log_analysis",

 "SOCKET=TCP;PEER_IP=127.0.0.1;MY_PORT=30300",
                                10000,
                                ALA_FALSE,
                                100,
                                &sHandle,
                                NULL);

 /* Adde XLog Sender Authentication Information */
   (void) ALA_AddAuthInfo(sHandle, "PEER_IP=127.0.0.2", NULL);

 /* Remove XLog Sender Authentication Information */
   (void)ALA_RemoveAuthInfo(sHandle, "PEER_IP=127.0.0.2", NULL);

 /* Invoke Log Analysis API */
…

 /* Remove XLog Collector */
   (void)ALA_DestroyXLogCollector(sHandle, NULL);
}

void testXLogCollectorUNIX()
 {
ALA_Handle sHandle;

/* Create XLog Collector that uses a UNIX domain
 * XLog Sender Name : log_analysis
 * The max. size of XLog Pool : 20000
 * Obtain transaction XLog in the order of commit : Enabled
 * The reference number of XLog for which ACK will be sent out : 50
 */
        (void)ALA_CreateXLogCollector("log_analysis",
                                     "SOCKET=UNIX",
                                      20000,
                                      ALA_TRUE,
                                      50,
                                      &sHandle,
                                      NULL);
/* Invoke Log Analysis API */

…
/* Remove XLog Collector */
 (void)ALA_DestroyXLogCollector(sHandle, NULL);
 }
```

ALA_AddAuthInfo

Syntax

```
ALA_RC ALA_AddAuthInfo(
    ALA_Handle     aHandle,
    const SChar  * aAuthInfo,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aAuthInfo	XLog Sender Authentication Information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Adds the authentication information for XLog Sender.

For TCP, aAuthInfo is a string in the "PEER_IP=*xlog_sender_ip*" format. The valid length for *xlog_sender_ip* is between 1 - 39.

Note

It can used only when the socket type is set to TCP.

Up to 32 pieces of authentication information can be specified for XLog Sender.

Related Function

ALA_CreateXLogCollector

ALA_RemoveAuthInfo

ALA_Handshake

Example

Please refer to ALA_CreateXLogCollector.

ALA_RemoveAuthInfo

Syntax

```
ALA_RC ALA_RemoveAuthInfo(
    ALA_Handle     aHandle,
    const SChar  * aAuthInfo,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aAuthInfo	XLog Sender Authentication Information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Removes the authentication information for XLog Sender.

For TCP, aAuthInfo is a string in the “PEER_IP=xlog_sender_ip” format. The valid length for xlog_sender_ip is between 1 – 39.

Note

It can used only when the socket type is set to TCP.

At least one piece of authentication information is required for XLog Sender.

Related Function

ALA_CreateXLogCollector

ALA_AddAuthInfo

ALA_Handshake

Example

Please refer to ALA_CreateXLogCollector.

ALA_SetHandshakeTimeout

Syntax

```
ALA_RC ALA_SetHandshakeTimeout(
    ALA_Handle     aHandle,
    UInt           aSecond,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aSecond	Handshake Timeout (unit: second, range: 1 - 0xFFFFFFFE)
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Sets the handshake timeout.

The handshake timeout applies when ALA_Handshake() is invoked.

The default handshake timeout is 600 seconds.

Related Function

ALA_Handshake

Example

Please refer to ALA_Handshake.

ALA_SetReceiveXLogTimeout

Syntax

```
ALA_RC ALA_SetReceiveXLogTimeout(
    ALA_Handle     aHandle,
    UInt           aSecond,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aSecond	XLog Receive Timeout (unit: second, range: 1 - 0xFFFFFFFE)
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Sets the XLog Receive timeout.

The XLog Receive timeout applies when ALA_ReceiveXLog() is invoked. The default XLog Receive timeout is 10 seconds.

Related Function

ALA_ReceiveXLog

Example

Please refer to ALA_Handshake.

ALA_Handshake

Syntax

```
ALA_RC ALA_Handshake(
    ALA_Handle     aHandle,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Listens for XLog Sender and performs handshaking.

The user settings and all data in XLog Collector excluding XLog Pool are initialized. The meta data is received and stored internally.

Note

If TCP is used, it fails when there is no matching authentication information.

It fails if the connected peer is not XLog Sender.

If XLog Sender is not connected within the handshake timeout, a timeout will occur.

ALA_ReceiveXLog(), ALA_GetXLog() and ALA_SendACK() should not be invoked before handshaking is completed.

Before handshaking is started, ALA_FreeXLog() should be performed on all XLog obtained via ALA_GetXLog() so that XLog Pool is not depleted.

Related Function

ALA_AddAuthInfo

ALA_RemoveAuthInfo

ALA_SetHandshakeTimeout

ALA_ReceiveXLog

ALA_SendACK

ALA_GetReplicationInfo

ALA_GetTableInfo

ALA_GetColumnInfo

ALA_GetIndexInfo

Example

```
#include <alaAPI.h>
…

void testXLogCollector(ALA_Handle aHandle)
{
ALA_XLog                * sXLog                 = NULL;
ALA_XLogHeader          * sXLogHeader           = NULL;
ALA_XLogCollectorStatus   sXLogCollectorStatus
ALA_BOOL                  sInsertXLogInQueue    = ALA_FALSE;
ALA_BOOL                  sExitFlag             = ALA_FALSE;

/* Set Handshake Timeout : 600 seconds */
(void)ALA_SetHandshakeTimeout(aHandle, 600, NULL);

/* Set XLog Receive Timeout : 10 seconds */
(void)ALA_SetReceiveXLogTimeout(aHandle, 10, NULL);

/* Listen for XLog Sender and Handshake */
(void)ALA_Handshake(aHandle, NULL);

/* Receive XLog until XLog Sender ends */
while(sExitFlag != ALA_TRUE)
{
/* Receive XLog and add it to XLog Queue */
sInsertXLogInQueue = ALA_FALSE;
while(sInsertXLogInQueue != ALA_TRUE)
{
 (void)ALA_ReceiveXLog(aHandle, &sInsertXLogInQueue, NULL);
}

/* Obtain XLog from XLog Queue.
 * Assuming that transaction XLog is obtained in the order in which records
are logged.
 */
(void)ALA_GetXLog(aHandle, &sXLog, NULL);

/* Analyze and Process XLog */
(void)ALA_GetXLogHeader(sXLog, &sXLogHeader, NULL);
```

```
if(sXLogHeader->mType == XLOG_TYPE_REPL_STOP)
{
sExitFlag = ALA_TRUE;
}
…

/* Send ACK to XLog Sender */
(void)ALA_SendACK(aHandle, NULL);

/* Return XLog to XLog Pool */
(void)ALA_FreeXLog(aHandle, sXLog, NULL);

/* Obtain the Status of XLog Collector */
(void)ALA_GetXLogCollectorStatus(aHandle,
 &sXLogCollectorStatus,
NULL);
}
}
```

ALA_ReceiveXLog

Syntax

```
ALA_RC ALA_ReceiveXLog(
    ALA_Handle     aHandle,
    ALA_BOOL     * aOutInsertXLogInQueue,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutInsertXLogInQueue	Whether XLog has been added to XLog Pool.
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Receives XLog and adds it to XLog Queue.

Obtains XLog from XLog Pool.

When transaction XLog is obtained in the order of commit, it is stored in the transaction table until COMMIT XLog is received.

It can be invoked together with ALA_GetXLog() at a time.

Note

ALA_Handshake() should be invoked first. It fails if a network error has already occurred.

If XLog is not received within the XLog Receive timeout, a timeout will occur.

It fails if there is no XLog available in XLog Pool.

When transaction XLog is obtained in the order of commit, it may not be added to XLog Queue even if it is received.

If a network error occurs or REPL_STOP XLog is received, the transactions that have been obtained via ALA_GetXLog() but not committed should be rolled back.

Memory for aOutInsertXLogInQueue should be allocated in advance.

Related Function

ALA_SetReceiveXLogTimeout

ALA_Handshake

ALA_GetXLog

ALA_SendACK

ALA_FreeXLog

Example

Please refer to ALA_Handshake.

ALA_GetXLog

Syntax

```
ALA_RC ALA_GetXLog(
    ALA_Handle        aHandle,
    const ALA_XLog ** aOutXLog,
    ALA_ErrorMgr    * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutXLog	XLog obtained from XLog Queue.
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains XLog from XLog Queue.

It can be invoked together with ALA_ReceiveXLog() at a time.

If there is no XLog in XLog Queue, aOutXLog is NULL.

Note

XLog is managed by a user until ALA_FreeXLog() is invoked.

Related Function

ALA_ReceiveXLog

ALA_FreeXLog

ALA_GetXLogHeader

ALA_GetXLogPrimaryKey

ALA_GetXLogColumn

ALA_GetXLogSavepoint

ALA_GetXLogLOB

Example

Please refer to ALA_Handshake.

ALA_SendACK

Syntax

```
ALA_RC ALA_SendACK(
    ALA_Handle     aHandle,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Sends ACK to XLog Sender.

If ALA_GetXLog() succeeds for more than the reference number of XLog for which ACK will be sent out, or there is a record showing that KEEP_ALIVE or REPL_STOP XLog has been received, ACK is sent to XLog Sender.

If there is REPL_STOP XLog received, a network connection will be disconnected.

ACK contains Restart SN. Restart SN is determined as the one with the minimum value among the following SN's:

- The minimum BEGIN XLog SN for the active transactions checked at the point when ALA_GetXLog() is invoked lastly.
- The SN for the last XLog obtained with ALA_GetXLog() if there is no active transaction,
- The minimum BEGIN XLog SN for uncommitted transactions that are stored in the transaction table if transaction XLog is obtained in the order of commit.

Note

It affects the meta table and flush of XLog Sender. If ACK is not sent within the

REPLICATION_RECEIVE_TIMEOUT property of XLog Sender, XLog Sender drops the network connection. And also, if it is not invoked for a long time, XLog Sender may stop sending XLog, updates Restart SN with the SN for the most recently recorded log, and start over.

Since Restart SN in XLog Sender can be updated, all XLog obtained with ALA_GetXLog() should be processed before ALA_SendACK() is invoked.

If a network error occurs, XLog Sender attempts to handshake periodically. If it succeeds, it sends XLog, begining with Restart SN.

Related Function

ALA_Handshake

ALA_ReceiveXLog

Example

Please refer to ALA_Handshake.

ALA_FreeXLog

Syntax

```
ALA_RC ALA_FreeXLog(
    ALA_Handle     aHandle,
    ALA_XLog     * aXLog,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aXLog	XLog that will be returned to XLog Pool.
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Returns XLog to XLog Pool.

Note

If the XLog obtained with ALA_GetXLog() is not returned overlong, XLog Pool may get depleted.

Related Function

ALA_ReceiveXLog

ALA_GetXLog

Example

Please refer to ALA_Handshake.

ALA_DestroyXLogCollector

Syntax

```
ALA_RC ALA_DestroyXLogCollector(
    ALA_Handle     aHandle,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Removes XLog Collector.

Note

Before it is invoked, , ALA_FreeXLog() should be performed on all XLog obtained via ALA_GetXLog().

Regardless of the result, the Log Analysis API that is related to the corresponding XLog Collector should not be invoked.

Related Function

ALA_CreateXLogCollector

Example

Please refer to ALA_CreateXLogCollector.

ALA_GetXLogCollectorStatus

Syntax

```
ALA_RC ALA_GetXLogCollectorStatus(
    ALA_Handle                aHandle,
    ALA_XLogCollectorStatus * aOutXLogCollectorStatus,
    ALA_ErrorMgr            * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutXLogCollectorStatus	The status structure of XLog Collector
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the status of XLog Collector.

mMyIP, mMyPort, mPeerIP, mPeerPort and mSocketFile are updated with ALA_Handshake().

mXLogCountInPool is incremented/decremented with ALA_ReceiveXLog() and it is decremented with ALA_FreeXLog().

mLastArrivedSN is the SN for the last XLog received via ALA_ReceiveXLog(). mLastProcessedSN is the SN for the last XLog received via ALA_GetXLog().

Since mNetworkValid indicates the network validity, it can be changed when ALA_Handshake(), ALA_ReceiveXLog() or ALA_SendACK() is invoked.

The status structure of XLog Collector can be defined as follows:

```
typedef struct ALA_XLogCollectorStatus
{
 SChar mMyIP[ALA_IP_LEN];
    SInt mMyPort;
    SChar mPeerIP[ALA_IP_LEN];
    SInt mPeerPort;
```

```
    SChar mSocketFile[ALA_SOCKET_FILENAME_LEN];
    UInt mXLogCountInPool;
    ALA_SN mLastArrivedSN;
    ALA_SN mLastProcessedSN;
    ALA_BOOL mNetworkValid;
} ALA_XLogCollectorStatus;
```

Structure Member	Description
mMyIP	[TCP] XLog Collector IP
mMyPort	[TCP] XLog Collector Port
mPeerIP	[TCP] XLog Sender IP
mPeerPort	[TCP] XLog Sender Port
mSocketFile	[UNIX Domain] Socket File Name
mXLogCountInPool	The number of XLog remaining in XLog Pool.
mLastArrivedSN	The SN for the last XLog received.
mLastProcessedSN	The SN for the last XLog processed.
mNetworkValid	The network validity

Note

Memory for aOutXLogCollectorStatus should be allocated in advance.

Related Function

ALA_Handshake

ALA_ReceiveXLog

ALA_GetXLog

ALA_SendACK

ALA_FreeXLog

Example

Please refer to ALA_Handshake.

ALA_GetXLogHeader

Syntax

```
ALA_RC ALA_GetXLogHeader(
    const ALA_XLog        * aXLog,
    const ALA_XLogHeader ** aOutXLogHeader,
    ALA_ErrorMgr          * aOutErrorMgr);
```

Argument

Argument	Description
aXLog	XLog
aOutXLogHeader	XLog Header
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the XLog header from XLog.

Related Function

ALA_GetXLog

ALA_GetXLogPrimaryKey

ALA_GetXLogColumn

ALA_GetXLogSavepoint

ALA_GetXLogLOB

Example

```
#include <alaAPI.h>
…
```

```
void testXLogGetPart(const ALA_XLog * aXLog)
{
     ALA_XLogHeader                 * sXLogHeader = NULL;
     ALA_XLogPrimaryKey             * sXLogPrimaryKey = NULL;
     ALA_XLogColumn                 * sXLogColumn = NULL;
     ALA_XLogSavepoint              * sXLogSavepoint = NULL;
     ALA_XLogLOB                    * sXLogLOB = NULL;

     /* Obtain XLog Header */
      (void)ALA_GetXLogHeader(aXLog, &sXLogHeader, NULL);

     /* Obtain XLog Primary Key */
      (void)ALA_GetXLogPrimaryKey(aXLog, &sXLogPrimaryKey, NULL);

     /* Obtain XLog Column */
      (void)ALA_GetXLogColumn(aXLog, &sXLogColumn, NULL);

     /* Obtain XLog Savepoint */
      (void)ALA_GetXLogSavepoint(aXLog, &sXLogSavepoint, NULL);

     /* Obtain XLog LOB */
      (void)ALA_GetXLogLOB(aXLog, &sXLogLOB, NULL);
}
```

ALA_GetXLogPrimaryKey

Syntax

```
ALA_RC ALA_GetXLogPrimaryKey(
    const ALA_XLog            * aXLog,
    const ALA_XLogPrimaryKey ** aOutXLogPrimaryKey,
    ALA_ErrorMgr              * aOutErrorMgr);
```

Argument

Argument	Description
aXLog	XLog
aOutXLogPrimaryKey	XLog Primary Key
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the XLog header from XLog.

Related Function

ALA_GetXLog

ALA_GetXLogHeader

ALA_GetXLogColumn

ALA_GetXLogSavepoint

ALA_GetXLogLOB

Example

Please refer to ALA_GetXLogHeader.

ALA_GetXLogColumn

Syntax

```
ALA_RC ALA_GetXLogColumn(
    const ALA_XLog        * aXLog,
    const ALA_XLogColumn ** aOutXLogColumn,
    ALA_ErrorMgr          * aOutErrorMgr);
```

Argument

Argument	Description
aXLog	XLog
aOutXLogColumn	XLog Column
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the XLog column from XLog.

Related Function

ALA_GetXLog

ALA_GetXLogHeader

ALA_GetXLogPrimaryKey

ALA_GetXLogSavepoint

ALA_GetXLogLOB

Example

Please refer to ALA_GetXLogHeader.

ALA_GetXLogSavepoint

Syntax

```
ALA_RC ALA_GetXLogSavepoint(
    const ALA_XLog           * aXLog,
    const ALA_XLogSavepoint ** aOutXLogSavepoint,
    ALA_ErrorMgr             * aOutErrorMgr);
```

Argument

Argument	Description
aXLog	XLog
aOutXLogSavepoint	XLog Savepoint
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the XLog savepoint from XLog.

Related Function

ALA_GetXLog

ALA_GetXLogHeader

ALA_GetXLogPrimaryKey

ALA_GetXLogColumn

ALA_GetXLogLOB

Example

Please refer to ALA_GetXLogHeader.

ALA_GetXLogLOB

Syntax

```
ALA_RC ALA_GetXLogLOB(
    const ALA_XLog     * aXLog,
    const ALA_XLogLOB ** aOutXLogLOB,
    ALA_ErrorMgr       * aOutErrorMgr);
```

Argument

Argument	Description
aXLog	XLog
aOutXLogLOB	XLog LOB
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the XLog LOB from XLog.

Related Function

ALA_GetXLog

ALA_GetXLogHeader

ALA_GetXLogPrimaryKey

ALA_GetXLogColumn

ALA_GetXLogSavepoint

Example

Please refer to ALA_GetXLogHeader.

ALA_GetProtocolVersion

Syntax

```
ALA_RC ALA_GetProtocolVersion(
    const ALA_ProtocolVersion * aOutProtocolVersion,
    ALA_ErrorMgr              * aOutErrorMgr);
```

Argument

Argument	Description
aOutProtocolVersion	Protocol Version Structure
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the protocol version of Log Analysis API.

It can be obtained regardless of handshaking.

Note

Memory for aOutProtocolVersion should be allocated in advance.

Example

```
#include <alaAPI.h>
…

void testProtocolVersion()
{
ALA_ProtocolVersion sProtocolVersion;

/* Obtain Protocol Version */
(void)ALA_GetProtocolVersion(&sProtocolVersion, NULL);
}
```

ALA_GetReplicationInfo

Syntax

```
ALA_RC ALA_GetReplicationInfo(
    ALA_Handle                aHandle,
    const ALA_Replication ** aOutReplication,
    ALA_ErrorMgr           * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aOutReplication	Replication Information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the Replication information.

Note

If there is no corresponding meta data, it returns NULL as an argument.

The meta data obtained before handshaking should not be used again and again.

Related Function

ALA_Handshake

ALA_GetTableInfo

ALA_GetTableInfoByName

ALA_GetColumnInfo

Example

```
#include <alaAPI.h>
…

void testMetaInformation(ALA_Handle aHandle)
{
ALA_Replication * sReplication      = NULL;
ALA_Table       * sTable            = NULL;
ALA_Table       * sTableByTableOID  = NULL;
ALA_Table       * sTableByName      = NULL;
ALA_Column      * sPKColumn         = NULL;
ALA_Column      * sColumn           = NULL;
ALA_Index       * sIndex            = NULL;
UInt sTablePos;
UInt sPKColumnPos;
UInt sColumnPos;
UInt sIndexPos;

/* Obtain replication information */
(void)ALA_GetReplicationInfo(aHandle, &sReplication, NULL);
for(sTablePos = 0; sTablePos &lt; sReplication->mTableCount; sTablePos++)
{
sTable = &(sReplication->mTableArray[sTablePos]);

/* Obtain table information by table OID */
(void)ALA_GetTableInfo(aHandle,
sTable->mTableOID,
&sTableByTableOID,
NULL);

if(sTableByTableOID != sTable)
{
/* Fatal Error : Error in Log Analysis API */
break;
}

/* Obtain table information by name */
(void)ALA_GetTableInfoByName(aHandle,
sTable->mFromUserName,
sTable->mFromTableName,
&sTableByName,
NULL);

if(sTableByName != sTable)
{
/* Fatal Error : Error in Log Analysis API */
break;
}

/* Process primary key column */
for(sPKColumnPos = 0; sPKColumnPos < sTable->mPKColumnCount; sPKColumnPos++)
{
/* Obtain the primary key column information by primary key column ID */

(void)ALA_GetColumnInfo(sTable,
sTable->mPKColumnArray[sPKColumnPos]->mColumnID,
&sPKColumn,
NULL);
```

```
if(sPKColumn != sTable->mPKColumnArray[sPKColumnPos])
{
/* Fatal Error : Error in Log Analysis API */
break;
}

/* Process primary key column */
…
}

/* Process column */

for(sColumnPos = 0; sColumnPos < sTable->mColumnCount; sColumnPos++)
{
/* Obtain column information by column ID */
(void)ALA_GetColumnInfo(sTable,
sTable->mColumnArray[sColumnPos].mColumnID,
&sColumn,
NULL);
if(sColumn != &(sTable->mColumnArray[sColumnPos]))
{
/* Fatal Error : Error in Log Analysis API */
break;
}

/* Process column */
…
}

/* Process Index */
for(sIndexPos = 0; sIndexPos < sTable->mIndexCount; sIndexPos++)
{
   /* Obtain index information by index ID */
   (void)ALA_GetIndexInfo(sTable,
sTable->mIndexArray[sIndexPos].mIndexID,
&sIndex,
NULL);

 if(sIndex != &(sTable->mIndexArray[sIndexPos]))
 {
/* Fatal Error : Error in Log Analysis API */
break;
 }

 /* Process index */
 …
}
}
}
```

ALA_GetTableInfo

Syntax

```
ALA_RC ALA_GetTableInfo(
    ALA_Handle          aHandle,
    ULong               aTableOID,
    const ALA_Table  ** aOutTable,
    ALA_ErrorMgr      * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aTableOID	The table OID that will be searched for.
aOutTable	Table information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Retrieves the table information by table OID.

Note

If there is no corresponding meta data, it returns NULL as an argument.

The meta data obtained before handshaking should not be used again and again.

Related Function

ALA_Handshake

ALA_GetReplicationInfo

ALA_GetTableInfoByName

ALA_GetColumnInfo

ALA_GetIndexInfo

Example

Please refer to ALA_GetReplicationInfo.

ALA_GetTableInfoByName

Syntax

```
ALA_RC ALA_GetTableInfoByName(
    ALA_Handle         aHandle,
    const SChar      * aFromUserName,
    const SChar      * aFromTableName,
    const ALA_Table ** aOutTable,
    ALA_ErrorMgr     * aOutErrorMgr);
```

Argument

Argument	Description
aHandle	XLog Collector Handle
aFromUserName	The From User Name that will be searched for
aFromTableName	The From Table Name that will be searched for
aOutTable	Table information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Retrieves the table information by user name and table name.

Note

If there is no corresponding meta data, it returns NULL as an argument.

The meta data obtained before handshaking should not be used again and again.

Related Function

ALA_Handshake

ALA_GetReplicationInfo

ALA_GetTableInfo

ALA_GetColumnInfo

ALA_GetIndexInfo

Example

Please refer to ALA_GetReplicationInfo.

ALA_GetColumnInfo

Syntax

```
ALA_RC ALA_GetColumnInfo(
    const ALA_Table   * aTable,
    UInt                aColumnID,
    const ALA_Column ** aOutColumn,
    ALA_ErrorMgr      * aOutErrorMgr);
```

Argument

Argument	Description
aTable	Target Table information
aColumnID	The column ID that will be searched for.
aOutColumn	Column information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Retrieves the column information in a table by Column ID.

Note

If there is no corresponding meta data, it returns NULL as an argument.

The meta data obtained before handshaking should not be used again and again.

Related Function

ALA_Handshake

ALA_GetReplicationInfo

ALA_GetTableInfo

ALA_GetTableInfoByName

ALA_GetIndexInfo

Example

Please refer to ALA_GetReplicationInfo.

ALA_GetIndexInfo

Syntax

```
ALA_RC ALA_GetIndexInfo(
    const ALA_Table  * aTable,
    UInt               aIndexID,
    const ALA_Index ** aOutIndex,
    ALA_ErrorMgr     * aOutErrorMgr);
```

Argument

Argument	Description
aTable	Target Table information
aIndexID	The index ID that will be searched for.
aOutIndex	Index information
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Retrieves the index information in a table by Index ID.

Note

If there is no corresponding meta data, it returns NULL as an argument.

The meta data obtained before handshaking should not be used again and again.

Related Function

ALA_Handshake

ALA_GetReplicationInfo

ALA_GetTableInfo

ALA_GetTableInfoByName

ALA_GetColumnInfo

Example

Please refer to ALA_GetReplicationInfo.

ALA_GetInternalNumericInfo

Syntax

```
ALA_RC ALA_GetInternalNumericInfo(
    ALA_Column   * aColumn,
    ALA_Value    * aAltibaseValue,
    SInt         * aOutSign,
    SInt         * aOutExponent,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aColumn	The column information for the internal data of Altibase
aAltibaseValue	The internal data of Altibase
aOutSign	Sign
aOutExponent	Exponent
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the sign and exponent of FLOAT and NUMERIC.

If aOutSign is 1, it represents a positive number. If it is 0, it represents a negative number.

aOutExponent is an exponent for a decimal number.

Example

```
#include <alaAPI.h>
…

void testInternalNumeric(ALA_Column * aColumn, ALA_Value * aAltibaseValue)
{
```

```
SInt sNumericSign;
SInt sNumericExponent;

     /* Obtain the internal numeric information */
(void)ALA_GetInternalNumericInfo(aColumn,
aAltibaseValue,
&sNumericSign,
&sNumericExponent,
NULL);
}
```

ALA_GetAltibaseText

Syntax

```
ALA_RC ALA_GetAltibaseText(
    ALA_Column   * aColumn,
    ALA_Value    * aValue,
    UInt           aBufferSize,
    SChar        * aOutBuffer,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aColumn	The column information for the internal data of Altibase
aValue	The internal data value of Altibase
aBufferSize	Buffer Size
aOutBuffer	Buffer
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Converts the internal data of Altibase to a string.

For time/date type, the date format is 'YYYY-MM-DD HH:MI:SS.SSSSSS'.

BIT is converted to the BIT'*value*' format and VARBIT is converted to the VARBIT'*value*' format.

Simple Rules

You should specify the value of national character set as ALTIBASE_ALA_NCHARSET that is an environment variable in the server where you want to analyze logs if using NCHAR or NVARCHAR typed data.

ex) If character set of NCHAR and NVARCHAR is set to UTF8 in target server,

```
export ALTIBASE_ALA_NCHARSET=UTF8;
```

ALA prints it as UCS-2(UTF-16) by calling aOutBuffer parameter of ALA_GetAltibaseText function.

```
ex) \0031\0020\0020\0020
```

BLOB, CLOB and GEOMETRY are not supported.

Related Function

ALA_GetAltibaseSQL

Example

```
#include <alaAPI.h>
…

void testAltibaseText(ALA_Table * aTable, ALA_XLog * aXLog)
{
SChar      sBuffer[1024];

/* Obtain Altibase SQL */
(void)ALA_GetAltibaseSQL(aTable,
aXLog,
1024,
sBuffer,
NULL);
}
```

ALA_GetAltibaseSQL

Syntax

```
ALA_RC ALA_GetAltibaseSQL(
    ALA_Table    * aTable,
    ALA_XLog     * aXLog,
    UInt           aBufferSize,
    SChar        * aOutBuffer,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aTable	The table information for XLog
aXLog	XLog
aBufferSize	Buffer Size
aOutBuffer	Buffer
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Converts transaction XLog to an Altibase SQL string.

For time/date type, the date format is 'YYYY-MM-DD HH:MI:SS.SSSSSS'.

BIT is converted to the BIT'*value*' format and VARBIT is converted to the VARBIT'*value*' format. NCHAR and NVARCHAR is converted to the UNISTR('value') format.

When the Insert, Update or Delete SQL is created, the target table is specified using mToUserName and mToTableName in aTable.

Note

BLOB, CLOB and GEOMETRY are not supported.

It may not be compatible with SQL for other DBMS.

Related Function

ALA_GetAltibaseText

Example

```
#include <alaAPI.h>
…

void testAltibaseSQL(ALA_Table * aTable, ALA_XLog * aXLog)
{
ALA_Column  * sColumn;
SChar         sBuffer[1024];
UInt          sPKColumnPos;
UInt          sColumnPos;

/* Process the primary key column */
for(sPKColumnPos = 0;
sPKColumnPos < aXLog->mPrimaryKey.mPKColCnt;
sPKColumnPos++)
{
/* The primary key sequence for XLog and the primary key sequence for the
table are the same */
sColumn = aTable->mPKColumnArray[sPKColumnPos];

/* Obtain the Altibase text */
(void)ALA_GetAltibaseText(sColumn,
&(aXLog->mPrimaryKey.mPKColArray[sPKColumnPos]),
1024,
sBuffer,
NULL);
}

/* Process column */
for(sColumnPos = 0; sColumnPos < aXLog->mColumn.mColCnt; sColumnPos++)
{
/* Obtain the column information */
(void)ALA_GetColumnInfo(aTable,
aXLog->mColumn.mCIDArray[sColumnPos],
&sColumn,
NULL);

/* Obtain the Altibase text for Before Image */
(void)ALA_GetAltibaseText(sColumn,
&(aXLog->mColumn.mBColArray[sColumnPos]),
1024,
sBuffer,
NULL);

/* Obtain the Altibase text for After Image */
(void)ALA_GetAltibaseText(sColumn,
&(aXLog->mColumn.mAColArray[sColumnPos]),
1024,
sBuffer,
NULL);
}
}
```

ALA_GetODBCCValue

Syntax

```
ALA_RC ALA_GetODBCCValue(
    ALA_Column   * aColumn,
    ALA_Value    * aAltibaseValue,
    SInt           aODBCCTypeID,
    UInt           aODBCCValueBufferSize,
    void         * aOutODBCCValueBuffer,
    ALA_BOOL     * aOutIsNull,
    UInt         * aOutODBCCValueSize,
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aColumn	The column information for the internal data of Altibase
aAltibaseValue	The internal data value of Altibase
aODBCCTypeID	The ODBC C type to convert
aODBCCValueBufferSize	The size of the result buffer
aOutODBCCValueBuffer	The result buffer
aOutIsNull	Whether the internal data of Altibase is NULL.
aOutODBCCValueSize	The size of the ODBC C value converted
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Convert the internal data of Altibase to the ODBC C value.

Table 4-1 ODBC C Conversion

ALTIBASEInternal Data / ODBC C Value	SQL_C_CHAR	SQL_C_NUMERIC	SQL_C_BIT	SQL_C_STINYINTSQL_C_UTINYINT	SQL_C_SSHORTSQL_C_USHORT	SQL_C_SLONGSQL_C_ULONG	SQL_C_SBIGINTSQL_C_UBIGINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP
FLOAT	O	O	O	O	O	O	O	O	O	O	
NUMERIC	O	O	O	O	O	O	O	O	O	O	
DOUBLE	O	O	O	O	O	O	O	O	O	O	
REAL	O	O	O	O	O	O	O	O	O	O	
BIGINT	O	O	O	O	O	O	O	O	O	O	
INTEGER	O	O	O	O	O	O	O	O	O	O	
SMALLINT	O	O	O	O	O	O	O	O	O	O	
DATE	O									O	O
CHAR	O	O	O	O	O	O	O	O	O	O	O
VARCHAR	O	O	O	O	O	O	O	O	O	O	O
NCHAR	O	O	O	O	O	O	O	O	O	O	O
NVARCHAR	O	O	O	O	O	O	O	O	O	O	O
BYTE	O									O	
NIBBLE	O									O	
BIT	O	O	O	O	O	O	O	O	O	O	
VARBIT	O	O	O	O	O	O	O	O	O	O	

Note

BLOB, CLOB and GEOMETRY are not supported.

ODBC 3.0 or later is supported.

Example

```
#include <alaAPI.h>
```

```
…

void testODBCCConversion(ALA_Table * aTable, ALA_XLog * aXLog)
{
    ALA_Column * sColumn;
    SChar sBuffer[1024];
    ALA_BOOL sIsNull;
    UInt sODBCCValueSize;
    UInt sPKColumnPos;

    /* Convert the primary key to SQL_C_CHAR */
    for(sPKColumnPos = 0;
        sPKColumnPos < aXLog->mPrimaryKey.mPKColCnt;
        sPKColumnPos++)
    {
    /* The primary key sequence for XLog and the primary key sequence for the
table are the same */
    sColumn = aTable->mPKColumnArray[sPKColumnPos];
    /* Convert the internal data to SQL_C_CHAR */
    (void)ALA_GetODBCCValue(sColumn,
        &(aXLog->mPrimaryKey.mPKColArray[sPKColumnPos],
        SQL_C_CHAR,
        1024,
        sBuffer,
        &sIsNull,
        &sODBCCValueSize,
        NULL);
    }
}
```

ALA_ClearErrorMgr

Syntax

```
ALA_RC ALA_ClearErrorMgr(
    ALA_ErrorMgr * aOutErrorMgr);
```

Argument

Argument	Description
aOutErrorMgr	Error Manager

Result

ALA_SUCCESS

ALA_FAILURE

Description

Initializes Error Manager.

Note

Before Error Manager is used for the first time, it should be initialized.

Related Function

ALA_GetErrorCode

ALA_GetErrorLevel

ALA_GetErrorMessage

Example

```
#include <alaAPI.h>
…

void testErrorHandling()
{
ALA_ErrorMgr sErrorMgr
UInt sErrorCode;
```

```
ALA_ErrorLevel sErrorLevel;
SChar * sErrorMessage;
/* Initialize Error Manager */
(void)ALA_ClearErrorMgr(&sErrorMgr);

/* Invoking of Log Analysis API fails*/
…

/* Obtain the error code */
(void)ALA_GetErrorCode(&sErrorMgr, &sErrorCode);

/* Obtain the error level */
(void)ALA_GetErrorLevel(&sErrorMgr, &sErrorLevel);

/* Obtain the error message */
(void)ALA_GetErrorMessage(&sErrorMgr, &sErrorMessage);
}
```

ALA_GetErrorCode

Syntax

```
ALA_RC ALA_GetErrorCode(
    const ALA_ErrorMgr * aErrorMgr,
    UInt               * aOutErrorCode);
```

Argument

Argument	Description
aErrorMgr	Error Manager
aOutErrorCode	Error Code

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains an error code.

An error code is a unique value that identifies an error.

Note

Memory for aOutErrorCode should be allocated in advance.

Since mErrorCode in the ALA_ErrorMgr structure contains internal data, an error Code should be obtained via ALA_GetErrorCode().

Related Function

ALA_ClearErrorMgr

ALA_GetErrorLevel

ALA_GetErrorMessage

Example

Please refer to ALA_ClearErrorMgr.

ALA_GetErrorLevel

Syntax

```
ALA_RC ALA_GetErrorLevel(
    const ALA_ErrorMgr * aErrorMgr,
    ALA_ErrorLevel     * aOutErrorLevel);
```

Argument

Argument	Description
aErrorMgr	Error Manager
aOutErrorLevel	Error Level

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains the error level.

Since ALA_ERROR_FATAL is a fatal error, ALA_DestroyXLogCollector() should be invoked to terminate the corresponding XLog Collector.

Since ALA_ERROR_ABORT indicates that XLog Collector is in an abnormal status, ALA_Handshake() should be invoked to perform handshaking again for the corresponding XLog Collector.

ALA_ERROR_INFO indicates that invoking Log Analysis API has failed. An appropriate action should be taken based on the error code.

Note

Memory for aOutErrorLevel should be allocated in advance.

Related Function

ALA_ClearErrorMgr

ALA_GetErrorCode

ALA_GetErrorMessage

Example

Please refer to ALA_ClearErrorMgr.

ALA_GetErrorMessage

Syntax

```
ALA_RC ALA_GetErrorMessage(
    const ALA_ErrorMgr  * aErrorMgr,
    const SChar        ** aOutErrorMessage);
```

Argument

Argument	Description
aErrorMgr	Error Manager
aOutErrorMessage	Error Message

Result

ALA_SUCCESS

ALA_FAILURE

Description

Obtains a specific error message.

Note

Since aOutErrorMessage is replaced with a pointer to the error message, memory is not allocated for aOutErrorMessage.

Related Function

ALA_ClearErrorMgr

ALA_GetErrorCode

ALA_GetErrorLevel

Example

Please refer to ALA_ClearErrorMgr.

Appendix A. Error Codes

Error Code Table

The following table describes error codes and their possible causes.

FATAL Error

Error Code	Description	Can be returned from
0x50008	Attempted to begin an active transaction.	ALA_ReceiveXLog ALA_GetXLog
0x5000A	Mutex Initialization Failure	ALA_CreateXLogCollector ALA_Handshake
0x5000B	Mutex Remove Failure	ALA_Handshake ALA_DestroyXLogCollector
0x5000C	Mutex Lock Failure	ALA_AddAuthInfo ALA_RemoveAuthInfo ALA_Handshake ALA_ReceiveXLog ALA_GetXLog ALA_SendACK ALA_FreeXLog ALA_DestroyXLogCollector ALA_GetXLogCollectorStatus
0x5000D	Mutex Unlock Failure	

ABORT Error

Error Code	Description	Can be returned from
0x51006	Memory Allocation Failure	All Log Analysis API's
0x5101E	Failed to allocate memory from pool.	ALA_ReceiveXLog
0x5101F	Failed to free memory from pool.	ALA_Handshake ALA_ReceiveXLog ALA_FreeXLog ALA_DestroyXLogCollector
0x51020	Memory Pool Initialization Failure	ALA_CreateXLogCollector

Error Code	Description	Can be returned from
0x51021	Memory Pool Remove Failure	ALA_DestroyXLogCollector
0x51013	Network Context Initialization Failure	ALA_Handshake ALA_ReceiveXLog ALA_SendACK
0x51019	Network Protocol Remove Failure	
0x5101A	Network Context End Failure	
0x51017	The network session has already ended.	ALA_ReceiveXLog ALA_SendACK
0x51018	Abnormal Network Protocol	ALA_Handshake ALA_ReceiveXLog
0x51016	Network Read Failure	
0x5101B	Network Write Failure	ALA_Handshake ALA_SendACK
0x5101C	Network Flush Failure	
0x51015	Network Timeout (Network Error)	ALA_Handshake
0x5102C	Network Session Add Failure	ALA_Handshake
0x51024	The protocol version is not the same.	ALA_Handshake
0x51027	Link Allocation Failure	ALA_Handshake
0x51028	Link Listen Failure	ALA_Handshake
0x51029	Link Wait Failure	ALA_Handshake
0x5102A	Link Accept Failure	ALA_Handshake
0x5102B	Link Set Failure	ALA_Handshake
0x51022	Link Shutdown Failure	ALA_Handshake ALA_DestroyXLogCollector
0x51023	Link Free Failure	
0x51012	The meta information does not exist.	ALA_Handshake ALA_GetXLog ALA_GetReplicationInfo ALA_GetTableInfo ALA_GetTableInfoByName
0x5103F	The table information does not exist.	ALA_GetXLog
0x51040	The column information does not exist.	ALA_GetXLog

INFO Error

Error Code	Description	Can be returned from
0x52034	Log Analysis API Environment Create Failed	ALA_InitializeAPI
0x52035	Log Analysis API Environment Remove Failed	ALA_DestroyAPI
0x52000	Log Manager Initialization Failure	ALA_EnableLogging
0x52001	Log File Open Failure	ALA_EnableLogging
0x52004	Log Manager Lock Failure	All Log Analysis API's
0x52005	Log Manager Unlock Failure	All Log Analysis API's
0x52003	Log Manager Remove Failure	ALA_DisableLogging
0x52002	Log File Close Failure	ALA_DisableLogging
0x52009	Not an active transaction	ALA_GetXLog
0x5200E	The linked list is not empty.	ALA_Handshake ALA_DestroyXLogCollector
0x52033	XLog Pool is empty.	ALA_ReceiveXLog
0x5200F	NULL Parameter	All Log Analysis API's
0x5201D	Invalid Parameter	All Log Analysis API's
0x52014	Network Timeout (can be retried)	ALA_ReceiveXLog
0x52026	A socket type that is not supported.	ALA_Handshake
0x52025	A socket type is not selected.	ALA_Handshake
0x5202F	The socket type does not support the corresponding Log Analysis API.	ALA_AddAuthInfo ALA_RemoveAuthInfo
0x5202D	The XLog Sender name is different.	ALA_Handshake
0x52030	There is only one piece of authentication information available.	ALA_RemoveAuthInfo
0x52031	No more authentication information can be added.	ALA_AddAuthInfo
0x52032	There is no authentication information available for a peer.	ALA_Handshake
0x52010	Invalid Role	ALA_Handshake
0x52011	Invalid Replication Flags	ALA_Handshake
0x52007	Geometry Endian Conversion Failure	ALA_GetXLog

Error Code	Description	Can be returned from
0x52036	Unable to obtain the MTD module.	ALA_GetXLog ALA_GetAltibaseText ALA_GetAltibaseSQL
0x52037	Failed to create text with the MTD module.	ALA_GetAltibaseText
0x52038	CMT Initialization Failure	ALA_GetODBCCValue
0x52039	CMT End Failure	ALA_GetODBCCValue
0x5203A	Analysis Header Create Failed (ODBC Conversion)	ALA_GetODBCCValue
0x5203B	Analysis Header Remove Failure (ODBC Conversion)	ALA_GetODBCCValue
0x5203C	Failed to convert from MT to CMT.	ALA_GetODBCCValue
0x5203D	Failed to convert from CMT to ulnColumn.	ALA_GetODBCCValue
0x5203E	Failed to convert from ulnColumn to ODBC C.	ALA_GetODBCCValue

Appendix B. Sample Codes

Sample Code : Replication to Altibase DBMS

Sample code is provided as sample/ALA/Altibase/ReplToAltiSample.c.

Following is a brief description of how to use Log Analyzer. Refer to Chapter 2. XLog Sender for details.

XLog Sender Creation

```
CREATE REPLICATION ALA1 FOR ANALYSIS
    WITH '127.0.0.1', 47146
    FROM ala.ala_t1 TO ala.ala_t1;
```

XLog Collector Execution

```
./ReplToAltiSample
```

XLog Sender Startup

```
ALTER REPLICATION ALA1 START;
```

Sample Code

This sample code is available in the 'sample/ALA/Altibase/ReplToAltiSample.c' file.

```
/
*************************************************************************
**
 * Replication to Altibase DBMS Sample *
 * based on Committed Transaction Only *

*************************************************************************
**/
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/* Include Altibase ODBC header */
#include <sqlcli.h>

/* Include Altibase Log Analysis API header */
#include <alaAPI.h>

/* User-specific Definitions */
#define QUERY_SIZE (4196)            /* SQL Query Buffer Size */
```

```
#define ALA_LOG_FILE "ALA1.log"      /* Log File Name */
#define ALA_NAME "ALA1"              /* XLog Sender Name */
#define SOCKET_TYPE "TCP"            /* TCP or UNIX */
#define PEER_IP "127.0.0.1"          /* TCP : XLog Sender IP */
#define MY_PORT (47146)              /* TCP : XLog Collector Listen Port */
#define SLAVE_IP "127.0.0.1"         /* ODBC : Target Altibase DBMS IP */
#define SLAVE_PORT (43146)           /* ODBC : Target Altibase DBMS Port */

/* Get XLog from XLog Sender, after handshake with XLog Sender */
ALA_RC runXLogCollector(ALA_Handle, ALA_ErrorMgr *);

/* And, apply XLog to Altibase DBMS */
ALA_RC applyXLogToAltibase(ALA_Handle, ALA_XLog *, ALA_ErrorMgr *);

/* Print error to console */
void printSqlErr(SQLHDBC, SQLHSTMT);
void printAlaErr(ALA_ErrorMgr * aErrorMgr);

/* ODBC variables */
SQLHENV gEnv;
SQLHDBC gDbc;
SQLHSTMT gStmt;

/* Start function */
int main(void)
{
 ALA_Handle sHandle;                 /* XLog Collector Handle */
 ALA_ErrorMgr sErrorMgr;             /* Error Manager */
 char sSocketInfo[128];       /* XLog Sender/Collector Socket Information */
 SQLCHAR sConInfo[128];              /* ODBC Connection Information */
 unsigned int sStep = 0;

 /*****************************************************************************
 * Altibase ODBC Initialization *
 *****************************************************************************/

 /* If you call SQLAllocEnv() that is included in Altibase ODBC,
  * you have to set ALA_TRUE to the first parameter
  * when you call ALA_IntializeAPI()
 */
 if(SQLAllocEnv(&gEnv) != SQL_SUCCESS)
 {
 goto FINALYZE;
 }
 sStep = 1;

 if(SQLAllocConnect(gEnv, &gDbc) != SQL_SUCCESS)
 {
 goto FINALYZE;
 }
 sStep = 2;

 memset(sConInfo, 0x00, 128);
 sprintf((char *)sConInfo, "DSN=%s;UID=SYS;PWD=MANAGER;PORT_NO=%d",
 SLAVE_IP,
                              SLAVE_PORT);

 if(SQLDriverConnect(gDbc, NULL, sConInfo, SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_NOPROMPT)
 != SQL_SUCCESS)
 {
 printSqlErr(gDbc, gStmt);
 goto FINALYZE;
 }
 sStep = 3;
```

```
 /* Autocommit OFF */
 if(SQLSetConnectAttr(gDbc, SQL_ATTR_AUTOCOMMIT, (SQL-
POINTER)SQL_AUTOCOMMIT_OFF, 0)
 != SQL_SUCCESS)
 {
 printSqlErr(gDbc, gStmt);
 goto FINALYZE;
 }

 /*************************************************************************
 * ALA Initialization *
 *************************************************************************/

 /* Initialize Error Manager */
 (void)ALA_ClearErrorMgr(&sErrorMgr);

 /* Initialize ALA API environment */
 if(ALA_InitializeAPI(ALA_TRUE, &sErrorMgr) != ALA_SUCCESS)
 {
 printAlaErr(&sErrorMgr);
 goto FINALYZE;
 }
 sStep = 4;

 /* Initialize ALA Logging */
 if(ALA_EnableLogging((const signed char *)".", /* Current Directory */
 (const signed char *)ALA_LOG_FILE,             /* Log File Name */
 10 * 1024 * 1024,                              /* Log File Size */
 20,                              /* Maximum Previous Log File Count */
              &sErrorMgr)
           != ALA_SUCCESS)
 {
       printAlaErr(&sErrorMgr);
       goto FINALYZE;
 }
 sStep = 5;

 /* Create XLogCollector */
 memset(sSocketInfo, 0x00, 128);
 sprintf(sSocketInfo, "SOCKET=%s;PEER_IP=%s;MY_PORT=%d",
                       SOCKET_TYPE,
                       PEER_IP,
                       MY_PORT);
 if(ALA_CreateXLogCollector((const signed char *)ALA_NAME,
                           (const signed char *)sSocketInfo,
                            10000,               /* XLog Pool Size */
                            ALA_TRUE, /* Use Committed Transaction Buffer */
                            100,                 /* ACK Per XLog Count */
                            &sHandle,
                            &sErrorMgr)
             != ALA_SUCCESS)
 {
     printAlaErr(&sErrorMgr);
     goto FINALYZE;
 }
 sStep = 6;

 /* Set Timeouts */
 if(ALA_SetHandshakeTimeout(sHandle, 600, &sErrorMgr) != ALA_SUCCESS)
 {
     printAlaErr(&sErrorMgr);
     goto FINALYZE;
 }
 if(ALA_SetReceiveXLogTimeout(sHandle, 10, &sErrorMgr) != ALA_SUCCESS)
```

```
    {
        printAlaErr(&sErrorMgr);
        goto FINALYZE;
    }

    /*************************************************************************
    * Using XLog Collector *
    *************************************************************************/

    (void)runXLogCollector(sHandle, &sErrorMgr);

    FINALYZE:
    /*************************************************************************
    * Finalization *
    *************************************************************************/

    switch(sStep)
    {
         case 6:
              /* Destroy XLog Collector */
              (void)ALA_DestroyXLogCollector(sHandle, &sErrorMgr);

         case 5:
              /* Finalize Logging */
              (void)ALA_DisableLogging(&sErrorMgr);

         case 4:
              /* Destroy ALA API environment */
              (void)ALA_DestroyAPI(ALA_TRUE, &sErrorMgr);

         case 3:
              (void)SQLDisconnect(gDbc);

         case 2:
              (void)SQLFreeConnect(gDbc);

         case 1:
              (void)SQLFreeEnv(gEnv);

         default:
                   break;
    }
    return 0;
}
ALA_RC runXLog Collector(ALA_Handle aHandle, ALA_ErrorMgr * aErrorMgr)
{
     ALA_XLog * sXLog = NULL;
     ALA_XLogHeader * sXLogHeader = NULL;
     UInt sErrorCode;
     ALA_ErrorLevel sErrorLevel;
     ALA_BOOL sReplStopFlag = ALA_FALSE;
     ALA_BOOL sDummyFlag = ALA_FALSE;
     ALA_BOOL sAckFlag;

     /* Run until ALA_ERROR_FATAL Error occurs or REPL_STOP XLog arrives */
      while(sReplStopFlag != ALA_TRUE)
       {
         /* Wait and Handshake with XLog Sender */
            if(ALA_Handshake(aHandle, aErrorMgr) != ALA_SUCCESS)
            {
              printAlaErr(aErrorMgr);
              (void)ALA_GetErrorLevel(aErrorMgr, &sErrorLevel);
               if(sErrorLevel == ALA_ERROR_FATAL)
               {
                  return ALA_FAILURE;
```

```
        }
        /* Wait and Handshake with XLog Sender */
        continue;
      }

       while(sReplStopFlag != ALA_TRUE)
      {
        /* Get XLog from XLog Queue */
         if(ALA_GetXLog(aHandle, (const ALA_XLog **)&sXLog, aErrorMgr)
          != ALA_SUCCESS)
                  {
                     printAlaErr(aErrorMgr);
                     (void)ALA_GetErrorLevel(aErrorMgr, &sErrorLevel);
                     if(sErrorLevel == ALA_ERROR_FATAL)
                     {
                        return ALA_FAILURE;
                     }
                     /* Wait and Handshake with XLog Sender */
                      break;
        }
        else
        {
             /* If XLog is NULL, then Receive XLog */
             if(sXLog == NULL)
             {
               /* Receive XLog and Insert into Queue */
                if(ALA_ReceiveXLog(aHandle, &sDummyFlag, aErrorMgr)
!= ALA_SUCCESS)
             {
                printAlaErr(aErrorMgr);
                (void)ALA_GetErrorLevel(aErrorMgr, &sErrorLevel);
                if(sErrorLevel == ALA_ERROR_FATAL)
                {
                    return ALA_FAILURE;
                 }
                else
                {
                    (void)ALA_GetErrorCode(aErrorMgr, &sErrorCode);
                    if(sErrorCode == 0x52014) /* Timeout */
                 {
                    /* Receive XLog and Insert into Queue */
                    continue;
                 }
             }
              /* Wait and Handshake with XLog Sender */
              break;
      }

      /* Get XLog from XLog Queue */
       continue;
      }

      /* Get XLog Header */
      (void)ALA_GetXLogHeader(sXLog,
                              (const ALA_XLogHeader **)&sXLogHeader,
                               aErrorMgr);

       /* Check REPL_STOP XLog */
       if(sXLogHeader->mType == XLOG_TYPE_REPL_STOP)
       {
            sReplStopFlag = ALA_TRUE;
       }

        /* Apply XLog to Altibase DBMS */
        sAckFlag = ALA_FALSE;
```

```
                switch(sXLogHeader->mType)
                {
                   se XLOG_TYPE_COMMIT :
                   case XLOG_TYPE_ABORT : /* Unused in Committed Transaction
                                             Only */
                   case XLOG_TYPE_REPL_STOP :
                        (void)applyXLogToAltibase(aHandle, sXLog, aErrorMgr);
                        sAckFlag = ALA_TRUE;
                        break;

                   case XLOG_TYPE_INSERT :
                   case XLOG_TYPE_UPDATE :
                   case XLOG_TYPE_DELETE :
                   case XLOG_TYPE_SP_SET : /* Unused in Committed Transaction
                                              Only */
                  case XLOG_TYPE_SP_ABORT : /* Unused in Committed Transaction
                                               Only */
                   (void)applyXLogToAltibase(aHandle, sXLog, aErrorMgr);
                    break;

                   case XLOG_TYPE_KEEP_ALIVE :
                        sAckFlag = ALA_TRUE;
                        break;

                   case XLOG_TYPE_BEGIN :
                   case XLOG_TYPE_LOB_CURSOR_OPEN :
                   case XLOG_TYPE_LOB_CURSOR_CLOSE :
                   case XLOG_TYPE_LOB_PREPARE4WRITE :
                   case XLOG_TYPE_LOB_PARTIAL_WRITE :
                   case XLOG_TYPE_LOB_FINISH2WRITE :
                   default :
                       break;
                }

                /* Free XLog */
                if(ALA_FreeXLog(aHandle, sXLog, aErrorMgr) != ALA_SUCCESS)
                {
                    printAlaErr(aErrorMgr);
                    (void)ALA_GetErrorLevel(aErrorMgr, &sErrorLevel);
                    if(sErrorLevel == ALA_ERROR_FATAL)
                    {
                        return ALA_FAILURE;
                    }
                    /* Wait and Handshake with XLog Sender */
                    break;
                }

                /* Send ACK to XLog Sender */
                if(sAckFlag != ALA_FALSE)
                {
                    if(ALA_SendACK(aHandle, aErrorMgr) != ALA_SUCCESS)
                    {
                         printAlaErr(aErrorMgr);
                         (void)ALA_GetErrorLevel(aErrorMgr, &sErrorLevel);
                         if(sErrorLevel == ALA_ERROR_FATAL)
                         {
                             return ALA_FAILURE;
                         }
                         /* Wait and Handshake with XLog Sender */
                         break;
                     }
                }
              } /* else */
            } /* while */
```

```
     /* Rollback Current Transaction */
      (void)SQLEndTran(SQL_HANDLE_DBC, gDbc, SQL_ROLLBACK);
   } /* while */

   return ALA_SUCCESS;
}

ALA_RC applyXLogToAltibase(ALA_Handle aHandle, ALA_XLog * aXLog,
ALA_ErrorMgr * aErrorMgr)
{
    ALA_Table         * sTable = NULL;
    ALA_XLogHeader    * sXLogHeader = NULL;
    char                sQuery[QUERY_SIZE];
    char              * sImplictSPPos;

 /* Get XLog Header */
(void)ALA_GetXLogHeader(aXLog,
                        (const ALA_XLogHeader **)&sXLogHeader,
                        aErrorMgr);

  /* if COMMIT XLog, then Commit Current Transaction */
  if(sXLogHeader->mType == XLOG_TYPE_COMMIT)
  {
       (void)SQLEndTran(SQL_HANDLE_DBC, gDbc, SQL_COMMIT);
  }
  /* if ABORT XLog, then Rollback Current Transaction */
  else if(sXLogHeader->mType == XLOG_TYPE_ABORT)
  {
       (void)SQLEndTran(SQL_HANDLE_DBC, gDbc, SQL_ROLLBACK);
  }
  /* if REPL_STOP XLog, then Rollback Current Transaction */
  else if(sXLogHeader->mType == XLOG_TYPE_REPL_STOP)
  {
       (void)SQLEndTran(SQL_HANDLE_DBC, gDbc, SQL_ROLLBACK);
  }
  /* etc. */
  else
  {
       /* Get Table Information */
       if(ALA_GetTableInfo(aHandle,
                           sXLogHeader->mTableOID,
                          (const ALA_Table **)&sTable,
                           aErrorMgr) != ALA_SUCCESS)
       {
            printAlaErr(aErrorMgr);
            return ALA_FAILURE;
       }

       /* Get Altibase SQL from XLog */
       memset(sQuery, 0x00, QUERY_SIZE);
       if(ALA_GetAltibaseSQL(sTable,
                             aXLog,
                             QUERY_SIZE,
                             (signed char *)sQuery, aErrorMgr)
            != ALA_SUCCESS)
      {
            printAlaErr(aErrorMgr);
            return ALA_FAILURE;
      }

      /* In order to Apply Implicit Savepoint to Altibase DBMS,
       * '$' characters of Savepoint's Name has to be changed.
       * Unused in Committed Transaction Only */
      if((sXLogHeader->mType == XLOG_TYPE_SP_SET) ||
         (sXLogHeader->mType == XLOG_TYPE_SP_ABORT))
```

```
                {
                    while((sImplictSPPos = strchr(sQuery, '$')) != NULL)
                    {
                        *sImplictSPPos = '_';
                    }
                }

                /* Apply SQL to DBMS with ODBC */
                if(SQLAllocStmt(gDbc, &gStmt) != SQL_SUCCESS)
                {
                     return ALA_FAILURE;
                }

                if(SQLExecDirect(gStmt, (SQLCHAR *)sQuery, SQL_NTS) !=
                SQL_SUCCESS)
                {
                     printSqlErr(gDbc, gStmt);
                     (void)SQLFreeStmt(gStmt, SQL_DROP);
                     return ALA_FAILURE;
                }

                (void)SQLFreeStmt(gStmt, SQL_DROP);
           }
      return ALA_SUCCESS;
}

void printSqlErr(SQLHDBC aDbc, SQLHSTMT aStmt)
{
      SQLINTEGER errNo;
      SQLSMALLINT msgLength;
      SQLCHAR errMsg[1024];

      if(SQLError(SQL_NULL_HENV, aDbc, aStmt,
                  NULL, &errNo,
                  errMsg, sizeof(errMsg), &msgLength)
              == SQL_SUCCESS)
      {
           printf("SQL Error : %d, %s\n", (int)errNo, (char *)errMsg);
      }
}

void printAlaErr(ALA_ErrorMgr * aErrorMgr)
{
      char * sErrorMessage = NULL;
      int sErrorCode;

      (void)ALA_GetErrorCode(aErrorMgr, (unsigned int *)&sErrorCode);
      (void)ALA_GetErrorMessage(aErrorMgr, (const signed char **)&sErrorMes
       sage);

       printf("ALA Error : %d, %s\n", sErrorCode, sErrorMessage);
}
```

Index