ALTIBASE Application Development

Application Program Interface Users' Manual

release 5.3.3

ALTIBASE Application Development Application Program Interface 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

About This Manual

This manual describes how to use Altibase API.

Target Users

This manual could be useful for the following Altibase users.

- Database administrators
- Application designers
- Programmers

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

- Basic knowledge required for computers, operating systems, and operating system command
- Experience in using the relational database or understanding of the database concepts
- Computer programming experience

Software Environment

This manual has been prepared assuming Altibase 5.3.1 will be used as the database server.

Organization

This manual has been organized as follows:

- Chapter 1.JDBC

  This chapter briefly describes JDBC installation, using JDBC to connect to Altibase, solution for Korean language set problem, connection pool configuration and JDBC API.

- Chapter 2.PHP Interface

  This chapter describes how to create a PHP module at ext in PHP, and interface with Altibase.

- Chapter 3.PERL DBD DBI

  This chapter describes PERL DBD DBI, and PERL package installation and Altibase DBD installation to use PERL DBD DBI, and how to verify Altibase DBD.

- Chapter 4..NET Data Provider

  This chapter describes how to Microsoft's ADO.NET interface with Altibase DBMS.

- Chapter 6.XA Interface

  This chapter introduces the data structure and functions that are needed to use the XA functions supported by Altibase and provides basic procedures for using ODBC, SES and JDBC in

XA environment.C/C++ Precompiler.

Documentation Rule

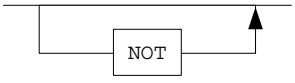
This chapter describes the rules o f this manual. With understanding of this rule, it is easy to search information in this manual and other manuals.

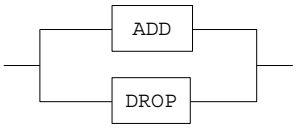
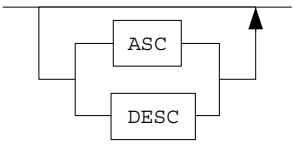
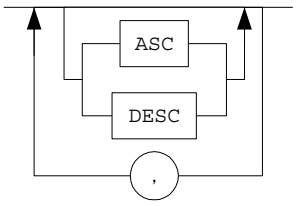
Rules are as follows:

- Syntax diagram
- Sample code rule

Syntax Diagram

This manual describes the command syntax using the diagram composed of the following elements:

Elements	Meaning
	The command starts. The syntax element which is not a complete command starts with an arrow.
	The command continues to the next line. The syntax element which is not a complete command terminates with this symbol.
	The command continues from the previous line. The syntax element which is a complete command starts with this symbol.
	End of statement.
	Mandatory
	Optional

Elements	Meaning
	Mandatory field with optional items. Only one field must be provided.
	Optional field with optional item.
	Optional multiple fields are allowed. The comma must be in front of every repetition.

Sample Code Rule

The code example explains SQL, stored procedure, iSQL, or other command line syntax.

The following table describes the printing rules used in the code example.

Rules	Meaning	Example
[ ]	Indicates optional fields.	VARCHAR [(size)] [[FIXED |] VARIABLE]
{ }	Indicates mandatory fields. Indicates to make sure to select at least one.	{ ENABLE | DISABLE | COMPILE }
|	Argument indicating optional or mandatory fields.	{ ENABLE | DISABLE | COMPILE }[ENABLE | DISABLE | COMPILE]
...	Repetition of the previous argument.Omit the example codes.	SQL> SELECT ename FROM employee; ENAME ---------- SWNO HJNO HSCHOI ...20 rows selected.

Rules	Meaning	Example
Other symbols.	Symbols other than the above.	EXEC :p1 := 1; acc NUMBER(11,2);
Italic	Indicates variable or value that must be provided by user.	SELECT * FROM table_name; CONNECT userID/password;
Small letters	Program elements provided by the user such as table names, column names, file names, etc.	SELECT ename FROM employee;
Capital letters	Elements provided by the system or keyword appeared in the syntax.	DESC SYSTEM_.SYS_INDICES_;

Related data

For more detailed information, see the following document list.

- Altibase Administration Installation User's Manual
- Altibase Administration Administrator's Manual
- Altibase Administration Replication User's Manual
- Altibase Application Development Precompiler User's Manual
- Altibase Application Development ODBC User's Manual
- Altibase Tools iSQL User's Manual
- Altibase Tools Utilities User's Manual
- ALTIBAE 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 JDBC

This chapter briefly describes JDBC installation, using JDBC to connect to Altibase, solution for Korean language set problem, connection pool configuration and JDBC API.

Installation

Download the JDBC driver (Altibase.jar) from the Altibase website (www.altibase.com). 2. Copy to the resource directory of each WAS. If necessary, edit the configuration file for each WAS.

Checking JDBC Driver Version

Following shows how to check the driver version of the currently used JDBC and JDK.

```
shell> java -jar $ALTIBASE_HOME/lib/Altibase.jar
JDBC Driver Info : Altibase Ver = 5.3.1.0 for JavaVM v1.4, CMP:5.5.1, $Revi-
sion: 14502 $
```

JAVA Application

Append the pathe of "Altibase.jar" to the CLASSPATH environment like following example:

```
CLASSPATH=$CLASSPATH:/usr/local/lib/Altibase.jar
export CLASSPATH
```

Tomcat

If you copy the Altibase.jar file in the $TOMCAT_HOME/common/lib directory, you will be able to use the Altibase JDBC driver in all Web Applications. $TOMCAT_HOME is the home directory of Tomcat installation.

WebLogic

Copy the "Altibase.jar" file to the $WL_HOME/lib/ folder. Add the path of Altibase.jar to CLASSPATH of startWebLogic.sh. $WL_HOME is the home directory of Weblogic installation.

Jeus

Copy the "Altibase.jar" file to the /$JEUS_HOME/lib/datasource/ foler.

$JEUS_HOME is the home directory of Jeus installation.

Using JDBC to Connect to Altibase

Connection Sequence

Loading Driver

Load the Altibase JDBC driver.

Class.forName("Altibase.jdbc.driver.AltibaseDriver”);

Here, there is no need to use the drivermanager to create and register an instance for the driver. If the "Class.forName” shown above is called, it will create instatace and register automatically.

This process makes it possible to load the driver, and connect to Altibase.

Connecting to a database

For connection, use the type of connection character string supported by the Altibase JDBC driver.

Connection character string format:

Connecting TCP/IP :

jdbc:Altibase://server_ip:server_port/dbname

Connecting IPC using JNI

jdbc:Altibase://localhost:port_no/dbname?user=SYS&password=MANAGER&encoding=KSC5601&CONNTYPE=3

jdbc:Altibase://IPC:port_no/dbname

Example

The following is JDBC example program to connect to a database existing as Altibase installation direcotry, $ALTIBASE_HOME/sample/JDBC/JdbcTest.java.

```
import java.util.Properties;
import java.sql.*;
class JdbcTest
{
 public static void main(String args[]) {

 Properties props = new Properties();
 Connection con = null;
 Statement stmt = null;
 PreparedStatement pstmt = null;
 ResultSet res;

 if ( args.length == 0 )
 {
 System.err.println("Usage : java JdbcTest port_no \n");
 System.exit(1);
 }

 String port = args[0];
 String url = "jdbc:Altibase://127.0.0.1:" + port + "/mydb";
 String user = "SYS";
```

```
String passwd = "MANAGER";

props.put("user", user);
props.put("password", passwd);
props.put("privilege", "sysdba"); /* This denotes access to database
remotely with sysdba privilege. */

/* Deploy Altibase's JDBC Driver */
try
{
Class.forName("Altibase.jdbc.driver.AltibaseDriver");
}
catch ( Exception e )
{
System.err.println("Can't register Altibase Driver");
return;
}

/* Initialize environment */
try
{
con = DriverManager.getConnection(url,props);
stmt = con.createStatement();
}
catch ( Exception e )
{
e.printStackTrace();
}

try
{
stmt.execute("DROP TABLE TEST001");
}
catch ( SQLException se )
{
}

try
{
stmt.execute("CREATE TABLE TEST001 ( name varchar(20), age number(3) )");

pstmt = con.prepareStatement("INSERT INTO TEST001 VALUES(?,?)");

pstmt.setString(1,"Alexandr Macedon");
pstmt.setInt(2,28);
pstmt.execute();


pstmt.setString(1,"Nikola Tesla");
pstmt.setInt(2,25);
pstmt.execute();


pstmt.setString(1,"Frankenstein");
pstmt.setInt(2,34);
pstmt.execute();

res = stmt.executeQuery("SELECT * FROM TEST001");

/* Fetch all data */
while(res.next())
{
System.out.println(" Name : "+res.getString(1)+", Age : "+res.getInt(2));
}
```

```
 /* Finalize process */
 stmt.close();
 pstmt.close();
 con.close();
 } catch ( Exception e ) {
 e.printStackTrace();
 }
 }
}
```

How to execute the example program is as follows.

```
$ javac JdbcTest.java
$ java JdbcTest 20300
Name : Alexandr Macedon, Age : 28
Name : Nikola Tesla, Age : 25
Name : Frankenstein, Age : 34
```

Solution for Korean Language Set Problem

Character Set Conversion

ALTIBASE JBDC obtains information of database character set in ALTIBASE server when connecting to application, and then converts Java string of UTF16 to database character set.

```
UTF16 Java String <-> Altibase Server DB Character Set
```

Settings in JSP

On the top of JSP page,

```
add <%@ page contentType="text/html;charset=euc-kr" %>.
```

To be normally performed in the JSP and Servlet progam,

Korean character executes

```
request.setCharacterEncoding("KSC5601");.
```

National Character Process

This section describes how to use national characters such as NCHAR and NVARCHAR in JDBC,

Data Lookup and Change

You can look up and change NCAHR and NVARCHAR typed data by using JDBC with getString and setString like CHAR and VARCHAR typed data.

Using Constant Character String

How to use constant character string with national character type in SQL statements is as follows.

- You should specify NcharLiteralReplace as true.
- You should add 'N' in front of character string to use constant character string with national character type in SQL statements.

Example

```
// create table t1 (c1 nvarchar(1000));
Properties sProps;
sProps.put( "user", "SYS");
sProps.put( "password", "MANAGER");
sProps.put( "NcharLiteralReplace", "true");
Connection sCon = DriverManager.getConnection( sURL, sProps );
Statement sStmt = sCon.createStatement();
sStmt.execute("insert into t1 values (N'AB 가나 ')");
ResultSet sRS = sStmt.executeQuery( "select * from t1 where c1 like N'% 가나
%'");
```

Connection Pool Configuration

Depending on the WAS type, a connection pool can be created using a GUI interface. However, this section describes the configuration method that edits the configuration file only. Since different WAS versions provide different configuration methods, check to see if there are any differences between the versions.

Tomcat 4.x

Assume that the jakarta commons-dbcp package is used.

Procedure

Install the jakarta commons-dbcp package.

For more information on how to install the jakarta commons-dbcp package, please see http://jakarta.apache.org.

When the 'jakarta commons-dbcp' package has been successfully installed, the following resource should be registered in the '$TOMCAT_HOME/conf/server.xml' file.

```
<!------- server.xml ------->
<ResourceParams name="jdbc/altidb">
 <parameter>
 <name>factory</name>
 <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
 </parameter>
 <!?Maximum connection count -->
 <parameter>
 <name>maxActive</name>
 <value> 5</value>
 </parameter>
 <!?Maximum connection count, which is available to occur without executing
it in pool -->
 <parameter>
 <name>maxIdle</name>
 <value>3</value>
 </parameter>
 <!?Maximum waiting time until connection is completed. (unit : millisecond--
>
 <parameter>
 <name>maxWait</name>
 <value>10000</value>
 </parameter>
 <parameter>
 <name>removeAbandoned</name>
 <value>true</value>
 </parameter>
 <parameter>
 <name>removeAbandonedTimeout</name>
 <value>60</value>
 </parameter>
 <parameter>
 <name>username</name>
 <value>SYS</value>
 </parameter>
 <parameter>
 <name>password</name>
```

```
 <value>MANAGER</value>
 </parameter>
<parameter>
 <name>driverClassName</name>
 <value>Altibase.jdbc.driver.AltibaseDriver</value>
 </parameter>
 <parameter>
 <name>url</name>
 <value>jdbc:Altibase://127.0.0.1:20300/mydb</value>
 </parameter>
</ResourceParams>
```

$ Append the followings in the 'TOMCAT_HOME/conf/web.xml' file

```
<!----- web.xml ----->
<resource-ref>
 <description>Altibase Datasource </description>
 <res-ref-name>jdbc/altidb</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

WebLogic 6.x

Create a JDBC connection pool by editing the <JDBCConnectionPool> element in the $WL_HOME/config/$DomainName/config.xml file.

```
<!------ config.xml -------->
<JDBCConnectionPool CapacityIncrement="5"
 DriverName="Altibase.jdbc.driver.AltibaseDriver"
 <!—Initial connection count -->
 InitialCapacity="5"
 <!—Maximum connection count -->
 MaxCapacity="50"
 Name="altiPool"
 RefreshMinutes="10"
 ShrinkPeriodMinutes="15"
 ShrinkingEnabled="true"
<!—Set Custom Property --> Properties="user=SYS;password=MANAGER;encod-
ing=KSC5601;portNumber=20300;databaseName=mydb;serverName=192.168.1.1"
 Targets="myserver"
 TestTableName="dual"
 URL="jdbc:Altibase://192.168.1.1:20300/mydb"
/>
```

Jeus 3.x

Set the POOL by editing the <DataSoruce> element in the '$JEUS_HOME/config/JeusMain.xml file.

```
<!------ JeusMain.xml ----->
<DataSource>
 <Database>
 <Vendor>others</Vendor>
 <ExportName>Alti_XA_DB</ExportName>
<!—Sets Data Source Class --> <DataSourceClassName>Alti-
base.jdbc.driver.ABConnectionPoolDataSource</DataSourceClassName>
 <DatabaseName>mydb</DatabaseName>
 <User>SYS</User>
```

```
 <Password>MANAGER</Password>
 <PortNumber>20300</PortNumber>
 <ServerName>192.168.1.1</ServerName>
 <!--- Sets the custom property --->
<DataSourceType>ConnectionPoolDataSource</DataSourceType>
 <!— Sets the connection pool -->
 <ConnectionPool>
 <MinPoolSize>4</MinPoolSize>
 <InitialPoolSize>4</InitialPoolSize>
 <MaxPoolSize>20</MaxPoolSize>
 <PropertyCycle>1</PropertyCycle>
 <!?Idle connection check interval -->
 <MaxIdleTime>300</MaxIdleTime>
 <ResizingPeriod>60</ResizingPeriod>
 <!?Max. DB operation time -->
 <OperationTimeout>500</OperationTimeout>
 </ConnectionPool>
</Database>
</DataSource>
```

JDBC 2.1 Core API and JDBC 2.0 Optional Package API

Functions currently supported by Altibase JDBC, functions to be supported in the future, and unsupported functions are described here by the JDBC version and package.

JDBC 3.0 API

- java.sql package => JDBC core API
- javax.sql package => JDBC Optional Package API

JDBC 3.0 API includes all previous JDBC API versions.

- JDBC 2.1 core API
- JDBC 2.0 Optional Package API

The above two combined are called JDBC 2.0 API.

- JDBC 1.2 API
- JDBC 1.0 API

java.sql Package Support Status

Altibase JDBC Driver support status in terms of JDBC 3.0

Driver

Return	Function Name	Support
boolean	acceptsURL(String url)	O
Connection	connect(String url, Properties info)	O
Int	getMajorversion()	O
Int	getMinorVersion()	O
DriverPropertyInfo[]	getPropertyInfo(String url, Properties info)	O
boolean	jdbcCompliant()	O

Connection

Return	Function Name	Support
Void	clearWarnings()	O
Void	close()	O
Void	commit()	O
Statement	createStatement()	O
Statement	createStatement(int resultSetType, int resultSetConcurrency)	O
Statement	createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	X
Boolean	getAutoCommit()	O
String	getCatalog()	
Int	getHoldability()	X
DatabaseMetaData	getMetaData()	O
Int	getTransactionIsolation()	O
Map	getTypeMap()	O
SQLWarning	getWarnings()	O
Boolean	isClosed()	O
Boolean	isReadOnly()	O
String	nativeSQL(String sql)	O
CallableStatement	prepareCall(String sql)	O
CallableStatement	prepareCall(String sql, int resultSetType, int resultSetConcurrency)	X
CallableStatement	prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	X
PreparedStatement	prepareStatement(String sql)	X
PreparedStatement	prepareStatement(String sql, int autoGeneratedKeys)	
PreparedStatement	prepareStatement(String sql, int[] columnIndexes)	X
PreparedStatement	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	O
PreparedStatement	prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	X
PreparedStatement	prepareStatement(String sql, String[] columnNames)	X

Return	Function Name	Support
Void	releaseSavepoint(Savepoint savepoint)	O
Void	rollback()	O
Void	rollback(Savepoint savepoint)	O
Void	setAutoCommit(boolean autoCommit)	O
Void	setCatalog(String catalog)	
Void	setHoldability(int holdability)	X
Void	setReadOnly(boolean readOnly	X
Savepoint	setSavepoint()	O
Savepoint	setSavepoint(String name)	O
Void	setTransactionIsolation(int level)	O
Void	setTypeMap(Map map)	X

DatabaseMetaData

Return Type	Function Name	Support
Boolean	allProceduresAreCallable()	O
Boolean	‘	O
Boolean	dataDefinitionCausesTransactionCommit()	O
Boolean	dataDefinitionIgnoredInTransactions()	O
Boolean	deletesAreDetected(int type)	O
Boolean	doesMaxRowSizeIncludeBlobs()	O
ResultSet	getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	Δ
ResultSet	getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	O
ResultSet	getCatalogs()	Δ
String	getCatalogSeparator()	Δ
String	getCatalogTerm()	Δ
ResultSet	getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	O
ResultSet	getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	O

Return Type	Function Name	Support
Connection	getConnection()	O
ResultSet	getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)	O
Int	getDatabaseMajorVersion()	O
Int	getDatabaseMinorVersion()	O
String	getDatabaseProductName()	O
String	getDatabaseProductVersion()	O
int	getDefaultTransactionIsolation()	O
int	getDriverMajorVersion()	O
int	getDriverMinorVersion()	O
String	getDriverName()	O
String	getDriverVersion()	O
ResultSet	getExportedKeys(String catalog, String schema, String table)	O
String	getExtraNameCharacters()	Δ
String	getIdentifierQuoteString()	Δ
ResultSet	getImportedKeys(String catalog, String schema, String table)	O
ResultSet	getIndexInfo(String catalog, String schema, String table, Boolean unique, Boolean approximate)	O
int	getJDBCMajorVersion()	O
int	getJDBCMinorVersion()	O
int	getMaxBinaryLiteralLength()	O
int	getMaxCatalogNameLength()	O
int	getMaxCharLiteralLength()	O
int	getMaxColumnNameLength()	O
Int	getMaxColumnsInGroupBy()	O
Int	getMaxColumnsInIndex()	O
int	getMaxColumnsInOrderBy()	O
int	getMaxColumnsInSelect()	O
int	getMaxColumnsInTable()	O
int	getMaxConnections()	O

Return Type	Function Name	Support
int	getMaxCursorNameLength()	O
int	getMaxIndexLength()	O
int	getMaxProcedureNameLength()	O
int	getMaxRowSize()	O
int	getMaxSchemaNameLength()	O
int	getMaxStatementLength()	O
int	getMaxStatements()	O
int	getMaxTableNameLength()	O
int	getMaxTablesInSelect()	O
int	getMaxUserNameLength()	O
String	getNumericFunctions()	O
ResultSet	getPrimaryKeys(String catalog, String schema, String table)	O
ResultSet	getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	O
ResultSet	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	O
String	getProcedureTerm()	O
int	getResultSetHoldability()	O
ResultSet	getSchemas()	O
String	getSchemaTerm()	O
String	getSearchStringEscape()	O
String	getSQLKeywords()	O
int	getSQLStateType()	O
String	getStringFunctions()	O
ResultSet	getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	X
ResultSet	getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	X
String	getSystemFunctions()	O
ResultSet	getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	O

Return Type	Function Name	Support
ResultSet	getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	O
ResultSet	getTableTypes()	O
String	getTimeDateFunctions()	O
ResultSet	getTypeInfo()	O
ResultSet	getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	O
String	getURL()	O
String	getUserName()	O
ResultSet	getVersionColumns(String catalog, String schema, String table)	X
boolean	insertsAreDetected(int type)	O
boolean	isCatalogAtStart()	O
boolean	isReadOnly()	O
boolean	locatorsUpdateCopy()	O
boolean	nullPlusNonNullIsNull()	O
boolean	nullsAreSortedAtEnd()	O
boolean	nullsAreSortedAtStart()	O
boolean	nullsAreSortedHigh()	O
boolean	nullsAreSortedLow()	O
boolean	othersDeletesAreVisible(int type)	O
boolean	othersInsertsAreVisible(int type)	O
boolean	othersUpdatesAreVisible(int type)	O
boolean	ownDeletesAreVisible(int type)	O
boolean	ownInsertsAreVisible(int type)	O
boolean	ownUpdatesAreVisible(int type)	O
boolean	storesLowerCaseIdentifiers()	O
boolean	storesLowerCaseQuotedIdentifiers()	O
boolean	storesMixedCaseIdentifiers()	O
boolean	storesMixedCaseQuotedIdentifiers()	O
boolean	storesUpperCaseIdentifiers()	O
boolean	storesUpperCaseQuotedIdentifiers()	O

Return Type	Function Name	Support
boolean	supportsAlterTableWithAddColumn()	O
boolean	supportsAlterTableWithDropColumn()	O
boolean	supportsANSI92EntryLevelSQL()	O
boolean	supportsANSI92FullSQL()	O
boolean	supportsANSI92IntermediateSQL()	O
boolean	supportsBatchUpdates()	O
boolean	supportsCatalogsInDataManipulation()	O
boolean	supportsCatalogsInIndexDefinitions()	O
boolean	supportsCatalogsInPrivilegeDefinitions()	O
boolean	supportsCatalogsInProcedureCalls()	O
boolean	supportsCatalogsInTableDefinitions()	O
boolean	supportsColumnAliasing()	O
boolean	supportsConvert()	O
boolean	supportsConvert(int fromType, int toType)	O
boolean	supportsCoreSQLGrammar()	O
boolean	supportsCorrelatedSubqueries()	O
boolean	supportsDataDefinitionAndDataManipulationTransactions()	O
boolean	supportsDataManipulationTransactionsOnly()	O
boolean	supportsDifferentTableCorrelationNames()	O
boolean	supportsExpressionsInOrderBy()	O
boolean	supportsExtendedSQLGrammar()	O
boolean	supportsFullOuterJoins()	O
boolean	supportsGetGeneratedKeys()	O
boolean	supportsGroupBy()	O
boolean	supportsGroupByBeyondSelect()	O
boolean	supportsGroupByUnrelated()	O
boolean	supportsIntegrityEnhancementFacility()	O
boolean	supportsLikeEscapeClause()	O
boolean	supportsLimitedOuterJoins()	O
boolean	supportsMinimumSQLGrammar()	O

Return Type	Function Name	Support
boolean	supportsMixedCaseIdentifiers()	O
boolean	supportsMixedCaseQuotedIdentifiers()	O
boolean	supportsMultipleOpenResults()	O
boolean	supportsMultipleResultSets()	O
boolean	supportsMultipleTransactions()	O
boolean	supportsNamedParameters()	O
boolean	supportsNonNullableColumns()	O
boolean	supportsOpenCursorsAcrossCommit()	O
boolean	supportsOpenCursorsAcrossRollback()	O
boolean	supportsOpenStatementsAcrossCommit()	O
boolean	supportsOpenStatementsAcrossRollback()	O
boolean	supportsOrderByUnrelated()	O
boolean	supportsOuterJoins()	O
boolean	supportsPositionedDelete()	O
boolean	supportsPositionedUpdate()	O
boolean	supportsResultSetConcurrency(int type, int concurrency)	O
boolean	supportsResultSetHoldability(int holdability)	O
boolean	supportsResultSetType(int type)	O
boolean	supportsSavepoints()	O
boolean	supportsSchemasInDataManipulation()	O
boolean	supportsSchemasInIndexDefinitions()	O
boolean	supportsSchemasInPrivilegeDefinitions()	O
boolean	supportsSchemasInProcedureCalls()	O
boolean	supportsSchemasInTableDefinitions()	O
boolean	supportsSelectForUpdate()	O
boolean	supportsStatementPooling()	O
boolean	supportsStoredProcedures()	O
boolean	supportsSubqueriesInComparisons()	O
boolean	supportsSubqueriesInExists()	O
boolean	supportsSubqueriesInIns()	O

Return Type	Function Name	Support
boolean	supportsSubqueriesInQuantifieds()	O
boolean	supportsTableCorrelationNames()	O
boolean	supportsTransactionIsolationLevel(int level)	O
boolean	supportsTransactions()	O
boolean	supportsUnion()	O
boolean	supportsUnionAll()	O
boolean	updatesAreDetected(int type)	O
boolean	usesLocalFilePerTable()	O
boolean	usesLocalFiles()	O

ResultSet

Return	Function Name	Support
boolean	absolute(int row)	O
void	afterLast()	O
void	beforeFirst()	O
void	cancelRowUpdates()	X
void	clearWarnings()	O
void	close()	O
void	deleteRow()	X
int	findColumn(String columnName)	O
boolean	first()	O
Array	getArray(int i)	X
Array	getArray(String colName)	X
InputStream	getAsciiStream(int columnIndex)	O
InputStream	getAsciiStream(String columnName)	O
BigDecimal	getBigDecimal(int columnIndex)	O
BigDecimal	getBigDecimal(int columnIndex, int scale)	O
BigDecimal	getBigDecimal(String columnName)	O
BigDecimal	getBigDecimal(String columnName, int scale)	O

Return	Function Name	Support
InputStream	getBinaryStream(int columnIndex).	O
InputStream	getBinaryStream(String columnName)	O
Blob	getBlob(int i)	O
Blob	getBlob(String colName)	O
boolean	getBoolean(int columnIndex)	O
boolean	getBoolean(String columnName)	O
byte	getByte(int columnIndex)	O
byte	getByte(String columnName)	O
byte[]	getBytes(int columnIndex)	O
byte[]	getBytes(String columnName)	O
Reader	getCharacterStream(int columnIndex)	O
Reader	getCharacterStream(String columnName)	O
Clob	getClob(int i)	X
Clob	getClob(String colName)	X
int	getConcurrency()	O
String	getCursorName()	X
Date	getDate(int columnIndex)	O
Date	getDate(int columnIndex, Calendar cal)	Δ
Date	getDate(String columnName)	O
Date	getDate(String columnName, Calendar cal)	Δ
double	getDouble(int columnIndex)	O
double	getDouble(String columnName)	O
int	getFetchDirection()	O
int	getFetchSize()	O
float	getFloat(int columnIndex)	O
float	getFloat(String columnName)	O
int	getInt(int columnIndex)	O
int	getInt(String columnName)	O
long	getLong(int columnIndex)	O
long	getLong(String columnName)	O
ResultSetMetaData	getMetaData()	O

Return	Function Name	Support
Object	getObject(int columnIndex)	O
Object	getObject(int i, Map map)	X
Object	getObject(String columnName)	O
Object	getObject(String colName, Map map)	X
Ref	getRef(int i)	X
Ref	getRef(String colName)	X
int	getRow()	O
short	getShort(int columnIndex)	O
short	getShort(String columnName)	O
Statement	getStatement()	O
String	getString(int columnIndex)	O
String	getString(String columnName)	O
Time	getTime(int columnIndex)	O
Time	getTime(int columnIndex, Calendar cal)	Δ
Time	getTime(String columnName)	O
Time	getTime(String columnName, Calendar cal)	Δ
Timestamp	getTimestamp(int columnIndex)	O
Timestamp	getTimestamp(int columnIndex, Calendar cal)	Δ
Timestamp	getTimestamp(String columnName)	O
Timestamp	getTimestamp(String columnName, Calendar cal)	Δ
int	getType()	O
InputStream	getUnicodeStream(int columnIndex)	X
InputStream	getUnicodeStream(String columnName)	X
URL	getURL(int columnIndex)	X
URL	getURL(String columnName)	X
SQLWarning	getWarnings()	O
void	insertRow().	X
boolean	isAfterLast()	O
boolean	isBeforeFirst()	O
boolean	isFirst()	O
boolean	isLast()	O

Return	Function Name	Support
boolean	last()	O
void	moveToCurrentRow()	O
void	moveToInsertRow()	X
boolean	next()	O
boolean	previous()	O
void	refreshRow()	X
boolean	relative(int rows)	O
boolean	rowDeleted()	X
boolean	rowInserted()	X
boolean	rowUpdated()	X
void	setFetchDirection(int direction)	Δ
void	setFetchSize(int rows)	O
void	updateArray(int columnIndex, Array x)	X
void	updateArray(String columnName, Array x)	X
void	updateAsciiStream(int columnIndex, InputStream x, int length)	X
void	updateAsciiStream(String columnName, InputStream x, int length)	X
void	updateBigDecimal(int columnIndex, BigDecimal x)	X
void	updateBigDecimal(String columnName, BigDecimal x)	X
void	updateBinaryStream(int columnIndex, InputStream x, int length)	X
void	updateBinaryStream(String columnName, Input-Stream x, int length)	X
void	updateBlob(int columnIndex, Blob x)	X
void	updateBlob(String columnName, Blob x)	X
void	updateBoolean(int columnIndex, Boolean x)	X
void	updateBoolean(String columnName, Boolean x)	X
void	updateByte(int columnIndex, byte x)	X
void	updateByte(String columnName, byte x)	X
void	updateBytes(int columnIndex, byte[] x)	X
void	updateBytes(String columnName, byte[] x)	X

Return	Function Name	Support
void	updateCharacterStream(int columnIndex, Reader x, int length)	X
void	updateCharacterStream(String columnName, Reader reader, int length)	X
void	updateClob(int columnIndex, Clob x)	X
void	updateClob(String columnName, Clob x)	X
void	updateDate(int columnIndex, Date x)	X
void	updateDate(String columnName, Date x)	X
void	updateDouble(int columnIndex, double x)	X
void	updateDouble(String columnName, double x)	X
void	updateFloat(int columnIndex, float x)	X
void	updateFloat(String columnName, float x)	X
void	updateInt(int columnIndex, int x)	X
void	updateInt(String columnName, int x)	X
void	updateLong(int columnIndex, long x)	X
void	updateLong(String columnName, long x)	X
void	updateNull(int columnIndex)	X
void	updateNull(String columnName)	X
void	updateObject(int columnIndex, Object x)	X
void	updateObject(int columnIndex, Object x, int scale)	X
void	updateObject(String columnName, Object x)	X
void	updateObject(String columnName, Object x, int scale)	X
void	updateRef(int columnIndex, Ref x)	X
void	updateRef(String columnName, Ref x)	X
void	updateRow()	X
void	updateShort(int columnIndex, short x)	X
void	updateShort(String columnName, short x)	X
void	updateString(int columnIndex, String x)	X
void	updateString(String columnName, String x)	X
void	updateTime(int columnIndex, Time x)	X
void	updateTime(String columnName, Time x)	X
void	updateTimestamp(int columnIndex, Timestamp x)	X

Return	Function Name	Support
void	updateTimestamp(String columnName, Timestamp x)	X
boolean	wasNull()	O

ResultSetMetaData

Return Type	Function Name	Support
String	getCatalogName(int column)	Δ
String	getColumnClassName(int column)	O
Int	getColumnCount()	O
int	getColumnDisplaySize(int column)	O
String	getColumnLabel(int column)	O
String	getColumnName(int column)	O
int	getColumnType(int column)	O
String	getColumnTypeName(int column)	O
int	getPrecision(int column)	O
int	getScale(int column)	O
String	getSchemaName(int column)	O
String	getTableName(int column)	O
boolean	isAutoIncrement(int column)	O
boolean	isCaseSensitive(int column)	O
boolean	isCurrency(int column)	O
boolean	isDefinitelyWritable(int column)	O
int	isNullable(int column)	O
boolean	isReadOnly(int column)	O
boolean	isSearchable(int column)	O
boolean	isSigned(int column)	O
boolean	isWritable(int column)	O

Statement

Return Type	Function Name	Support
void	addBatch(String sql)	O
void	cancel()	O
void	clearBatch()	O
void	clearWarnings()	O
void	close()	O
boolean	execute(String sql)	O
boolean	execute(String sql, int autoGeneratedKeys)	Δ
boolean	execute(String sql, int[] columnIndexes)	X
boolean	execute(String sql, String[] columnNames)	X
int[]	executeBatch()	O
ResultSet	executeQuery(String sql)	O
int	executeUpdate(String sql)	O
int	executeUpdate(String sql, int autoGeneratedKeys)	X
int	executeUpdate(String sql, int[] columnIndexes)	X
int	executeUpdate(String sql, String[] columnNames)	X
Connection	getConnection()	O
int	getFetchDirection()	O
int	getFetchSize()	O
ResultSet	getGeneratedKeys()	X
int	getMaxFieldSize()	O
int	getMaxRows()	O
boolean	getMoreResults()	O
boolean	getMoreResults(int current)	O
int	getQueryTimeout()	O
ResultSet	getResultSet()	O
int	getResultSetConcurrency()	O
int	getResultSetHoldability()	O
int	getResultSetType()	O
int	getUpdateCount()	O

Return Type	Function Name	Support
SQLWarning	getWarnings()	O
void	setCursorName(String name)	X
void	setEscapeProcessing(Boolean enable)	X
void	setFetchDirection(int direction)	Δ
void	setFetchSize(int rows)	O
void	setMaxFieldSize(int max)	O
void	setMaxRows(int max)	O
void	setQueryTimeout(int seconds)	O

PreparedStatement

Return Type	Function Name	Support
void	addBatch()	O
void	clearParameters()	O
boolean	execute()	O
ResultSet	executeQuery()	O
int	executeUpdate()	O
ResultSetMetaData	getMetaData()	O
ParameterMetaData	getParameterMetaData()	X
void	setArray(int i, Array x)	X
void	setAsciiStream(int parameterIndex, InputStream x, int length)	O
void	setBigDecimal(int parameterIndex, BigDecimal x)	O
void	setBinaryStream(int parameterIndex, InputStream x, int length)	O
void	setBlob(int i, Blob x)	O
void	setBoolean(int parameterIndex, Boolean x)	O
void	setByte(int parameterIndex, byte x)	O
void	setBytes(int parameterIndex, byte[] x)	O
void	setCharacterStream(int parameterIndex, Reader reader, int length)	O

Return Type	Function Name	Support
void	setClob(int i, Clob x)	X
void	setDate(int parameterIndex, Date x)	O
void	setDate(int parameterIndex, Date x, Calendar cal)	X
void	setDouble(int parameterIndex, double x)	O
void	setFloat(int parameterIndex, float x)	O
void	setInt(int parameterIndex, int x)	O
void	setLong(int parameterIndex, long x)	O
void	setNull(int parameterIndex, int sqlType)	O
Void	setNull(int paramIndex, int sqlType, String typeName)	O
Void	setObject(int parameterIndex, Object x)	O
Void	setObject(int parameterIndex, Object x, int targetSql-Type)	O
Void	setObject(int parameterIndex, Object x, int targetSql-Type, int scale)	O
void	setRef(int i, Ref x)	X
void	setShort(int parameterIndex, short x)	O
void	setString(int parameterIndex, String x)	O
void	setTime(int parameterIndex, Time x)	O
void	setTime(int parameterIndex, Time x, Calendar cal)	X
void	setTimestamp(int parameterIndex, Timestamp x)	O
void	setTimestamp(int parameterIndex, Timestamp x, Cal-endar cal)	X
void	setUnicodeStream(int parameterIndex, InputStream x, int length)	X
void	setURL(int parameterIndex, URL x)	X

CallableStatement

Return Type	Function Name	Support
Array	getArray(int i)	X
Array	getArray(String parameterName)	X
BigDecimal	getBigDecimal(int parameterIndex)	O

Return Type	Function Name	Support
BigDecimal	getBigDecimal(int parameterIndex, int scale)	O
BigDecimal	getBigDecimal(String parameterName)	X
Blob	getBlob(int i)	O
Blob	getBlob(String parameterName)	X
boolean	getBoolean(int parameterIndex)	O
boolean	getBoolean(String parameterName)	X
byte	getByte(int parameterIndex)	O
byte	getByte(String parameterName)	X
byte[]	getBytes(int parameterIndex)	O
byte[]	getBytes(String parameterName)	X
Clob	getClob(int i)	X
Clob	getClob(String parameterName)	X
Date	getDate(int parameterIndex)	O
Date	getDate(int parameterIndex, Calendar cal)	Δ
Date	getDate(String parameterName)	O
Date	getDate(String parameterName, Calendar cal)	Δ
double	getDouble(int parameterIndex)	O
double	getDouble(String parameterName)	X
float	getFloat(int parameterIndex)	O
float	getFloat(String parameterName)	X
int	getInt(int parameterIndex)	O
int	getInt(String parameterName)	X
long	getLong(int parameterIndex)	O
long	getLong(String parameterName)	X
Object	getObject(int parameterIndex)	O
Object	getObject(int i, Map map)	Δ
Object	getObject(String parameterName)	X
Object	getObject(String parameterName, Map map)	X
Ref	getRef(int i)	X
Ref	getRef(String parameterName)	X
short	getShort(int parameterIndex)	O

Return Type	Function Name	Support
short	getShort(String parameterName)	X
String	getString(int parameterIndex)	O
String	getString(String parameterName)	X
Time	getTime(int parameterIndex)	O
Time	getTime(int parameterIndex, Calendar cal)	Δ
Time	getTime(String parameterName)	X
Time	getTime(String parameterName, Calendar cal)	X
Timestamp	getTimestamp(int parameterIndex)	O
Timestamp	getTimestamp(int parameterIndex, Calendar cal)	Δ
Timestamp	getTimestamp(String parameterName)	X
Timestamp	getTimestamp(String parameterName, Calendar cal)	X
URL	getURL(int parameterIndex)	X
URL	getURL(String parameterName)	X
void	registerOutParameter(int parameterIndex, int sqlType)	O
void	registerOutParameter(int parameterIndex, int sqlType, int scale)	O
void	registerOutParameter(int paramIndex, int sqlType, String typeName)	O
void	registerOutParameter(String parameterName, int sql-Type)	X
void	registerOutParameter(String parameterName, int sql-Type, int scale)	X
void	registerOutParameter(String parameterName, int sql-Type, String typeName)	X
void	setAsciiStream(String parameterName, lnputStream x, int length)	X
void	setBigDecimal(String parameterName, BigDecimal x)	X
void	setBinaryStream(String parameterName, lnputStream x, int length)	X
void	setBoolean(String parameterName, Boolean x)	X
void	setByte(String parameterName, byte x)	X
void	setBytes(String parameterName, byte[] x)	X
void	setCharacterStream(String parameterName, Reader reader, int length)	X

Return Type	Function Name	Support
void	setDate(String parameterName, Date x)	X
void	setDate(String parameterName, Date x, Calendar cal)	X
void	setDouble(String parameterName, double x)	X
void	setFloat(String parameterName, float x)	X
void	setInt(String parameterName, int x)	X
void	setLong(String parameterName, long x)	X
void	setNull(String parameterName, int sqlType)	X
void	setNull(String parameterName, int sqlType, String typeName)	X
void	setObject(String parameterName, Object x)	X
void	setObject(String parameterName, Object x, int targetSqlType)	X
void	setObject(String parameterName, Object x, int targetSqlType, int scale)	X
void	setShort(String parameterName, short x)	X
void	setString(String parameterName, String x)	X
void	setTime(String parameterName, Time x)	X
void	setTime(String parameterName, Time x, Calendar cal)	X
void	setTimestamp(String parameterName, Timestamp x)	X
void	setTimestamp(String parameterName, Timestamp x, Calendar cal)	X
void	setURL(String parameterName, URL val)	X
boolean	wasNull()	O

Blob

Return Type	Function Name	Support
InputStream	getBinaryStream()	O
byte[]	getBytes(long pos, int length)	O
long	length()	O
long	position(Blob pattern, long start)	X
long	position(byte[] pattern, long start)	X

Return Type	Function Name	Support
OutputStream	setBinaryStream(long pos)	O
int	setBytes(long pos, byte[] bytes)	O
int	setBytes(long pos, byte[] bytes, int offset, int len)	O
void	truncate(long len)	O

Clob

Return Type	Function Name	Support
int	setString(long pos, String str)	O
int	setString(long pos, String str, int offset, int len)	O

SavePoint

Return Type	Function Name	Support
int	getSavepointId()	O
String	getSavepointName()	O

javax.sql package Support Status

Altibase JDBC support status in terms of JDBC 3.0

ConnectionPoolDataSource

Return Type	Function Name	Support
int	getLoginTimeout()	X
PrintWriter	getLogWriter()	O
PooledConnection	getPooledConnection()	O
PooledConnection	getPooledConnection(String user, String password)	O
void	setLoginTimeout(int seconds)	X
void	setLogWriter(PrintWriter out)	O

DataSource

Return Type	Function Name	Support
Connection	getConnection()	O
Connection	getConnection(String username, String password)	O
int	getLoginTimeout()	X
PrintWriter	getLogWriter()	O
void	setLoginTimeout(int seconds)	X
void	setLogWriter(PrintWriter out)	O

PooledConnection

Return Type	Function Name	Support
void	addConnectionEventListener(ConnectionEventListener listener)	O
void	close()	O
Connection	getConnection()	O
void	removeConnectionEventListener(ConnectionEventListener listener)	O

XAConnection

Return Type	Function Name	Support
XAResource	getXAResource()	O

XADataSource

Return Type	Function Name	Support
int	getLoginTimeout()	X
PrintWriter	getLogWriter()	O
XAConnection	getXAConnection()	O

Return Type	Function Name	Support
XAConnection	getXAConnection(String user, String password)	O
void	setLoginTimeout(int seconds)	X
void	setLogWriter(PrintWriter out)	O

javax.transaction.xa package Support Status

XAResource

Return Type	Function Name	Support
void	commit(Xid xid, Boolean onePhase)	O
void	end(Xid xid, int flags)	O
void	forget(Xid xid)	O
int	getTransactionTimeout()	O
boolean	isSameRM(XAResource xares)	O
int	prepare(Xid xid)	O
Xid[]	recover(int flag)	O
void	rollback(Xid xid)	O
boolean	setTransactionTimeout(int seconds)	X
void	start(Xid xid, int flags)	O

Xid

Return Type	Function Name	Support
byte[ ]	getBranchQualfier()	O
Int	getFormatId()	O
byte[ ]	getGlobalTransactionId()	O

Cautions

When you upload the data by using setByte, a data size is under constraint.

When you upload the data over 64KB, you should upload by using stream of BLOB.

JDBC Connection Fail-over

When your operating database system is failed, if you have multiple operating database servers, you can make Altibase move your connections to an other server.

How it Operates

The ABConnectionPoolDataSource class provides connection fail-over functionality. If you specify the URL of other servers using the class named ABConnectionPoolDataSource(String[] aURL), Altibase generates connection pool using the server URL list. A system failure occurred, Altibase detacts it and tries to connect to the next server in the URL list.

Example

```
String[] serverList = new String[2];
 serverList [0] = "jdbc:Altibase://localhost:20556/mydb?user=SYS&pass-
word=MANAGER&CONNTYPE=1";
 serverList [1] = "jdbc:Altibase://localhost:20557/mydb?user=SYS&pass-
word=MANAGER&CONNTYPE=1";
ABConnectionPoolDataSource pool = new ABConnectionPoolDataSource(serverList);
```

Cautions

When Altibase moves connections associated transactions are rollbacked automatically.

If Altibase failed to connect to the last server in the reservered URL list, it retries to conect to the first server once, if it fail again Altibase return connection error.

2 PHP Interface

This chapter describes how to create a PHP module at ext in PHP, and interface with Altibase.

Additional Information on the Altibase PHP Module

1. The data types supported are shown below:

   resource, int, bool, double, float, string, array, HashTable

2. You must match the port with that of the Altibase server in db.php of the sample program.

Installing ODBC Manager for PHP Interface

This section describes procedures for installing ODBC Manager for PHP interface in Unix or Windows environment.

Unix ODBC

In Linux or Unix environment, complete the following steps to install ODBC Manger:

Download unixODBC.

It can be downloaded from the unixODBC website (http://www.unixodbc.org).

Install unixODBC.

When installing the downloaded source file to a specific location, enter "./configure --prefix=path".

```
./configure -prefix=installation path -enable-gui=no --enable-drivers=no
make
make install
```

Configure the unixODBC environment.

1. Set $ODBCSYSINI as an environment variable. Set the value of the $ODBCSYSINI environment variable to $HOME of the user's account.

   ```
   export ODBCSYSINI=~
   ```

2. Based on the library path and bit among environment variables, add the path where unixODBC Driver Manager is installed as in the below. The library path can be LD_LIBRARY_PATH, LD_LIBRARY_PATH_64 or SHLIB_PATH based on platform.

   In the following example, unixODBC is installed at /usr/local/odbcDriverManager32 or /usr/local/odbcDriverManager (/usr/local/odbcDriverManager64).

   ```
   export LD_LIBRARY_PATH=/usr/local/odbcDriverManager32/
   lib:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH_64=/usr/local/odbcDriverManager64/
   lib:$LD_LIBRARY_PATH_6
   ```

Create two files in the $ODBCSYSINI path as in the below:

- odbc.ini
- odbcinst.ini

odbcinst.ini should be created as an empty file with the size of 0 byte.

In odbc.ini, specify the DSN name, Altibase ODBC Driver installation, server address and port number.

Windows ODBC

In Windows environment, ODBC Manger is installed by default. Therefore, Altibase drivers are automatically registered with ODBC Manger when Altibase is installed.

To register drivers manually, use the odbcsetup.exe program provided by Altibase.

- To register drivers:

  odbcsetup -i

- To remove drivers:

  odbcsetup -u

Once drivers are registered, register with the system DSN manually.

PHP Functions for ODBC Connectivity

The user can use ODBC functions, which you can use in PHP for interface, because Altibase supports standard ODBC functions.

For information on them, please refer to PHP official site as follows:

http://php.morva.net/manual/en/index.php

Sample Test

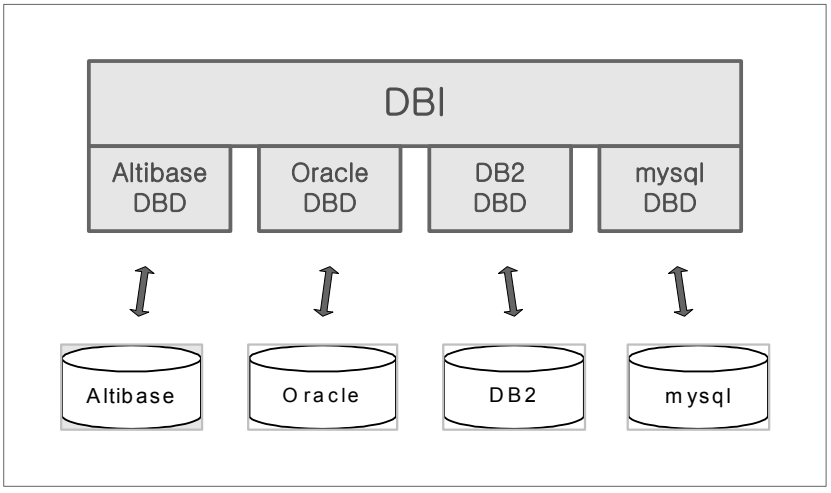
```
<?
// SYSTEM DSN, USER_ID, USER_PASSWORD
$conn = odbc_connect("Altibase", "SYS", "MANAGER");

if ($conn)
{
 // direct-execution
 echo "now, i create table t1 (id integer, name char(20)<br>";
 odbc_exec($conn, "drop table t1");
 odbc_exec($conn, "create table t1 (id integer, name char(20))");

// prepare-execution
 echo "now, i insert into t1 value (100, Lee)<br>";
 $stmt = odbc_prepare ($conn, "insert into t1 values (?, ?)");
 $Insert = array (100, "Lee");
 if (!odbc_execute($stmt, &$Insert))
 {
 echo ("error");
 }

// single-selection
 $res = odbc_do ($conn, "select id, name, sysdate from T1");
 odbc_fetch_row ($res);
 $ID = odbc_result($res, 1);
 $NAME = odbc_result($res, 2);
 $DATE = odbc_result($res, 3);
 echo ("id = $ID , name = $NAME datetime = $DATE <br>");
 odbc_close($conn);
}
?>
```

3 PERL DBD DBI

This chapter describes PERL DBD DBI, and PERL package installation and Altibase DBD installation to use PERL DBD DBI, and how to verify Altibase DBD.

Overview of PERL DBD DBI

DBI is the database API written in the form of a PERL5 module. This module defines a series of "methods" and "attributes" concerning the database in which DBD (Database Driver) is created. This driver can also be obtained through the PERL5 module. DBI serves as a switch between several DBDs in an application program. Actually it is DBD that communicates with the database. The driver of Altibase is called DBD::altibase. These methods and attributes are divided into database attributes & methods, and statement attributes & methods.



PERL Package Installation

PERL Package Installation Procedure

1. Download the PERL package according to the necessary server and version.(Example for Sun: http://www.sunfreeware.com)

    a. Decompress the package (ex. perl-5.8.5.tar.gz) in any directory.
    ```
    gzip -cd perl-5.8.5.tar.gz | tar xvf –
    ```
    b. Execute configure in the directory in which the perl package was decompressed.
    ```
    ./configure
    ```
    c. Execute make in the directory in which the perl package was decompressed.
    ```
    make
    ```
    d. Install as the root account in the directory in which the perl package was decompressed.
    ```
    make install
    ```

2. Download the event package (ex. Event-1.00 .tar.gz) to install the Event module.

    a. Decompress Event-1.00.tar.gz in any directory.
    ```
    gzip -cd Event-1.00.tar.gz | tar xvf
    ```
    b. Execute configure in the directory in which the perl package was decompressed.
    ```
    ./configure
    ```
    c. Execute configure in the directory in which the perl package was decompressed.
    ```
    make
    ```
    d. Install as the root account in the directory in which the Event package was decompressed.
    ```
    make install
    ```

Note: The event module checks the protocol type when data is received, and determines the length of the message.

It is possible to know how many bytes the buffer receives because the event sends this information after a wait.

In addition, as there are separated threads, it also enables synchronization. After all, it can detect which event is generated.

Installing Altibase DBD

Installing Altibase PERL DBD

Ensure that dlext is correct by using perl –V. It should be sl for HP and so for other platforms. If not correct, reinstall perl.

Install perl DBI.

First, install the perl DBI package that is needed for perl DBD compilation.

- Method 1) With root account:

```
# perl -MCPAN -e shell
prompt>  install DBI
```

- Method 2) If the above method does not work, download the package via ftp, compile it and then install.

  ftp://ftp.nuri.net/pub/CPAN/modules/by-module/DBI

  1. perl Makefile.PL
  2. make
  3. make install

Download and install Altibase PERL DBD. Connect to data.altibase.com and then download the file at the following location: Because perl is 32-bit in an installation system, Altibase 32-bit client package or 32-bit server package should be installed in the system. And also, the environment variable $ALTIBASE_HOME should be specified.

```
/data/download_back/altibase/PERL-DBD/altibase-perlDBD.tar.gz
gzip –cd altibase-perlDBD.tar.gz | tar –xvf -
```

Makefile is created with install.mk.

```
make -f install.mk
```

Make

The shared library for Altibase perl DBD, altibase.sl, is created. For platforms other than HP, altibas.so is created in blib/arch/auto/DBD/altibase.

Execute "make install” as the super user. Altibase perl DBD is installed in perl.

```
Ex.)
/opt/perl_5.8.8/bin/lib/site_perl/5.8.8/IA64.ARCHREV_0/auto/DBD/altibase
```

Set LD_PRELOAD.

This task should be performed for HP or certain other platforms. Otherwise, an error message will be displayed.

Execute perl test.pl.

After starting the Altibase server, change the test.pl code as in the below and execute perl test.pl.

```
my $dbh = DBI->connect("dbi:altibase:DSN=127.0.0.1;UID=SYS;PWD=MANAGER;CONN-
TYPE=1;NLS_USE=US7ASCII;PORT_NO=20999", "", "", {'RaiseError' => 1});
```

4 .NET Data Provider

This chapter describes how to Microsoft's ADO.NET interface with Altibase DBMS.

.NET Data Provider Overview

Overview

Altibase5 .NET Data Provider is implementation of Microsoft's ADO.NET interface for Altibase DBMS. That is, Altibase5 .NET Data Provider is required to use Altibase5 for .NET Framework-based applications.

ADO.NET interface uses .NET Framework to access data sources such as DBMS. .NET Framework provides access methods for OLEDB, SQL Server, ODBC and ORACLE by default. The role of the .NET Framework-based Data Provider is to access data sources, execute commands and retrieve the results. These results are processed by the developer through ADO.NET DataSet classes and then applied back to the data sources.

The biggest advantage of the ADO.NET interface is that even if Altibase is upgraded, the user can continue to use Data Provider or applications without having to change them, as long as ODBC drivers for the corresponding version are available.

For more information on ADO.NET, please refer to Microsoft website (http://www.msdn.com).

Requriements

- .NET Framework
  - — You should install .NET Framework depending on the version of ADO.NET. If you use ADO.NET 1.0, subsequent release from starting with version 1.1. of .NET Framework should be installed. If you use ADO.NET 2.0, subsequent release from starting with version 2.0 SP1 of ADO.NET should be installed.
  - — You should use suitable library for version of ADO.NET because .NET Data Provider provides library file depending on the version of ADO.NET.
- ALTIBASE CLI

  You should install ALTIBASE CLI library because .NET Data Provider connects to database with CLI library.

Restriction

- .NET Data Provider enables you to use CLI libarary with odbccli_sl.dll suited for system.
- You should use .NET Data Provider built for .NET Framework because implementation is different depending on the version of .NET Framework.
- ColumnName performs case-sensitive in DataReader and CommandBuilder. If you don't place column name inside quotation maks when creating table, column name is written in capital letters. In this case, you should write column name in capital letter for correct value.
- Data loss can occur when you retrieve data whose size is greater than the value of System.Decimal because data type, becomes specified as decimal .NET when you retrieve data which has NUMBER, NUMERIC, FLOATand DECIMAL types by using DataReader.GetValue().

Using .NET Data Provider

Altibase .NET Data Provider can be found in the "lib" folder under the Altibase installation directory (environment variable %ALTIBASE_HOME%). The path is as follows:

```
%ALTIBASE_HOME%\lib\Altibase.Data.AltibaseClient.dll
```

Compiling Application

The user can use Altibase .NET Data Provider to compile applications in one of two ways:

Compilation with Command

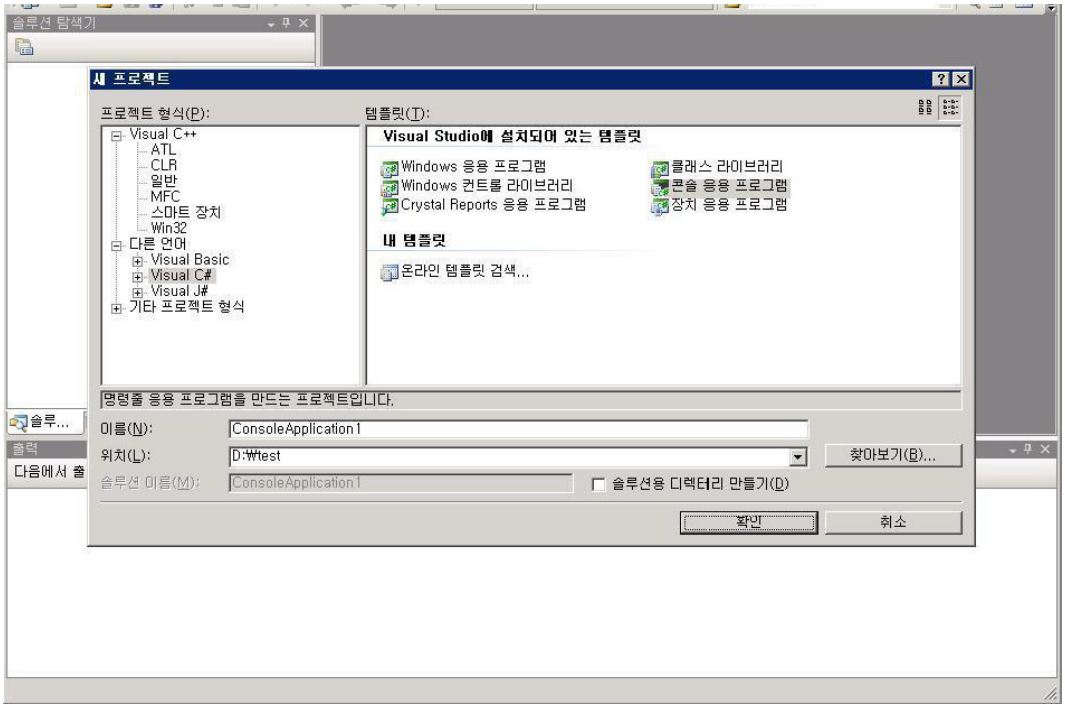
To compile codes with a command, refer to DLL as in the below:

```
csc /r:%ALTIBASE_HOME%\lib\Altibase.Data.AltibaseClient.dll program_name.cs
```

Compilation in IDE Envrionment

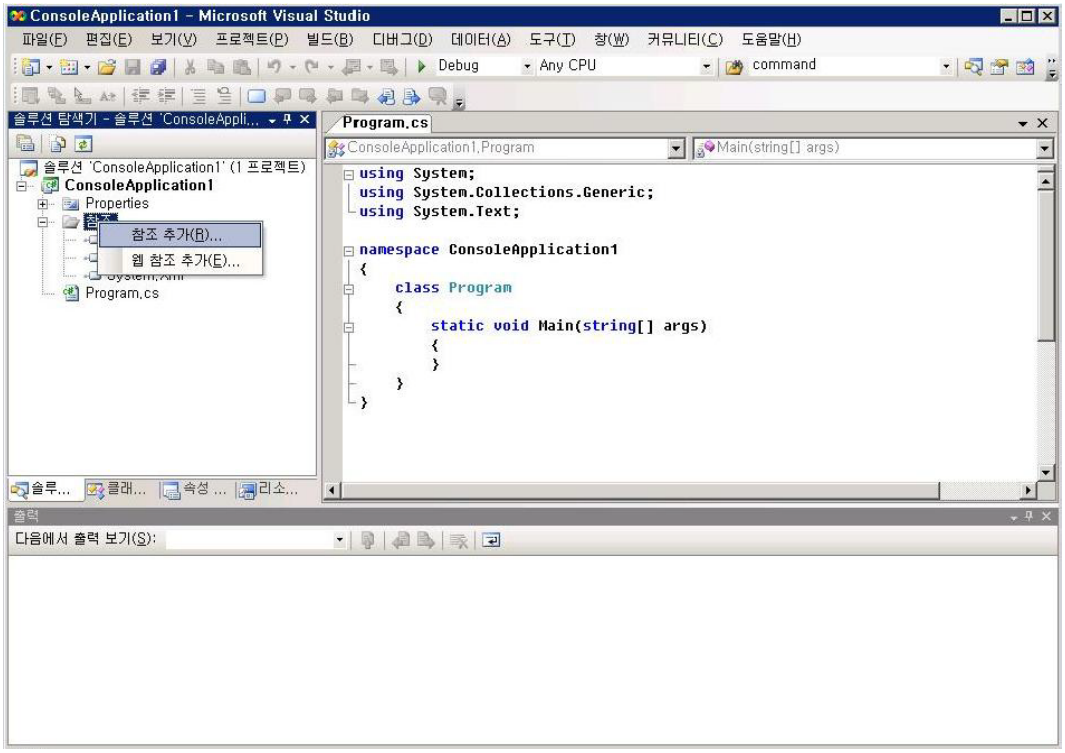
The following example shows how .NET Data Provider is registered in IDE environment.

Figure 4-1 Opening New Project



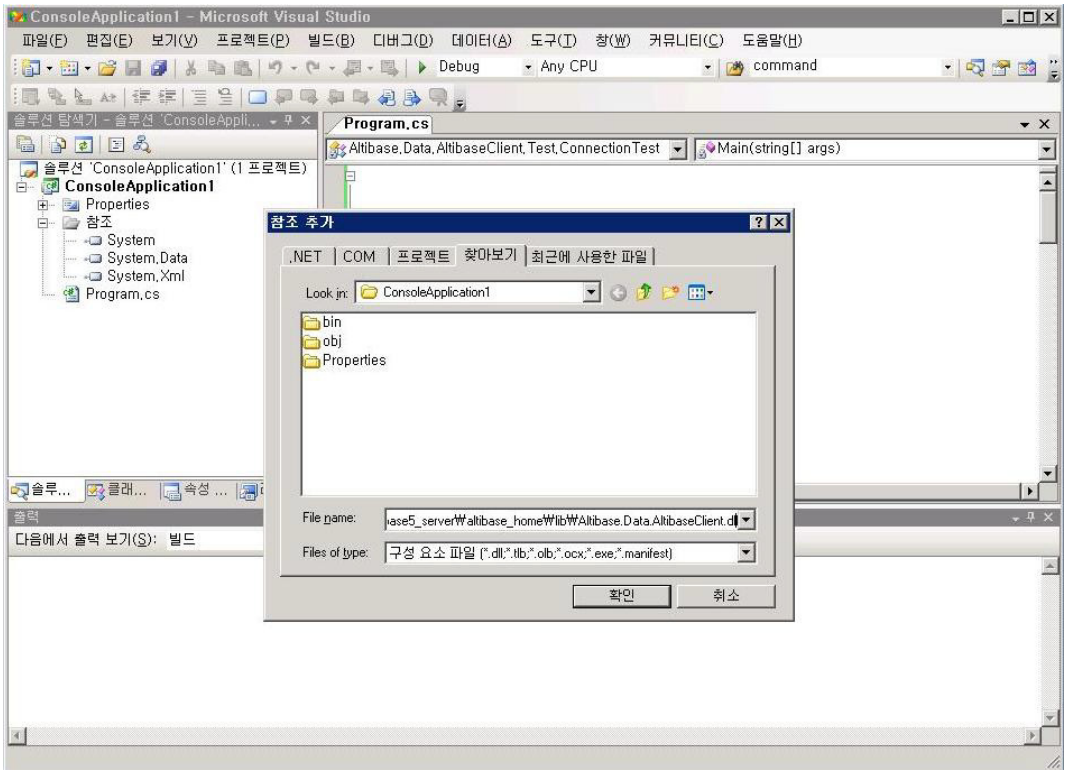
Open a new project [Figure 4-1].

Figure 4-2 Add Reference to Project



Click the Add Reference to Project menu to register .NET Data Provider [Figure 4-2].

Figure 4-3 Register Altibase .NET Data Provider with Project



Locate and register Altibase.Data.AltibaseClient.dll in the lib directory under the Altibase installation directory. The Altibase installation directory can be found in the environment variable %ALTIBASE_HOME%.

Build a project and execute the created executable file.

Binding Array

ALTIBASE .NET Data Provider can bind array to the parameter.

It has become much faster because it executes several arrays with less network cost than typical method, and supports not output and input/output parameter but input parameter currently.

Sequential order is as follows for binding array.

1. You should declare all variables for binding array. Array size should be greater than ArrayBind-Count value of AltibaseCommand class.

2. You should bind arrayed variables to the parameter. If column has char, varchar and blob types, you should specify ArrayBindSize of AltibaseParameter class as the largest size in array elements.

3. You should set ArrayBindCount value of AltibaseCommand class.

```
ex : You may insert 100 values of ArrayBindCount at a time.ArrayBindCount
= 100;
```

4. You should execute SQL statements.

Restriction

You should consider the followings when binding an array.

- Valid range of ArrayBindCount is from 1 to 65535. You should specify array size to the appropriate amount because this process can't be faster even if making array size lager.
- Error occurs if data length of single element array is greater than the value of ArrayBindSize in case that data has char, varchar and blob types.
- If database supports nchar and nvarchar, you should specify value unit of ArrayBindSize as not byte but the number of characters.
- If data has blob type, array type is object and array element is byte[].

  ```
  Ex : byte[] var1; byte[] var2; Object[] var = new Object[2] {var1, var2};
  ```

- Clob, byte, nibble, bit, varbit and geometry are not supported for binding an array.

Example

```
using System;
using System.Data;
using Altibase.Data.AltibaseClient;

class ArrayBind
{
 static void Main()
 {
 AltibaseConnection con = new AltibaseConnection();
 con.ConnectionString = "DSN=127.0.0.1;UID=sys;PWD=man-
ager;NLS_USE=KO16KSC5601";
 con.Open();
 Console.WriteLine("Connected successfully");

 const int arrayBindCount = 3;
 int[] c1 = new int[arrayBindCount] { 100, 200, 300};
 String[] c2 = new String[arrayBindCount] { "APPLE", "ORANGE", "GRAPE" };

 AltibaseCommand cmd = new AltibaseCommand();
 cmd.Connection = con;

 // table info ("orange" needs 12 bytes (utf16))
 // create table t1 (c1 int, c2 varchar(12))";
 //=====================================================
 // bind parameters
 //=====================================================

 cmd.CommandText = "insert into t1 values (?, ?)";
 AltibaseParameter prm1 = new AltibaseParameter("c1", DbType.Int32);
 prm1.Direction = ParameterDirection.Input;
 prm1.Value = c1;

 AltibaseParameter prm2 = new AltibaseParameter("c2", DbType.AnsiString);
 prm2.Direction = ParameterDirection.Input;
 prm2.Value = c2;
 prm2.ArrayBindSize = 12; // max element size in bytes
```

```
 cmd.Parameters.Add(prm1);
 cmd.Parameters.Add(prm2);

 //=====================================================
 // execute
 //=====================================================
 cmd.ArrayBindCount = arrayBindCount;
 cmd.ExecuteNonQuery();

 //=====================================================
 // select
 //=====================================================
 IDataReader sDataReader = null;
 cmd.Parameters.Clear();
 cmd.CommandText = "select * from t1";

 sDataReader = cmd.ExecuteReader();
 while (sDataReader.Read())
 {
 for (int i = 0; i < sDataReader.FieldCount; i++)
 {
 Console.Write("[" + sDataReader.GetValue(i) + "] ");
 }
 Console.WriteLine();
 }
 sDataReader.Close();
 con.Close();
 con.Dispose();
 }
}
```

Declaration

Declare Altibase Data Provider classes for use as in the below:

using Altibase.Data.AltibaseClient;

Transaction Process

You can process transaction with interface provided by ADO.NET.

In this way you should use AltibaseConnection.BeginTransaction() and AltibaseTransaction.

```
AltibaseConnection sConn = new AltibaseConnection(sConnStr);
sConn.Open();

// Transaction starts.
AltibaseTransaction sTrans = sConn.BeginTransaction();

AltibaseCommand sCmd = sConn.CreateCommand();
// Transaction Process ...
// TODO

// Transaction terminate
s.sTrans.Commit();
```

Schema

ALTIBASE supports GetSchema() method being used to return general schemas such as MetadataCollections, DataSourceInformation, DataTypes, Restrictions and ReservedWords, and schema containing information of meta table additionally as follows.

Table 4-1 Schema of Meta Table supported by ALTIBASE

Schema	Meta Table	Description
Users	SYS_USERS_	User meta table
Tables	SYS_TABLES_	Table meta table
Views	SYS_VIEWS_	View meta table
Sequences	V$SEQ	Sequence information
Synonyms	SYS_SYNONYMS_	Synonym meta table
Indexes	SYS_INDICES_	Meta table containing index information
Columns	SYS_COLUMNS_	Column meta table
Constraints	SYS_CONSTRAINTS_	Meta table containing constraint
Procedures	SYS_PROCEDURES_	Meta table of stored procedure and function
ProcedureParameters	SYS_PROC_PARAS_	Information of parameter used in stored procedure

For more information of schemas supported by ALTIBASE, see data dictionary of Administrator's Manual.

Class

Altibase .NET Data Provider provides functions for database connection, query execution and result browsing, and it uses four types of class in [Table 4.2]to perform these functions. For information on the functionality of sub methods in each class, please refer to Microsofts' ADO.NET standards.

Table 4-2 Classes for Connection, Query Execution and Result Retrieval

Class	Description
AltibaseConnection	Establish a connection to a database and start a transaction.
AltibaseCommand	Execute a query at a database and display parameters.
AltibaseDataReader	Retrieve and output the result of command execution from a database.

Class	Description
AltibaseDataAdapter	Fill DataSet with data and update data stored in a database.

Altibase .NET Data Provider provides the followng classes for exception handling, stored procedure and transaction processing.

Table 4-3 Classes for Exception Handling and Transaction Processing

Class	Description
AltibaseException	Display a database error, or a client error received from .Net Framework.
AltibaseParameter	Define I/O parameters for a command or stored procedure.
AltibaseTransaction	Allow execution of a transaction command in a database.

Data Type

AltibaseType class has been defined to declare the data type of table columns or AltibaseParameter. The [Table 4-4] shows the relationship the data types provided by Altibase and .NET Framework.

Table 4-4 The Relationships between Data Types

AltibaseType	ALTIBASE	.NET Framework
BigInt	BIGINT	Int64
Bit	BIT	Byte[]
Blob	BLOB	Byte[]
Byte	BYTE	Byte[]
Char	CHAR	String
Clob	CLOB	String
DateTime	DATE	DateTime
Decimal	DECIMAL	Decimal
Double	DOUBLE	Double
Float	FLOAT	Decimal
Geometry	GEOMETRY	Byte[]
Integer	INT	Int32
NChar	NCHAR	String
Nibble	NIBBLE	Byte[]

AltibaseType	ALTIBASE	.NET Framework
Number	NUMBER	Decimal
Numeric	NUMERIC	Decimal
NVarChar	NVARCHAR	String
Real	REAL	Float
SmallInt	SMALLINT	Int16
VarBit	VARBIT	Byte[]
VarChar	VARCHAR	String

You should add 'N' in front of character string to use constant character string with national character type in SQL statements.

Interface Setting for .NET Data Provider

Interface Setting

Independent programming is available for ADO .NET Provider (subsequent releases starting with version 2.0).

DbProviderFactories class enables user to obtain DbProviderFactory class of provider, which he requires. Therefore, he can create and use DbConnection or DbCommand objects with DbProviderFactory class.

However, factory information of .NET Data Provider should be registered in the setting file of .NET Framework to use DbProviderFactories.

There are two ways to register factory information.

- To register .NET Framework in the setting file
- To use the environment setting file of application

Environment setting file is in xml file format. You should add information of .NET Data Provider in subcategory of system.data - DbProviderFactories.

How to register the setting file of .NET Framework

Setting file of .NET Framework is saved in its subcategory, CONFIG folder, as machine.config. .NET Framework is tipycally installed in %windir%\Microsoft.NET\Framework for each its version. The location of setting file in .NET Framework 2.0, is as follows.

```
c:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config
```

You should open machine.config and add information of .NET Data Provider to subcategory of system.data - DbProviderFactories as follows.

```
<add name="Altibase Data Provider"
 invariant="Altibase.Data.AltibaseClient"
 description="Altibase Data Provider"
 type="Altibase.Data.AltibaseClient.AltibaseFactory,
 Altibase.Data.AltibaseClient" />
```

How to use environment setting file of application

Environment setting file of application is saved as "the name of executable file.config" in the directory where executable file exists. For example, environment setting file of application is AltiTest.exe.cofig if executable file is AltiTest.exe.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <system.data>
 <DbProviderFactories>
 <remove invariant="Altibase.Data.AltibaseClient" />
 <add name="Altibase Data Provider"
 invariant="Altibase.Data.AltibaseClient"
 description="Altibase Data Provider"
 type="Altibase.Data.AltibaseClient.AltibaseFactory,
```

```
 Altibase.Data.AltibaseClient" />
 </DbProviderFactories>
 </system.data>
</configuration>
```

Unsupported Interface

ALTIBASE doesn't support interfaces depending on the version of ADO.NET as follows. All unsupported components occur NotImplementedException.

Unsupported Interfaces in ADO.NET 1.X

Class	Identification	Component
AltibaseConnection	M	ChangeDatabase
	M	Clone
	P	Database
AltibaseCommand	M	Cancel
	M	Clone
	P	CommandTimeout
	P	CommandType
AltibaseDataReader	M	GetData
	P	Depth
AltibaseParameter	M	Clone
AltibaseParameterCollection	M	AddRange

- AltibaseDataReader.Depth always returns 0.

Unsupported Interfaces in ADO.NET 2.0

ADO.NET 2.0 also doesn't support interfaces, which ADO.NET 1.X doesn't.

Class	Identification	Component
AltibaseFactory	M	CreateDataSourceEnumerator
	M	CreatePermission

Class	Identification	Component
AltibaseConnection	M	EnlistTransaction(Transaction transcation)
	P	DataSource
	P	ServerVersion
AltibaseDataReader	M	GetDbDataReader
	M	GetProviderSpecificFieldType
	M	GetProviderSpecificValue
	M	GetProviderSpecificValues
	P	HasRows
	P	VisibleFieldCount
AltibaseDataAdapter	M	AddToBatch(IDbCommand command)
	M	CrearBatch
	M	ExecuteBatch
	M	GetBatchedParameter
	M	GetBatchedRecordsAffected(int commandIdentifier, out int recordsAffected, out Exception error)
	M	InitializeBatching
	M	TerminateBatching
AltibaseParameter	M	ResetDbType
AltibaseDataSourceEnumerator	A	
AltibasePermission	A	
AltibasePermissionAttribute	A	

- When the class is inherited, that method must be implemented, or the inheriting class becomes abstract as well. This method allows you to use basic method. The following members support basic implementation.

  — CreateCommandBuilder
  — CreateConnectionStringBuilder
  — CreateDataSourceEnumerator
  — CreatePermission
  — GetProviderSpecificFieldType
  — GetProviderSpecificValue
  — GetProviderSpecificValues
  — VisibleFieldCount
  — GetBatchedRecordsAffected(int commandIdentifier, out int recordsAffected, out Excep-

tion error)

- Member occurs NotImplementedException if it doesn't support basic implementation.

.NET Data Provider Example

You can contact ALTIBASE with AltibaseConnection and then create test_goods table. You should look up data after inserting them.

```
using Altibase.Data.AltibaseClient;

class ConnectionTest
{
 static void Main(string[] args)
 {
 string sConnectionString = "DSN=127.0.0.1;PORT_NO=20091;UID=sys;PWD=man-
ager;";
 // This writes host IP address of database server contacting DSN variable
and port of database server for PORT_NO.
 AltibaseConnection conn = new AltibaseConnection(sConnectionString);

 try
 {
 conn.Open(); // This denotes to contact database.
 AltibaseCommand command = new AltibaseCommand("drop table test_goods",
conn);

 try {
 command.ExecuteNonQuery(); // This executes queries.
 } catch (Exception ex) {}
 command.CommandText =
 "create table test_goods (
 gno char(10),
 gname char(20),
 location char(9),
 stock integer,
 price numeric(10, 2))";
 command.ExecuteNonQuery(); // This executes queries.
 command.CommandText =
 "insert into test_goods values
 ('A111100001','IM-300','AC0001',1000,78000)";
 command.ExecuteNonQuery();//This executes queries.
 command.CommandText =
 "insert into test_goods values
 ('A111100002','IM-310','DD0001',100,98000)";
 command.ExecuteNonQuery();//This executes queries.
 command.CommandText =
 "insert into test_goods values
 ('B111100001','NT-H5000','AC0002',780,35800)";
 command.ExecuteNonQuery();//This executes queries.
 command.CommandText = "select * from test_goods";
 AltibaseDataReader dr = command.ExecuteReader();
 Console.WriteLine(" GNO GNAME LOCATION STOCK PRICE ");
 Console.Write-
Line("==========================================================================
=============");
 while (dr.Read())
 {
 for (int i = 0; i < dr.FieldCount; i++)
 {
 Console.Write("\t{0}", dr[i]);
 }
 // This prints the retrived data.
 Console.WriteLine();
 }
 }
 catch (Exception ex)
```

```
 {
 Console.WriteLine(ex.ToString());
 }
 conn.Close(); // This shuts down database.
 }
}
```

```
Execution Results
GNO GNAME   LOCATION   STOCK  PRICE
================================================
A111100001  IM-300     AC0001  1000    78000
A111100002  IM-310     DD0001  100     98000
B111100001  NT-H5000   AC0002  780     35800
```

5 OLE DB

This chapter describes OLE DB, which Altibase supports. You can know how to install and delete OLE DB, and how to access in variable environment such as COM, ADO, ADO.NET and etc. And you can also know data types and properties, which Altibase support for OLE DB.

Overview of OLE DB

OLE DB(Object Linking and Embedding database) indicates the open interface, which Microsoft makes for access to general purpose database.

ODBC supports common API for access to database, but has limits of designing and building data based application in new way.

OLE DB as the new alternative supports open database access with new functionalities. OLE DB supports access to all types of data such as relational, non-relational and hierarchical data with using COM based programming interface.

Installation

This chapter describes installation and uninstallation of OLE DB. You can know the confirmation to use the existing ODBC driver after OLE DB installation and how to access in variable operation environment.

Installation and Uninstallation

You can install OLE DB with the following command because OLE DB isn't registered automatically when installing Altibase.

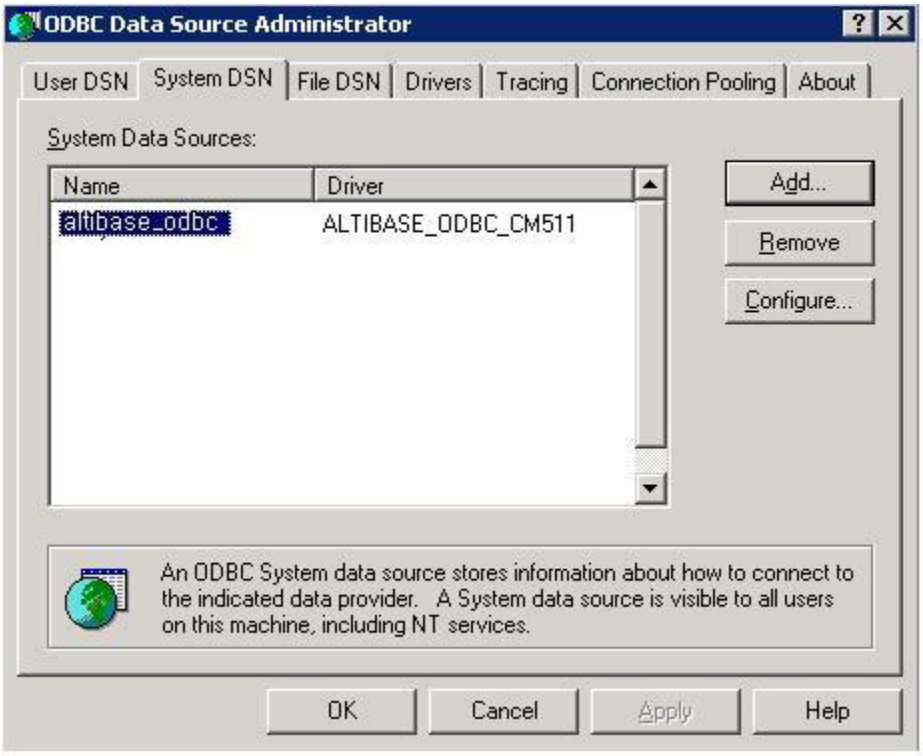
```
regsvr32.exe altioledb.dll
```

Use the following command to uninstall OLE DB.

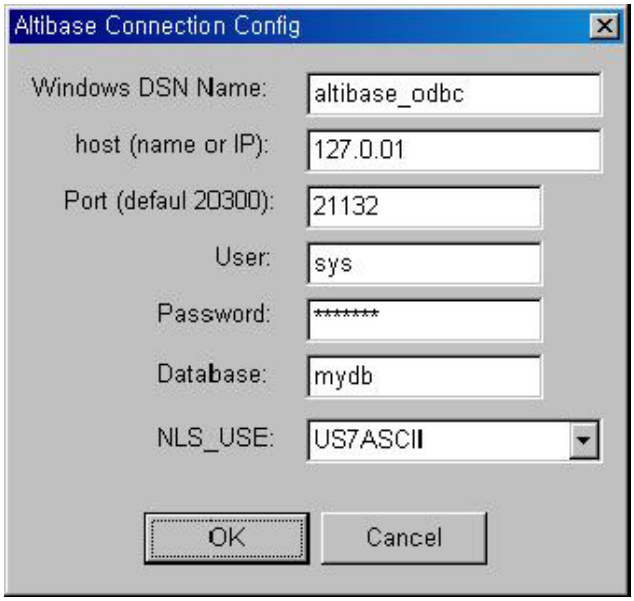
```
regsvr32.exe /u altioledb.dll
```

Confirmation after Installation

Altibase OLE DB uses ODBC driver. Therefore, you may confirm whether altiodbc.dll file exists in SYSTEM32 folder of WINDOWS or not.



If the file doesn't exist, you may register as the following picture because you use DSN name, which is registered in ODBC manager as the access string. You can register DNS name in setting - management tools - data source.



Access Methods

You can know access methods to use OLE DB in variable environment such as COM, ADO, ADO.net and etc. as follows. You can use connection strings, which Altibase driver supports for access.

Access at COM

All types of data access are available with using COM based programming interface. OLE DB is used through data provider with interface(altioledb.dll) to use COM based OLE DB efficiently.

```
CLSIDFromProgID(OLESTR("Altibase.OLEDB"), &clsid); // This outputs the value
of CLSID as the name of OLE DB.
CoCreateInstance(clsid, NULL, CLSCTX_INPROC_SERVER, IID_IDBInitialize,
(void**)&pIDBInitialize);

InitProperties.dwPropertyID = DBPROP_INIT_DATASOURCE;
InitProperties.vValue.vt = VT_BSTR;
InitProperties.vValue.bstrVal = SysAllocString((LPOLESTR)L"altibase_odbc");
// This indicates DSN name registered in ODBC manager.
```

Access at ADO

Input Altibase.OLEDB in Altibase provider as follows to access to the registered DSN(altibase_odbc) in ODBC manager at ADO, ADO.net.

```
Dim con As New ADODB.Connection
con.ConnectionString = "Provider=Altibase.OLEDB;DSN=altibase_odbc;"
con.Open
```

Access at ADO.net

```
using System.Data.OleDb // This indicates to use oledb.
```

```
OleDbConnection connection = new OleDbConnection();
connection.ConnectionString =
"Provider=Altibase.OLEDB;DSN=altibase_odbc;";
connection.Open(); // This indicates to access.
```

However, if DSN isn't registered in ODBC manager in case of accessing at ADO, ADO.net or you use OLE DB with inputting the server address manually, you can access with the following strings.

```
Provider=Altibase.OLEDB;UID=sys;PWD=manager;DSN=127.0.0.1;Alti-
base='PORT=21132;NLS_USE=US7ASCII';
```

```
    Provider=Altibase.OLEDB   Essential Requirements for input
    UID                       User ID
    PWD                       User Password
    DSN                       Server name registered in ODBC manager
    or server IP address
    Altibase =                Altibase Connection String
```

You can use Altibase connection string when you specify connection option like CLI. The format for specifying properties is as follows.

```
Altibase='Property=Value; Property=Value; Property=Value'
```

Each property is connected with ';', and you may input the property values in single quotation marks('').

Data Types

The following table indicates how data type conversion is between Altibase and OLE DB.

Altibase	OLE DB
BIGINT	DBTYPE_I8
BIT	DBTYPE_BOOL
BLOB	Not Support
BYTE	DBTYPE_BYTES
CHAR	DBTYPE_STR
CLOB	Not Support
DATE	DBTYPE_STR
DOUBLE	DBTYPE_R8
FLOAT	DBTYPE_STR
GEOMETRY	Not Support
INTEGER	DBTYPE_I4
NIBBLE	Not Support

Altibase	OLE DB
NUMERIC	DBTYPE_STR
DECIMAL	DBTYPE_STR
REAL	DBTYPE_R4
SMALLINT	DBTYPE_I2
VARBIT	DBTYPE_BOOL
VARCHAR	DBTYPE_STR

OLE DB doesn't support the conversion of types such as BLOB, CLOB, GEOMETRY, NIBBLE and etc. BIT, VARBIT is translated in BOOL type.

The following table indicates data types when their OLE DB is translated in Altibase.

OLE DB	Altibase
DBTYPE_I2	SMALLINT
DBTYPE_I4	INTEGER
DBTYPE_R4	REAL
DBTYPE_R8	DOUBLE
DBTYPE_CY	BIGINT
DBTYPE_UI1	SMALLINT
DBTYPE_I1	SMALLINT
DBTYPE_UI2	INTEGER
DBTYPE_UI4	BIGINT
DBTYPE_I8	BIGINT
DBTYPE_UI8	BIGINT
DBTYPE_BYTES	BYTE
DBTYPE_STR	VARCHAR
DBTYPE_WSTR	VARCHAR
DBTYPE_DBDATE	DATE
DBTYPE_DBTIME	DATE
DBTYPE_DBTIMESTAMP	DATE

Restriction

You should add 'N' in front of character string to use constant character string with national character type in SQL statemsnts.

Interface

Altibase supports OLE DB interface as follows.

OLEDB CoType	Support
Enumerators	X
Data source objects	O
Sessions	O
Commands	O
Multiple results objects	O
Rowsets	O
Views	X
Indexes	X
Transactions	X
Transaction options	X
Error	O
Custom errors	O
Error records	O
Binder objects	X
Row objects	O
Stream objects	X

Enumerators

Interface Name	Support
IParseDisplayName	X
ISourcesRowset	X
IDBInitialize	X
IDBProperties	X
ISupportErrorInfo	X

Data Source Objects

Interface Name	Support
IDBCreateSession	O
IDBInitialize	O
IDBProperties	O
IPersist	O
IConnectionPointContainer	X
IDBAsynchNotify	X
IDBAsynchStatus	X
IDBDataSourceAdmin	X
IDBInfo	O
IObjectAccessControl	X
IPersistFile	X
ISecurityInfo	X
ISupportErrorInfo	O
ITrusteeAdmin	X
ITrusteeGroupAdmin	X

Sessions

Interface Name	Support
IGetDataSource	O
IOpenRowset	O
ISessionProperties	O
IAlterIndex	X
IAlterTable	X
IBindResource	X
IConnectionPointContainer	X
ICreateRow	X

Interface Name	Support
IDBCreateCommand	O
IDBSchemaRowset	O
IIndexDefinition	X
ISupportErrorInfo	O
ITableCreation	X
ITableDefinition	X
ITableDefinitionWithConstraints	X
ITransaction	Abort isn't supported-GetTransactionInfoisn't supported
ITransactionJoin	X
ITransactionLocal	GetOptionsObjectisn't supported
ITransactionObject	X

Commands

Interface Name	Support
IAccessor	AddRefAccessor isn't supported
IColumnsInfo	O
ICommand	Cancel isn't supported
ICommandProperties	O
ICommandText	O
IConvertType	O
IColumnsRowset	X
ICommandPersist	X
ICommandPrepare	O
ICommandWithParameters	O
ISupportErrorInfo	O
ICommandStream	X

Multiple results objects

Interface Name	Support
IMultipleResults	O
ISupportErrorInfo	O
IDBAsyncStatus	X

Rowset

Interface Name	Support
IAccessor	AddRefAccessor isn't supported
IColumnsInfo	O
IConvertType	O
IParentRowset	X
IRowset	O
IRowsetInfo	GetReferencedRowset isn't supported
IChapteredRowset	X
IColumnsInfo2	X
IColumnsRowset	GetColumnsRowset isn't supported
IConnectionPointContainer	X
IDBAsynchStatus	X
IGetRow	X
IRowsetChange	X
IRowsetChapterMember	X
IRowsetCurrentIndex	X
IRowsetFind	X
IRowsetIdentity	X
IRowsetIndex	X
IRowsetNotify	X

Interface Name	Support
IRowsetLocate	X
IRowsetRefresh	X
IRowsetResynch	X
IRowsetScroll	X
IRowsetUpdate	X
IRowsetView	X
ISupportErrorInfo	O
IRowsetBookmark	X

Views

Interface Name	Support
IColumnsInfo	X
IAccessorX	X
ISupportErrorInfo	X
IViewChapter	X
IViewFilter	X
IViewRowset	X
IViewSort	X

Indexes

Interface Name	Support
IAccessor	X
IColumnsInfo	X
IConvertType	X
IRowset	X
IRowsetIndex	X
IRowsetInfo	X

Interface Name	Support
IRowsetChange	X
IRowsetCurrentIndex	X
IRowsetFind	X
IRowsetIdentity	X
IRowsetLocate	X
IRowsetNotify	X
IRowsetRefresh	X
IRowsetResynch	X
IRowsetScroll	X
IRowsetUpdate	X
IRowsetView	X
ISupportErrorInfo	X

Transactions

Interface Name	Support
IConnectionPointContainer	X
ITransaction	X
ISupportErrorInfo	X

Transaction Options

Interface Name	Support
ITransactionOptions	X
ISupportErrorInfo	X

Error

Interface Name	Support
IErrorRecords	O

Custom Errors

Interface Name	Support
ISQLErrorInfo	O

Error Records

Interface Name	Support
IErrorInfo	O

Binder Objects

Interface Name	Support
IBindResource	X
ICreateRow	X
IDBBinderProperties	X
IRegisterProvider	X
ISupportErrorInfo	X

Row Objects

Interface Name	Support
IColumnsInfo	O

Interface Name	Support
IConvertType	O
IGetSession	O
IRow	GetSourceRowsetisn't supported
IColumnsInfo2	X
IConnectionPointContainer	X
ICreateRow	X
IDBAsynchStatus	X
IDBCreateCommand	X
IDBInitialize	X
IRowChange	X
IRowSchemaChange	X
IScopedOperations	X
ISupportErrorInfo	O

Stream Objects

Interface Name	Support
ISequentialStream	X
IConnectionPointContainer	X
IDBAsynchStatus	X
IDBInitialize	X
IGetSourceRow	X
ISupportErrorInfo	X
IStream	X

Examples

ADO Examples

```
Dim adoRst
Dim adoCnn
Dim rsAddRtn
Dim strSQL
Dim strCnnStr
Dim a, b, c

strSQL = "SELECT * FROM T1"

WScript.StdOut.Write "Program Start"
WScript.StdOut.WriteBlankLines 1

set adoCnn = CreateObject("ADODB.Connection")
set adoRst = CreateObject("ADODB.Recordset")
adoRst.cursorlocation = 3
' This indicates to access.
adoCnn.Open "Provider=Altibase.OLEDB;DSN=altibase_odbc"
' This indicates to output records.
adoRst.Open strSQL, adoCnn

For a=0 To adoRst.RecordCount -1
b=adoRst.Fields(0)
c=adoRst.Fields(1)

WScript.StdOut.Write b
WScript.StdOut.WriteBlankLines 1
WScript.StdOut.Write c
WScript.StdOut.WriteBlankLines 1

adoRst.MoveNext
Next

WScript.StdOut.Write adoRst.RecordCount
WScript.StdOut.WriteBlankLines 1
adoRst.Close
```

ADO.NET Examples

```
using System;
using System.Text;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;

namespace Open
{
class Program
{
static void Main(string[] args)
{
IDbConnection sConnection = null;
IDbCommand sCommand = null;
IDataReader sDataReader = null;

// This indicates to specify the connection string.
```

```
sConnection = new OleDbConnection("Provider=Altibase.OLEDB;DSN=" + args[0]);

try
{
sConnection.Open(); // This indicates to access to DB.
Console.WriteLine("Connection Success");
sCommand = sConnection.CreateCommand();
Console.WriteLine("CreateCommand Success");

// This indiactes to specify SQL statement.
sCommand.CommandText = "select * from t1";
sCommand.CommandType = CommandType.Text;
sDataReader = sCommand.ExecuteReader();
Console.WriteLine("ExecuteReader Success");

// This indicates to output data.
while (sDataReader.Read())
{
// This indicates to output data as many as columns.
for (int i = 0; i < sDataReader.FieldCount; i++)
{
Console.WriteLine("GetDataTypeName : " + sDataReader.GetDataTypeName(i));
Console.WriteLine("IsDBNull : " + sDataReader.IsDBNull(i));
Console.WriteLine("Value : " + sDataReader.GetValue(i));
}
}
sDataReader.Close();

// This indicates to end the access.
sConnection.Close();
Console.WriteLine("Close Success");
}
catch(Exception e)
{
Console.WriteLine(e.Message);
}
}
}
}
```

6 XA Interface

This chapter introduces XA basic concept, interface, limitation, and XA functions supported by ALTIBASE, and also describes how to use global transaction in ODBC, SES and JDBC, and how to deal with problems occurred by application.

XA Interface

XA is a standard interface used for distributed transaction (or global transaction) processing specified by X/Open.

Distributed transaciton is called global transaction among systems in more than 2 networks. System has resource role of transaction, and TM creates and manages transaction for all functions related to this resource. In other words, XA enables distributed application to share resource provided by multiple database servers or execute transaction globally.

XA is useful for application processing transaction in more than one database.

XA Glossary

- Global Transaction

  This denotes total transactions processed by XA and is called distributed transaction.

- Transaction Branch

  Each of the RM's internal units of work in support of a global transaction is part of exactly transaction branch. There are a one-to-one correlation between one transaction branch and one XID (transaction ID of XA).

- In-doubt Transaction

  This denotes transaction branch until RM is reported at the prepare before receiving a commit or rollback, and is called pending transaction.

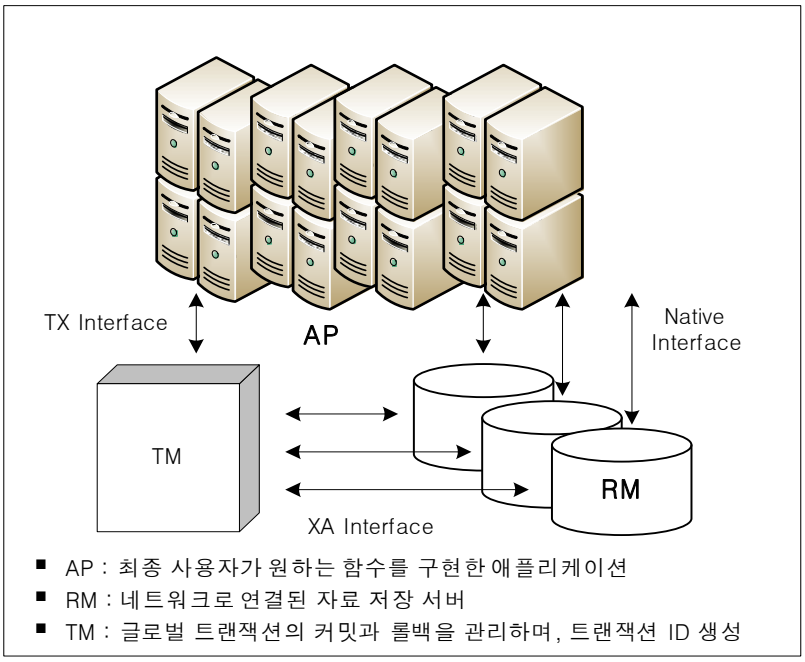
- Heuristic Transaction

  An heuristic completion occurs when a resource makes a unilateral decision during the completion stage of a distributed transaction to commit or rollback updates. Network failures or transaction timeouts are possible causes for heuristic completion such as heuristic commit and heuristic rollback.

XA Structure

XA structure is composed of AP(Application Program), TM(Transaction Manager) and RM(Resource Manager).

Figure 6-1 XA Structure



If XA reports TM that AP starts occurring distributed transaction, TM checks what database is the target of distributed transaction in RM. TM creates XID for transaction branch to be executed in RM, and then sends it to RM.

This XID starts to occur distributed transaction in database node.

And transactions sent from AP are regarded as those sent from XID until they terminate.

After they terminate, AP reprts TM that distributed transaction stops occurring . TM forces RM occurring distributed transaction with XID to make a commit or rollback.

XA and 2PC

ALTIBASE XA Interface performs 2-phase commit functionality. 2PC operation has prepare and commit steps.

In prepare step as the first phase of a 2PC, TM checks possibility of a commit in RM, that is, all database nodes participated in occurring distributed transaction. If committing the transaction, RM sends a prepare message to TM. Otherwise, RM returns value to roll back this.

In commit step as the second phase of a 2PC, TM awaits prepare acknowledgements. If receiving them normally, TM agrees for total RMs to make a commit decision. However, if even one RM doesn't prepare a transaction, TM agrees for it to make a rollback decision.

xa_switch_t Structure

The following is structure of xa_switch_t.

ALTIBASE provides it named as altibase_xa_switch including information of RM and entry point.

```
struct xa_switch_t {
 char name[RMNAMESZ];                /* name of resource manager */
 long flags;                         /* resource manager specific options */
 long version;

 int (*xa_open_entry)(/*_ char *, int, long _*/);
 /*xa_open fn pointer*/
 int (*xa_close_entry)(/*_ char *, int, long _*/);
 /*xa_close fn pointer*/
 int (*xa_start_entry)(/*_ XID *, int, long _*/);
 /*xa_start fn pointer*/
 int (*xa_end_entry)(/*_ XID *, int, long _*/);
 /*xa_end fn pointer*/
 int (*xa_rollback_entry)(/*_ XID *, int, long _*/);
 /*xa_rollback fn pointer*/
 int (*xa_prepare_entry)(/*_ XID *, int, long _*/);
 /*xa_prepare fn pointer*/
 int (*xa_commit_entry)(/*_ XID *, int, long _*/);
 /*xa_commit fn pointer*/
 int (*xa_recover_entry)(/*_ XID *, long, int, long _*/);
 /*xa_recover fn pointer*/
 int (*xa_forget_entry)(/*_ XID *, int, long _*/);
 /*xa_forget fn pointer*/
 int (*xa_complete_entry)(/*_ int *, int *, int, long _*/);
 /*xa_complete fn pointer*/
};
```

XA Library

Extra library isn't required for connecting to application with ALTIBASE XA because it is included in odbccil library. You should connect application with XA to libodbccli.a for using functionality related to XA.

XA Interface

This is assumed mutual interface between RM and TM. TM consists of XA routine controling RM to execute global trasaction and AX routine which RM requests to TM dynamically. However, you should call xa_start in RM before executing transaction because ALTIBASE doesn't support dynamic registration.

You may use function of altibase_xa_switch in xa_switch_t to use functions related to XA.

XA Functions

ALTBASE uses functions in altibase_xa_switch structure to use XA function.

Table 6-1 XA Interface

XA Interface	Description
xa_open	This denotes to connect to RM.
xa_close	This denotes to interrupt connection from RM.
xa_start	This denotes to start new transaction branch or the existing one again and to be linked to specified XID.
xa_end	This denotes to be separated from transaction branch.
xa_rollback	This denotes to rollback transaction branch related to specified XID.
xa_prepare	This denotes to prepare for a commit of transaction branch.
xa_commit	This denotes to commit transaction branch.
xa_recover	This shows XID list of transaction making prepare, heuristic commit or heurisitc rollback decisions.
xa_forget	This denotes to discard information of transaction branch making a heuristic completion in RM.

xa_open

This denotes to connect to RM.

```
int xa_open(char *xa_info, int rmid, long flags);
```

xa_info is null-terminated character string with server information, and its maximum length is 256byte. This has same format as SQLDriverConnect argument, and parameters such as XA_NAME and XA_LOG_DIR additionally. Refer to SQLDriverConnect in ODBC User's Manual for details about other parameters.

```
NAME=value;NAME=value;NAME=value;…
Example :
DSN=127.0.0.1;UID=SYS;PWD=MANAGER ;XA_NAME=conn1
```

Table 6-2 Parameters Added to XA Interface

XA Parameter	Description
XA_NAME	This is connection identifier in ALTIBASE Precompiler. If you omit this value when writing a application with ALTIABSE Precompiler, default connection is specified instead of this value. If you specify XA_NAME as conn1, you can use this value in AT clause when executing SQL statement.
XA_LOG_DIR	This shows ALTIBASE XA library errors and directory logging with their information. If $ALTIBASE_HOME is specified. default is $ALTIBASE_HOME/trc. Otherwise, default is current directory.

rmid writes server ID for access, and you can set its value randomly.

If flag isn't specified, you should specify it as the following.

- TMNOFLAGS

xa_close

This terminates connection to specifyed RM. However, this returns XA_OK even if xa_close is executed in this situation.

```
int xa_close(char *xa_info, int rmid, long flags);
```

xa_info is character string to write server information, and its maximum length is 256byte.

If flag isn't specified, you should specify it as the following.

- TMNOFLAGS

xa_start

This starts to execute transaction branch. xid is identifier of global transaction.

```
int xa_start(XID *xid, int rmid, long flags);
```

You can specify flags as the followings.

- TMRESUME

  This continues to execute transaction branch suspended.

- TMNOWAIT

  This returns XA_RETRY without a wait when xa_start is blocked.

- TMASYNC

  This starts to execute transaction branch in async mode. (unsupported in ALTIBASE)

- TMNOFLAGS

  This should be specified by default if you don't set flag.

- TMJOIN

  This creates transaction connected to the existing transaction branch.

xa_end

This terminates transaction branch.

```
int xa_end(XID *xid, int rmid, long flags);
```

You can specify flags as the followings.

- TMSUSPEND

  This makes transaction branch suspended, and then terminates it. This transaction branch continues to be executed by xa_start.

- TMSUCCESS

  You can't set this with TMSUSPEND or TMFAIL because this denotes termination successfully.

- TMFAIL

  This denotes termination abnormally. This transaction is specified as roll-back only. You can't use this with TMSUCCES or TMSUSPENT.

xa_rollback

This rolls back operation of transaction branch.

```
int xa_rollback(XID *xid, int rmid, long flags);
```

Flags have the followings.

- TMASYNC

  This executes xa_rollback in async mode (unsupported in ALTIBASE)

- TMNOFLAGS

  This should be specified by default if you don't set flag.

xa_prepare

You can execute this befire withdrawing or reflecting transaction in 2 phase commit protocol.

```
int xa_prepare(XID *xid, int rmid, long flags);
```

Flags have the followings.

- TMASYNC

  (unsupported in ALTIBASE)

- TMNOFLAGS

You should set this by default if you don't specify flag.

Xa_prepare returns the followings.

- XA_RDONLY

  This is returned when transaction occurrence doesn't make data changed. Transaction executed in RM(DBMS) need not commit or roll back.

- XA_OK

  This is returned in case of normal execution.

xa_commit

This reflects certain transaction branch,

```
int xa_commit(XID *xid, int rmid, long flags);
```

Flags have the followings.

- TMONEPHASE

  You can set this for 1 phase commit.

- TMNOFLAGS

  You can set this by default when other flags aren't specified.

xa_recover

This obtains xid list of transaction at the prepare in ALTIBASE server.

```
int xa_recover(XID *xids, long count, int rmid, long flags);
```

This returns the number of recoverd xid. You should set size of xids in count parameter. Flags have the followings.

- TMNOFLAGS

  You can have XID at current location.

xa_forget

This enables ALTIBASE server not to manage transaction completed heuristically.

```
int xa_forget(XID * xid, int rmid, long flags);
```

Flags have the following.

- TMNOFLAGS

  This should be always specified.

xa_complete

This denotes to await until operatoin terminates in case of operation in aync mode. This is unsupported in ALTIBASE and always returns errors.

XA Application Development

ODBC/XA Performance Process

To use XA functions, the user should use altibase_xa_switch functions in xa_switch_t construct.

xa_open

Connects to Resource Manager (RM, DBMS).

```
int xa_open(char *xa_info, int rmid, long flags);
```

xa_info is a null-terminated string. It has server information and its max length is 256 bytes.

rmid contains the ID of a server to connect to and can use any value.

Flags can have the following values:

- TMNOFLAGS

  It must be specified when other flag is not set.

xa_close

Closes a conenction with the specified RM. However, if xa_close is executed for the connection that is closed already, XA_OK will be returned.

```
int xa_close(char *xa_info, int rmid, long flags);
```

xa_info is a string that contains server information and its max length is 256 bytes.

Flags can have the following values:

- TMNOFLAGS

  It must be specified when other flag is not set.

xa_start

Starts a transaction branch.

```
int xa_start(XID *xid, int rmid, long flags);
```

xid is an identifier for a global transaction.

Flags can have the following values:

- TMRESUME

  Resumes the suspend transaction branch.

- TMNOWAIT

  When xa_start is blocked, return XA_RETRY without waiting.

- TMASYNC

Starts a transaction branch in the async mode (not supported by Altibase).

- TMNOFLAGS

  It must be specified when no flag is specified.

- TMJOIN

  Creates a transaction that is connected to a transaction branch which exists already.

xa_end

End a transaction branch.

```
int xa_end(XID *xid, int rmid, long flags);
```

Flags can have the following values:

- TMSUSPEND

  Switches a transaction branch to the suspended state and end it. This transaction branch can be executed by xa_start.

- TMSUCCESS

  Indicates successful termination. It cannot be used with TMSUSPEND or TMFAIL.

- TMFAIL

  Indicates abnormal termination. This transaction is specified as roll-back only. It cannot be used with TMSUCCES or TMSUSPEND.

xa_rollback

Rolls back an operation that was executed for the specified transaction branch.

```
int xa_rollback(XID *xid, int rmid, long flags);
```

Flags can have the following values:

- TMASYNC

  Enables xa_rollback in the async mode (not supported by Altibase).

- TMNOFLAGS

  It must be specified when no flag is specified.

xa_prepare

It should be executed before a transaction is committed or undone by the two phase commit protocol.

```
int xa_prepare(XID *xid, int rmid, long flags);
```

Flags can have the following values:

- TMASYNC

(Not supported by Altibase)

- TMNOFLAGS

  It must be specified when no flag is specified.

  It can return the following values:

- XA_RDONLY

  Indicates that a transaction does not change data. A transaction executed for RM (DBMS) does not require commit or rollback.

- XA_OK

  Indicates normal execution.

xa_commit

Commit a specific transaction branch.

```
int xa_commit(XID *xid, int rmid, long flags);
```

Flags can have the following values:

- TMONEPHASE

  It is set when one phase commit should be executed.

- TMNOFLAGS

  It is set when other flags are not specified.

xa_recover

Get the xid list of the prepared transactions from the Altibase server.

```
int xa_recover(XID *xids, long count, int rmid, long flags);
```

It returns the number of xids recovered.

Specify the size of xids in the Count parameter.

Flags can have the following values:

- TMNOFLAGS

  Get XID at the current location.

xa_forget

Have the heuristically completed transaction not managed by the Altibase server.

```
int xa_forget(XID * xid, int rmid, long flags);
```

Flags can have the following values:

- TMNOFLAGS

This value should be always specified.

xa_complete

When an operation is executed in the ASYNC mode, wait until the operation ends. It is not supported by Altibase and it will always return an error message.

Using XA

This section describes basic procedures for using ODBC, SES and JDBC in XA environment.

Executing ODBC/XA

- xa_open

  Connect to the specified server.

- SQLAllocHandle

  To connect to ODBC, create hdbc and henv.

- SQLSetConnectAttr

  Specify ODBC connection for xa connection.

- SQLConnect

  Because the physical connection was established by xa_open, no new connection is created. However, SQLConnect connects to the inside of ODBC and DML operations can not be perfomred without this step.

- xa_start

  Start a transaction branch for a specific xid.

- do_SQL_Statement();

  Execute operations such as SQLPrepare and SQLExecute. If the Commit statement is executed, the server will display an error message.

- xa_end

  End a transaction branch.

- xa_prepare

  Prepare a transaction branch for execution.

- xa_commit

  Execute a transaciton.

- SQLDisconnect

  Switch the internal state of ODBC to disconnected. However, the physical connection established by xa remains as it is.

- xa_close

  Close the xa conenction.

SQLSetConnectAttr

Call SQLSetConnectAttr to enable XA connection to use ODBC connection, ensuring that ODBC applications can use distributed transactions.

Provide the following parameters to associate SQLSetConnectAttr to XA connection:

```
SQLRETURN SQLSetConnectAttr  (
    SQLHDBC        hdbc,
    SQLINTEGER     fAttr,
    SQLPOINTER     vParam,
    SQLINTEGER     sLen);
```

- fAttr : ALTIBASE_XA_RMID

  Use XA connection for the connection specified by hdbc. For more information on XA connection, assign the following structure pointer to vParam.

- vParam

  It has the rmid value that is used when a connection is established by xa_open.

  Use the following parameter to establish a XA connection to a server without specifying rmid.

- fAttr

  SQL_ATTR_ENLIST_IN_DTC or SQL_ATTR_ENLIST_IN_XA

  Connect the current dbc for the first XA connection.

Executing SES/XA

The SES/XA program is an application that is created with SESC in XA environment. The following describes procedures for using SES/XA.

- xa_open

  Create a xa connection.

  If EXEC SQL CONNECT is not executed, the SESC operation for a server results in a disconnection error.

  EXEC SQL CONNECT :usr IDENTIFIED BY :pwd ENABLE XA RMID :rmid or

  EXEC SQL CONNECT :usr IDENTIFIED BY :pwd ENABLE XA

  Create a SESC connection associated with xa connection.

  Specify a host variable that stores the rmid value for the server connected by xa_open after ENABLE XA RMID to execute the connect statement. If rmid is not specifed, associate the current connection with the first xa connection.

- xa_start

  If a xa transaction is not started by xa_start, the EXEC SQL statement will generate an error.

EXEC SQL (DML)

Execute DML operations for a physical database.

- xa_end

  End a xa transaction branch.

- xa_prepare
- xa_commit
- xa_close

  Close the xa conenction to the server. After xa_close is executed, the EXEC_SQL statement displays an error message saying that the connection is closed.

The SES XA Connect Statement

```
EXEC SQL CONNECT :usr IDENTIFIED BY :pwd ENABLE XA RMID :rmid;
```

The usr, :pwd host variable is actually ignored, and the xa connection for the rmid specified in the rmid host variable is used to connecto to the sesc program.

```
EXEC SQL CONNECT :usr IDENTIFIED BY :pwd ENABLE XA;
```

The first xa connection is used as the current connection.

How to write a program in xa_open depending on specifying XA_NAME

Cursor is valid only in transaction when you write XA program.

In other words, cursor should be open after starting to execute transaction, and cursor should be closed before transaction commits/rollbacks.

How to write precompiler program in case of specifying XA_NAME as default connection

XA_NAME should not exist in character string of xa_open to set default connection as follows.

```
DSN=127.0.0.1;UID=SYS;PWD=MANAGER
```

And queries are as follows.

```
EXEC SQL UPDATE emp SET empno = 5;
```

How to write precompiler program in case of specifying XA_NAME as connection.

If you want to specify XA_NAME as connection in precompiler, XA_NAME=conn1 should be included in character string of xa_open.

If you want to write program with default connection and connection in which more then one XA_NAME are specified, you may set the following.

If you specify XA_NAME as conn1 and conn2, open_string is as follows in TM configuration.

```
DSN=127.0.0.1;UID=SYS;PWD=MANAGER;XA_NAME=conn1
DSN=127.0.0.1;UID=SYS;PWD=MANAGER;XA_NAME=conn2
```

```
DSN=127.0.0.1;UID=SYS;PWD=MANAGER
```

You can do the following in function program of application server.

```
EXEC SQL AT conn1 UPDATE emp SET empno = 5;
EXEC SQL AT conn2 UPDATE emp SET empno = 5;
EXEC SQL UPDATE emp SET empno = 5;
```

Executing JDBC/XA

The xa classes that are defined by the Altibase jdbc driver are as in the below:

```
Altibase.jdbc.driver.ABXADataSource
Altibase.jdbc.driver.ABXAResource
Altibase.jdbc.driver.XID
```

The ABXADataSource class is the only one that is direcly used by the user and the remaining classes do not need to be used by the user as they are those implement the JTA interface class.

1. Create ABXADataSource.

   ```
   ABXADataSource xaDataSource = new ABXADataSource();
   xaDataSource.setUrl(args[0]);
   xaDataSource.setUser("SYS");
   xaDataSource.setPassword("MANAGER");
   ```

2. Create XAConnection.

   Create XAConnection by calling the getXAConnection method in the XADataSource class.

   ```
   XAConnection xaConnection = xaDataSource.getXAConnection("SYS”, "MAN-
   AGER:”);
   ```

3. Create XAResource.

   Create XAResource by calling the getXAResource method in the XAConnection class.

   ```
   XAResource xaResource = xaConnection.getXaResource();
   ```

4. Create Connection.

   Create a connection for SQL by calling the getConnection method in the XAConnection class.

   ```
   Connection conn1 = xaConnection.getConnection();
   ```

5. Execute xa functions with XAResource.

   xa functions such as xa_start and xa_end can be executed with methods in the XAResource class.

   ```
   xaResource.start(xid, XAResource.TMNOFLAGS);
   ```

6. Execute SQL with the Connection object.

   ```
   Statement stmt = conn.createStatement();
   int cnt = st
   mt.executeUpdate("insert into t1 values (4321)");
   mt.executeUpdate("insert into t1 values (4321)");
   ```

XA Transaction Control

This section describes how to control transaction in ALTIBASE XA environment.

You can't make a commit or rollback decision with queries in each database in case of using XA library. Instead you should call API to start or stop executing transaction in TM.

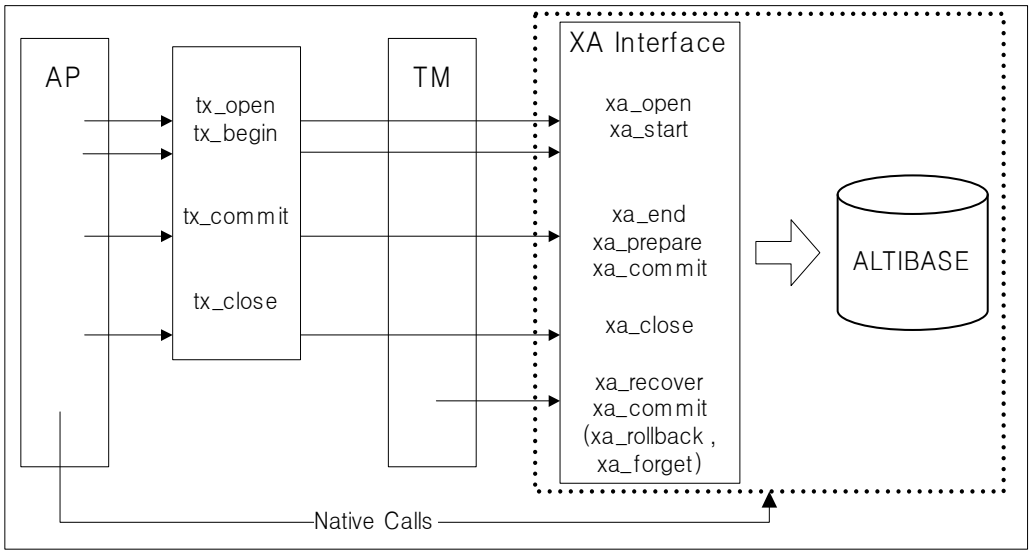
TM controls transaction through TX interface as follows typically.

Table 6-3 TX Interface

TX Interface	Description
tx_open	This denotes to log on RM.
tx_close	This denotes to log out RM.
tx_begin	This denotes to start executing new transaction.
tx_commit	This commits transaction.
tx_rollback	This rollbacks transaction.

The process is as follows to call TX and XA interfaces.

Figure 6-2 The Process of Calling TX and XA Interfaces



TPM application has client/server structure requesting for service, which its client provides in its server. Service is work unit logically. If you use ALTIBASE as RM, unit denotes SQL statement set.

Example

The following example assumes that application server already logs on TPM system.

How to start executing transaction in application server

The following example shows application server starts to execute transaction.

```
Client:
tpm_service("SERVICE1”);

Server:
SERVICE1()
{
<get service specific data>
tx_begin();
EXEC SQL UPDATE....;
tpm_service("SERVICE2");
tx_commit();
<return service status back to the client>
}
```

How to start executing transaction in application client

The following example shows application client starts to execute transaction.

```
Client:
tx_begin();
tpm_service("SERVICE1");
tmp_service("SERVICE2");
tx_commit();

Server:
SERVICE1()
{
<get service specific data>
EXEC SQL UPDATE...;
<return service status back to the client>}
SERVICE2()
{
<get service specific data>
EXEC SQL UPDATE...;
<return service status back to the client>
}
```

Changing the Existing Application into TPM Application

You should follow the process below to change the existing application (Precompiler or ODBCCLI) into TPM (Transaction Performance Mornitoring) application with using ALTIBASE XA library.

1. You should convert application into 'service' with framework.

   Framework means that application client requests 'service' from its server. Certain TPM can make request for using tx_open and tx_close. Certain TPM logs on or off application server implicitly.

2. You should chanage general connect statement. For example, you should change EXEC SQL CONNECT statement into tx_open() in precompiler, and SQLDriverConnect statement into tx_open() and SQLCONNECT in ODBCCLI..Refer to the process performing ODBC/XA for details.

3. You should change disconnect statement. You should change EXEC SQL DISCONNECT statement into tx_close() in precompiler, and SQLDisconnect statement into tx_close() in ODBCCLI.

4. You should change commit and rollback statements. For example, you should change EXEC SQL COMMIT into tx_commit, and EXEC SQL ROLLBACK into tx_rollback in precompiler. You should change SQLEndTran into tx_commiit or tx_rollback in OBDCCLI. They start to execute transaction by calling tx_begin().

5. Application should reset fetched transaction after its end.

   You should block cursor and clear the resource with CLOSE RELEASE statement after fetching data by using cursor before ending transaction.

ALTIBASE Statement	TPM Functions
CONNECT	tx_open
Starting to execute transaction implicitly	tx_begin
SQL	Executing SQL in service
COMMIT	tx_commit
ROLLBACK	tx_rollback
DISCONNECT	tx_close
SET TRANSACTION READ ONLY	Not allowed

XA Restriction

Several restrictions are as follows when you use XA.

- SQL Use Restriction
- Transaction Branch
- Association Migration
- Asynchronixation Call
- Dynamic Registeration

SQL Use Restriction

Rollback and Commit

XA application should not control global transaction with certain statement of ALTIBASE because TM manages global transaction.

You should use tx_commit or tx_rollback to shut down global transaction. You can't use EXEC SQL ROLLBACK statement or EXEC SQL COMMIT statement in precompiler. ODBCCLI should not make a commit or rollback decision with SQLEndTran.

DDL

ALTIBASE XA can't use DDL because DDL SQL statements make a commit decision internally.

Session Property

You can't change autocommit property. You can execute global transaction in autocommit off mode, and can't change autocommit property on this case.

Set Trasaction

You can't use SQL statements like SET TRANSACTION READ ONLY | READ WRITE | ISOLATION LEVEL ...

Connection or Disconnection with EXEC SQL Statements

You can't use EXEC SQL CONNECT and EXEC SQL DISCONNECT statements for connection or disconnection in precompiler.

Transaction Branch

Multiple application threads participate in executing one global transaction, and these threads have tightly-coupled or loosely-coupled relationshilps.

Tightly-coupled relationship denotes thread sharing resource.

These threads are processed as one object. RM enables transaction branch not to reach a resource deadlock in tightly-coupled thread. However, loosely-coupled thread doesn't guarantee this. RM regards transaction branches including loosely-coupled threads as different global trnasactions each other.

Relationship between XID and Thread

If TM creates new value of branch qualifier in XID, this thread has loosely-coupled relation with other threads in same branch.

RM executes this thread as global transaction. And if TM reuses branch qualifier of XID, this thread has tightly-coupled relation with other threads sharing this branch.

RN regards these tightly-coupled threads as one object, and guarantees that tightly-coupled threads are not in resource deadlock each other.

Association Migration

ALTIBASE doesn't support association migration (TM restarts to execute suspended branch associated with other branches.)

Async Call

ALTIBASE doesn't support async XA call.

Dynamic Registration

ALTIBASE server doesn't support dynamic registration but static registeration. DYnamic registeration denotes that RM registers global transaction in TM before starting to execute global transaction.

RM should know when transaction starts to be excuted by calling xa_start for static registration.

Server Shutdown

If prepared transaction exists during shutdown abort or abnormal shutdown, recovery is applied to this upon startup. And then you can execute this with xa_recover.

If prepared transaction exists during immediate shutdown or normal shutdown, ALTIBASE shuts down server by aborting a transaction and then recovery is applied to prepared transaction. In this event, prepared transaction is simply left in the state that it was in. Otherwise, ALTIBASE shuts down server normally without recovery upon startup.

JDBC Distributed Transaction

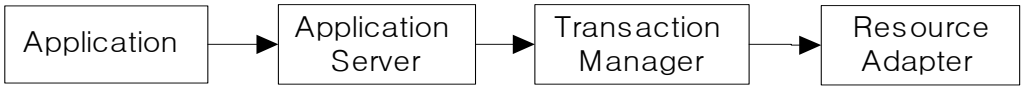
ALTIBASE JDBC to support distributed transaction follows connection pooling and OpenXA standard in JDBC 2.0 extension API.

All classes, which includes options for distributed transaction and follows XA standard, are provided as Altibase.jdbc.driver.

JTA(Java Transaction API) and Application Server

Application processes distributed transaction through its server as the following figure.

Figure 6-3 Distributed Transaction Process



Application server supports XAConnection to be connected to resource.

Application connects to server and executes queries with obtaining connection. And application server manages transaction with TM (Transaction Manager). TM can access to resource with resource adapter, which DBMS vendor provides.

If resource adapter is DBMS, JDBC driver package can be provided.

Resource adapter has 4 kinds of classes such as ResourceFactory, Transactional Resource(XAConnection) and Connection, XAResource.

ResourceFactory creates XAConnection such as XADataSourc of JDBC Spec. Application server obtains XAConnection (connection to DBMS) from XADataSource. And XAConnection obtains connection(java.sql.Connection) instance for application and XAResource instace for TM.

XA Component

This section discusses the XA components - standard XA interface specified in the JDBC 2.0 Optional package, and the ALTIBASE classes that implement them.

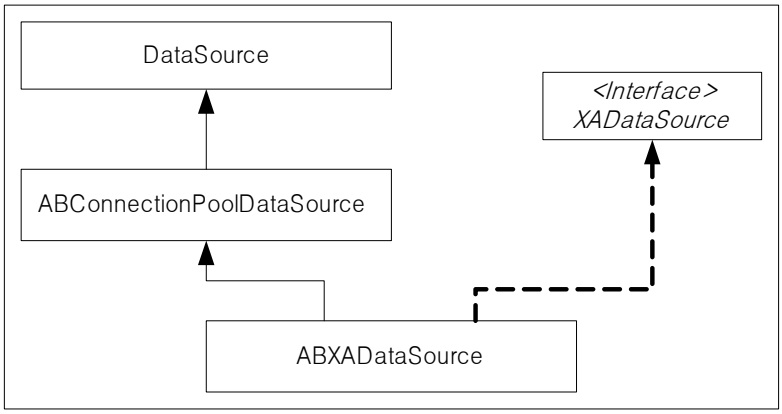
XADataSource Interface

The javax.sql.XADataSource interface outlines factories for XA Connection. The getXAConnection method returns an XA Connection instance.

```
public interface XADataSource
{
 XAConnection getXAConnection() throws SQLException;
 XAConnection getXAConnection(String user, String password)
 throws SQLException;
 ...
}
```

Altibase.jdbc.driver.ABXADataSource implements XADataSource interface, located in JDBC driver which ALTBASE provides, and also extends the Altibase.jdbc.driver.ABConnectionPoolDataSource (which extends the Altibase.jdbc.driver.DataSource), so includes all the connection properties described in DataSource and ABConnectionPoolDataSource.

Figure 6-4 ABXADataSource Class



ABXADataSource class getXAConnection methods return the implementation of XAConnection instances in ABPooledConnection class. You can register XA data source in JNDI.

XAConnection Interface

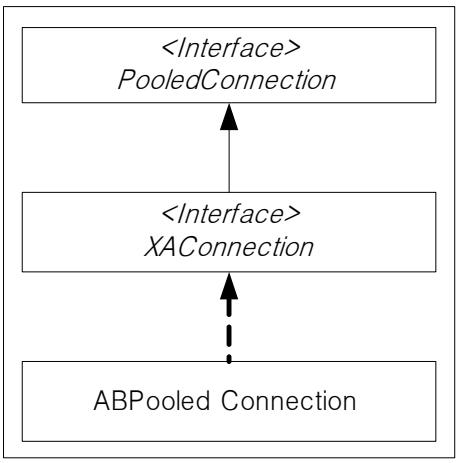
XAConnection interface is the subinterface of PooledConnection, and also includes getConnection, close, addConnectionEventListener and removeConnectionEventListener methods.

```
public interface XAConnection extends PooledConnection
{
javax.jta.xa.XAResource getXAResource() throws SQLException;
 …
}
```

XAConnection instance encapsulates a physical connection to a database, and also has the facility to produce the XAResource that will correspond to it for use in coordinating the distributed transaction. An XAConnection instance is an instance of Altibase.jdbc.driver.ABPooledConnection class in ALTIBASE JDBC driver.

The ABPooledConnection class getXAResource method returns an ABXAResource instance. The getConnection method returns an ABConnection instance.

Figure 6-5 ABPooledConnection Class



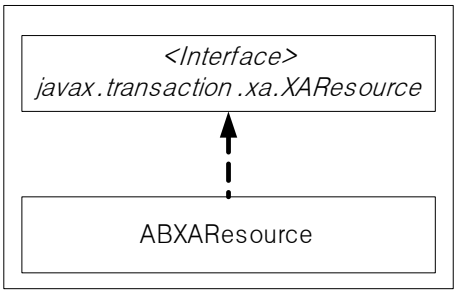
ABConnection instance returned by getConnection method acts as a temporary handle to the physical connection, and runs like common connection before participating in global transaction. If ABConnection instance participates in global transaction, auto-commit is specified as false. After global transaction terminates, auto-commit is restored to its original state prior to occurrence of global transaction.

Each time an XAConnection instance getConnection method is called, it returns a new connection instance. It had been returned by the same XAConnection instance and closes any previous connection instance tha still exists. It is advisable to explicitly close any previous connection instance before opening a new one. Calling the close method of an XAConnection instance closes the physical connection to the database.

XAResource Interface

The TM uses ABXAResource instances to coordinate all the transaction branches. An Altibse.jdbc.driver.ABXAResource instance is an instance of a class that implements the javax.transaction.xa.XAResource interface.

Figure 6-6 ABXAResource Class



The ALTIBASE JDBC driver creates and returns an ABXAResource instance whenever the ABPooledConnection class getXAResource method is called, and associates an ALTIBASE JDBC driver ABXARe-

source instance with an connection instance.

Transaction branch is executed through that connection.

The ABXAResource class has serveral methods to coordinate a transaction branch with the distributed transaction. A TM, receiving ABXAResource instances from a middle-tier component such as application server, typically invokes the following functionalities.

```
void start(Xid xid, int flags)
void end(Xid xid, int flags)
int prepare(Xid xid)
void commit(Xid xid, boolean onePhase)
void rollback(Xid xid)
public void forget(Xid xid)
public Xid[] recover(int flag)
```

Refer to javax.transaction.xa.XAResource of java API Spec for details.

Xid interface

TM creates transaction ID instances and uses them in coordinating the branches of distributed transaction. Each transaction branch is assigned a unique transaction ID, which includes the following informaiton :

```
Format identifier
Global transaction identifier
Branch qualifier
```

ALTIBASE implements javax.transaction.xa.Xid interface with XID class in Altibase.jdbc.driver package.

- ALTIBASE doesn't require the use of Altibase.jdbc.driver.XID for ABXAResource calls. You can use any class that implements javax.transaction.xa.Xid interface.

Error Handling

XA methods throw ABXAException when errors occur. ABXAException class is a subclass of javax.transaction.xa.XAException class.

Setting in Application Server

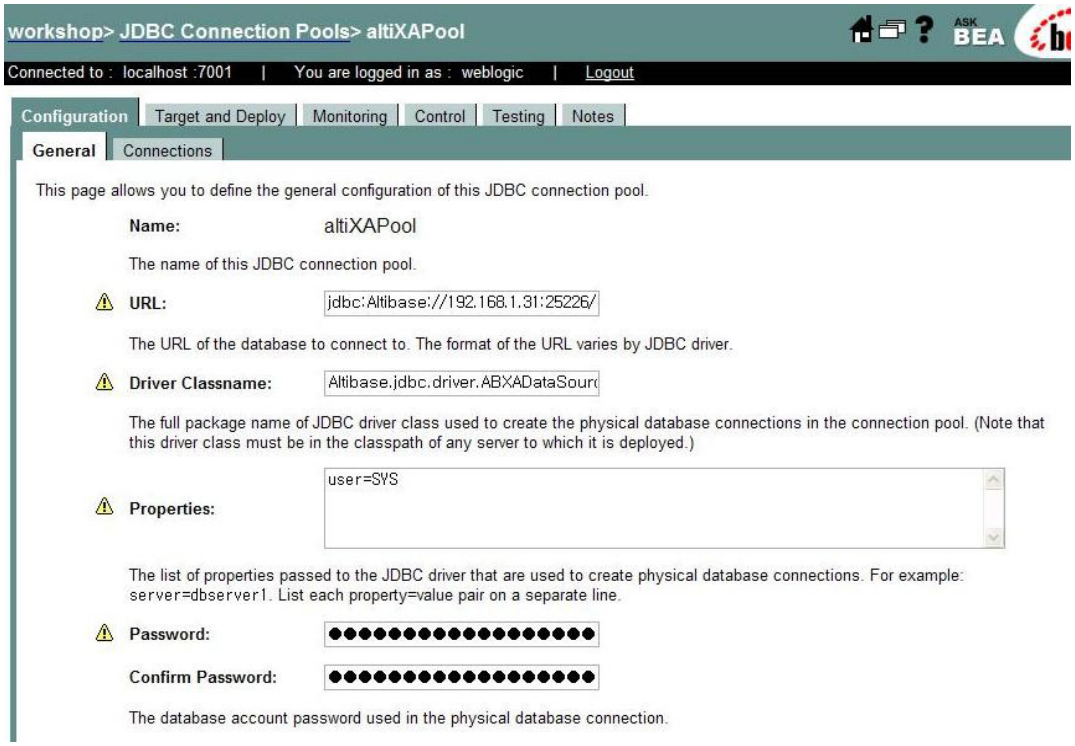
WebLogic

1. You should insert JDBC connection information after choosing Services -> JDBC -> Connection Pools -> Configure a new JDBC Connection Pool in weblogic console. (See [Figure 6-6] JDBC Connection Information)

Table 6-4 Connection Information Difference between NON-XA and XA

	NON-XA	XA
URL	jdbc:Altibase://[ip]:[port]/dbname	jdbc:Altibase://[ip]:[port]/dbname
Driver Classname	Altibase.jdbc.driver.AltibaseDriver	Altibase.jdbc.driver.ABXADataSource
Properties	User=[username]	User=[username]

Figure 6-7 JDBC Connection Information Insertion

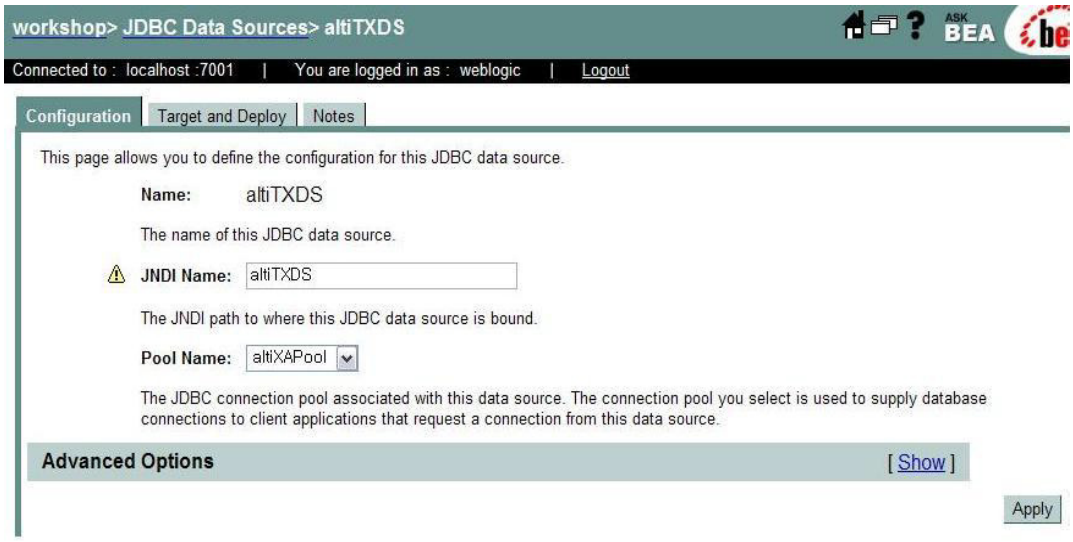


2. You should create DataSource with created Connection Pool, and then choose Services->JDBC->Data Sources->Configure a new JDBC Data Source.

   You should insert Name and JNDI Name, and then check "Honor Global Transaction". You should insert the name of pool created prior to PoolName in the following page. (weblogic 8.1) (See [Figure 6-7] Data Source Creation)

   - New DataSource is created in Services->JDBC->XA Data Sources for previous version of weblogic8.1.

Figure 6-8 Data Source Creation



Weblogic Application Example

```
// step 1. JNDI Lookup and get UserTransaction Object
Context ctx = null;
Hashtable env = new Hashtable();

// Parameter for weblogic
env.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFac-
tory");
env.put(Context.PROVIDER_URL,"t3://localhost:7001");
env.put(Context.SECURITY_PRINCIPAL,"weblogic");
env.put(Context.SECURITY_CREDENTIALS,"weblogic");

ctx = new InitialContext(env);
System.out.println("Context Created :"+ctx);

// step 2. get User Transaction Object
UserTransaction tx = (UserTransaction)ctx.lookup("javax.transaction.User-
Transaction");

// step 3 start Transaction
System.out.println("Start Transaction :"+tx);
tx.begin();

try{
// step 4. doing query
// step 4-1. get Datasource
DataSource xads1 = (DataSource)ctx.lookup("altiTXDS");
```

JEUS

You should set basic setup to create JDBC data source in JEUS.

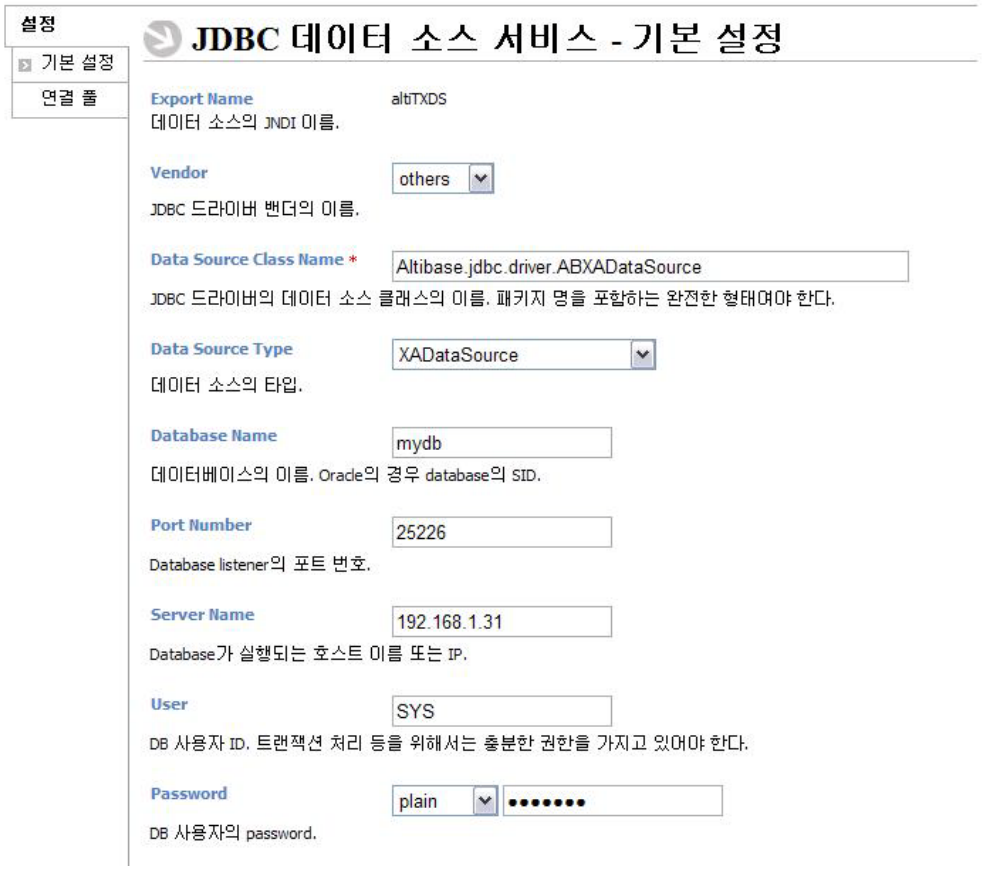
1. You should choose JEUS manager resource, and then select new JDBC data source creation.

2. You should insert the following information if window is displayed for basic setup.

```
DBMS : Other
Available Data Source :
Data Source Class Name: Altibase.jdbc.driver.ABXADataSource
Data Source Type : XADataSource
```

3. You should insert values for Database Name, Port Number, Server Name User and Password.

Figure 6-9 Data Source Setting in JEUS



JEUS Application Example

```
// step 1. JNDI Lookup and get UserTransaction Object
Context ctx = null;
Hashtable env = new Hashtable();

// Parameter for weblogic
env.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JNSContextFactory");
env.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
env.put(Context.PROVIDER_URL, "127.0.0.1");
env.put(Context.SECURITY_PRINCIPAL,"jeus");
env.put(Context.SECURITY_CREDENTIALS,"jeus");

ctx = new InitialContext(env);
System.out.println("Context Created :"+ctx);

// step 2. get User Transaction Object
```

```
UserTransaction tx = (UserTransaction)ctx.lookup("java:comp/UserTransac-
tion");

// step 3 start Transaction
System.out.println("Start Transaction :"+tx);
tx.begin();

try{
// step 4. doing query
// step 4-1. get Datasource
DataSource xads1 = (DataSource)ctx.lookup("altiTXDS");
```

Example

You can know how to implement distributed transaction by using ALTIABSE XA as the following example.

```
Start transaction branch #1.
Start transaction branch #2.
Execute DML operations on branch #1.
Execute DML operations on branch #2.
End transaction branch #1.
End transaction branch #2.
Prepare branch #1.
Prepare branch #2.
Commit branch #1.
Commit branch #2.

import java.sql.*;
import javax.sql.*;
import Altibase.jdbc.driver.*;
import javax.transaction.xa.*;

class XA4
{
 public static void main (String args [])
 throws SQLException
 {
 try
 {
 String URL1 = "jdbc:Altibase://localhost:25226/mydb";
 // You can put a database name after the @ sign in the connection URL.
 String URL2 = "jdbc:Altibase://localhost:25226/mydb";

 // Create first DataSource and get connection
 Altibase.jdbc.driver.DataSource ads1 = new Altibase.jdbc.driver.Data-
Source();
 ads1.setUrl(URL1);
 ads1.setUser("SYS");
 ads1.setPassword("MANAGER");
 Connection conna = ads1.getConnection();

 // Create second DataSource and get connection
 Altibase.jdbc.driver.DataSource ads2 = new Altibase.jdbc.driver.Data-
Source();
 ads2.setUrl(URL2);
 ads2.setUser("SYS");
 ads2.setPassword("MANAGER");
 Connection connb = ads2.getConnection();

 // Prepare a statement to create the table
 Statement stmta = conna.createStatement ();
```

```
// Prepare a statement to create the table
Statement stmtb = connb.createStatement ();
try
{
// Drop the test table
stmta.execute ("drop table my_table");
}
catch (SQLException e)
{
// Ignore an error here
}

try
{
// Create a test table
stmta.execute ("create table my_table (col1 int)");
}
catch (SQLException e)
{
// Ignore an error here too
}

try
{
// Drop the test table
stmtb.execute ("drop table my_tab");
}
catch (SQLException e)
{
// Ignore an error here
}

try
{
// Create a test table
stmtb.execute ("create table my_tab (col1 char(30))");
}
catch (SQLException e)
{
// Ignore an error here too
}

// Create XADataSource instances and set properties.
ABXADataSource axds1 = new ABXADataSource();
axds1.setUrl("jdbc:Altibase://localhost:25226/mydb");
axds1.setUser("SYS");
axds1.setPassword("MANAGER");

ABXADataSource axds2 = new ABXADataSource();

axds2.setUrl("jdbc:Altibase://localhost:25226/mydb");
axds2.setUser("SYS");
axds2.setPassword("MANAGER");

// Get XA connections to the underlying data sources
XAConnection pc1 = axds1.getXAConnection();
XAConnection pc2 = axds2.getXAConnection();
// Get the physical connections
Connection conn1 = pc1.getConnection();
Connection conn2 = pc2.getConnection();

// Get the XA resources
XAResource axar1 = pc1.getXAResource();
XAResource axar2 = pc2.getXAResource();
```

```
// Create the Xids With the Same Global Ids
Xid xid1 = createXid(1);
Xid xid2 = createXid(2);

// Start the Resources
axar1.start (xid1, XAResource.TMNOFLAGS);
axar2.start (xid2, XAResource.TMNOFLAGS);

// Execute SQL operations with conn1 and conn2
doSomeWork1 (conn1);
doSomeWork2 (conn2);

// END both the branches -- IMPORTANT
axar1.end(xid1, XAResource.TMSUCCESS);
axar2.end(xid2, XAResource.TMSUCCESS);

// Prepare the RMs
int prp1 = axar1.prepare (xid1);
int prp2 = axar2.prepare (xid2);
System.out.println("Return value of prepare 1 is " + prp1);
System.out.println("Return value of prepare 2 is " + prp2);
boolean do_commit = true;

if (!((prp1 == XAResource.XA_OK) || (prp1 == XAResource.XA_RDONLY)))
do_commit = false;
if (!((prp2 == XAResource.XA_OK) || (prp2 == XAResource.XA_RDONLY)))
do_commit = false;
System.out.println("do_commit is " + do_commit);
System.out.println("Is axar1 same as axar2 ? " + axar1.isSameRM(axar2));

if (prp1 == XAResource.XA_OK)
if (do_commit)
axar1.commit (xid1, false);
else
axar1.rollback (xid1);
if (prp2 == XAResource.XA_OK)
if (do_commit)
axar2.commit (xid2, false);
else
axar2.rollback (xid2);

// Close connections
conn1.close();
conn1 = null;
conn2.close();
conn2 = null;

pc1.close();
pc1 = null;
pc2.close();
pc2 = null;

ResultSet rset = stmta.executeQuery ("select col1 from my_table");
while (rset.next())
System.out.println("Col1 is " + rset.getInt(1));
rset.close();
rset = null;
rset = stmtb.executeQuery ("select col1 from my_tab");
while (rset.next())
System.out.println("Col1 is " + rset.getString(1));
rset.close();
rset = null;

stmta.close();
```

```
stmta = null;
stmtb.close();
stmtb = null;

conna.close();
conna = null;
connb.close();
connb = null;
} catch (SQLException sqe)
{
sqe.printStackTrace();
} catch (XAException xae)
{
System.out.println("XA Error is " + xae.getMessage());
}
}

static Xid createXid(int bids)
throws XAException
{
byte[] gid = new byte[1]; gid[0] = (byte)9;
byte[] bid = new byte[1]; bid[0] = (byte)bids;
byte[] gtrid = new byte[4];
byte[] bqual = new byte[4];
System.arraycopy(gid,0,gtrid,0,1);
System.arraycopy(bid,0,bqual,0,1);
Xid xid = new XID(0x1234,gtrid,bqual);
return xid;
}

private static void doSomeWork1 (Connection conn)
throws SQLException
{
String sql ;
Statement stmt = conn.createStatement();
sql = "insert into my_table values(1)";
stmt.executeUpdate(sql);
stmt.close();
}

private static void doSomeWork2 (Connection conn)
throws SQLException
{
String sql ;
Statement stmt = conn.createStatement();
sql = "insert into my_tab values('test')";
stmt.executeUpdate(sql);
stmt.close();
}
}
```

How to Solve Problems of Application Using XA

This section has focus on how to search information of system failure when problem occurs.

XA Tracking Information Check

ALTIBASE XA library records errors and tracking informaiton in trace file. You can check information such as error code and message if opening ths file.

For example, if xa_open fails, you can know such reasons that open string has errors or TP Manager doesn't search ALTIBASE server or fails to log on by checking tracking information.

Trace File Name and Location

```
altibase_xaXA_NAMEdate.log
```

- XA_NAME : This sets XA_NAME=value in open string. Otherwize, NULL is specified.
- data : This is the date specified in trace file(YYYYMMDD).

If environment variable is specified in $ALTIBASE_HOME, this is created in $ALTIBASE_HOME/trc. Otherwise, this is created in current directory.

Example

```
104744.19381.1:
ulxXaOpen     : XAER_RMERR : [ERR-4102E] Invalid password
```

'104744' is logging time(HHMISS), '19381' is Process ID(PID) and '1' is resource manager ID.

ulxXaOpen is moduel name, XAER_RMERR is XA spec. error, [ERR-4102E] is error code returned by ALTIBASE service, and 'invalid password' is error code returned by ALTIBASE server.

In-doubt Transaction Process

TM recognizes problem and provides functionality to recover in-doubt transaction automatically if in-doubt or pending transaction occurs. RM awaits until it is recovered and receive a commit decision automatically while locking prepared resource.

DBA should process transaction properly if other transaction requires data locked by in-doubt transaction or the problem hasn't been correncted. ALTIBASE provides performance view. This enables you to search for the state of in-dount transaction for processing in-doubt transaction.

Refer to Administrator's Manual for details about V$DBA_2PC_PENDING.

DBA can compulsorily commit or rollback transaction as follows to process it properly.

```
COMMIT FORCE global_tx_id;
ROLLBACK FORCE global_tx_id;
```

Example

You should check in-doubt trasaction and commit transaction randomly.

```
iSQL> select * From v$dba_2pc_pending;
LOCAL_TRAN_ID GLOBAL_TX_ID
------------------------------------------------------
9280 69.FAEDFAED.00000001
21315 69.FAEDFAED.00000002
2 rows selected.
iSQL> commit force '69.FAEDFAED.00000002';
Commit force success.
```

Heuristic Transaction Check

You can check information of heuristic transaction if it occurs.

Heuristic transaction denotes that RM itself makes a commit or rollback decision if in-doubt transaction can't receive command such as commit or rollback cause of some reasons.

If in-doubt transaction is forced to commit, this transaction becomes committed heuristically. And this information is inserted into SYS_XA_HEURISTIC_TRANS_.

If you want to delete this information, you should call xa_forget after executing xa_recovery, or run remove_xid().

Example

If DBA commits in-doubt transaction, changed information is inserted into SYS_XA_HEURISTIC_TRANS_.

```
iSQL> select * From v$dba_2pc_pending;
V$DBA_2PC_PENDING.LOCAL_TRAN_ID
V$DBA_2PC_PENDING.GLOBAL_TX_ID
----------------------------------------
100421
69.FAEDFAED.00000001
1 row selected.
iSQL> commit force '69.FAEDFAED.00000001';
Commit force success.
iSQL> select * from system_.sys_xa_heuristic_trans_;
SYS_XA_HEURISTIC_TRANS_.FORMAT_ID
SYS_XA_HEURISTIC_TRANS_.GLOBAL_TX_ID
SYS_XA_HEURISTIC_TRANS_.BRANCH_QUALIFIER
SYS_XA_HEURISTIC_TRANS_.STATUS
SYS_XA_HEURISTIC_TRANS_.OCCUR_TIME
---------------------------------------
69
FAEDFAED
00000001
1
2008/08/29 10:09:53
1 row selected.
ISQL> exec remove_xid('69.FAEDFAED.00000001');
Execute success.
iSQL> select * from system_.sys_xa_heuristic_trans_;
SYS_XA_HEURISTIC_TRANS_.FORMAT_ID
-----------------------------------
SYS_XA_HEURISTIC_TRANS_.GLOBAL_TX_ID
```

```
--------------------------------------------------------------------------------
------------------------------
SYS_XA_HEURISTIC_TRANS_.BRANCH_QUALIFIER
--------------------------------------------------------------------------------
------------------------------
SYS_XA_HEURISTIC_TRANS_.STATUS SYS_XA_HEURISTIC_TRANS_.OCCUR_TIME
-------------------------------------------------------------------
No rows selected.
```

Index