ALTIBASE® HDB™ Application Development

Application Program Interface User's Manual

Release 6.5.1

May 28, 2015

ALTIBASE

ALTIBASE HDB Application Development Application Program Interface User's Manual
Release 6.5.1
Copyright © 2001~2015 Altibase Corporation. All rights reserved.

This manual contains proprietary information of Altibase® Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright patent and other intellectual property law. Reverse engineering of the software is prohibited.

All trademarks, registered or otherwise, are the property of their respective owners

Altibase Corporation
10F, Daerung PostTower II, 182-13,
Guro-dong Guro-gu Seoul, 152-847, South Korea
Telephone: +82-2-2082-1000 Fax: 82-2-2082-1099
Homepage: http://www.altibase.com

Contents

Preface ..... **vi**
About This Manual vii
Audience vii
Software Environment vii
Organization vii
Documentation Conventions viii
Related Reading x
Online Manuals x
Altibase Welcomes Your Comments xi
1. JDBC **1**
1.1 Installation 2
1.1.1 Checking the JDBC Driver Version 2
1.1.2 JAVA Application 2
1.1.3 Tomcat 2
1.1.4 WebLogic 2
1.1.5 Jeus 2
1.2 Using JDBC to Connect to ALTIBASE HDB 3
1.2.1 Connection Sequence 3
1.3 Internet Protocol Version 6 (IPv6) Support 6
1.3.1 Using IPv6 Addresses 6
1.3.2 Notes 6
1.4 Handling National Character Sets 8
1.4.1 Character Set Conversion 8
1.4.2 Settings in JSP 8
1.5 Using National Character Sets in SQL with JDBC 9
1.5.1 Querying and Changing Data 9
1.5.2 Using Character-String Constants 9
1.6 Setting up a Connection Pool 10
1.6.1 Setting up a WAS (Web Application Server) 10
1.6.2 Setting up the JDBC Driver 12
1.7 JDBC 2.1 Core API and JDBC 2.0 Optional Package API 15
1.7.1 JDBC 3.0 API 15
1.7.2 java.sql Package Support Status 15
1.7.3 javax.sql package Support Status 35
1.7.4 javax.transaction.xa package Support Status 37
1.7.5 Data Size Consideration 37
1.8 JDBC Connection Failover 38
2. The PHP Interface **39**
2.1 About the PHP Module of ALTIBASE HDB 40
2.2 Installing the ODBC Manager for Integration with PHP 41
2.2.1 The ODBC Manager in Unix and Linux 41
2.2.2 The ODBC Manager in Windows 42
2.3 PHP Functions for ODBC Connectivity 43
2.3.1 Sample Test 43
3. PERL DBD DBI **45**
3.1 Overview of the Perl DBD and DBI 46
3.2 Perl Package Installation 47
3.2.1 The Perl Package Installation Procedure 47
3.3 Installing the ALTIBASE HDB DBD 48
3.3.1 The ALTIBASE HDB PERL DBD Installation Procedure 48
4. .NET Data Provider **51**
4.1 An Overview of the .NET Data Provider 52
4.1.1 Overview 52
4.1.2 Requirements 52
4.1.3 Considerations 52

4.2 Using the ALTIBASE HDB .NET Data Provider 54
4.2.1 Compiling .NET Applications 54
4.2.2 Array Binding 56
4.2.3 Declaring the ALTIBASE HDB .NET Data Provider 58
4.2.4 Processing Transactions 58
4.2.5 Schema 59
4.2.6 ALTIBASE HDB .NET Data Provider Classes 60
4.2.7 ALTIBASE HDB .NET Data Provider Data Types 61
4.2.8 Considerations 62
4.3 .NET Data Provider Interface 63
4.3.1 Unsupported Interfaces 63
4.4 .NET Data Provider Example 65
4.4.1 Connection String 65
4.4.2 Simple example with DDL and DML 65
4.4.3 Bulkcopying 66
5. OLE DB 69
5.1 Overview of OLE DB 70
5.2 Installation 71
5.2.1 Installing and Uninstalling OLE DB 71
5.2.2 Verifying the OLE DB Installation 71
5.2.3 Creating a Connection String 74
5.2.4 ALTIBASE HDB and OLE DB Data Types 76
5.3 Properties 78
5.3.1 DBPROPSET_DATASOURCEINFO 78
5.3.2 DBPROPSET_DBINIT 79
5.3.3 DBPROPSET_OGIS_SPATIAL_OPS 79
5.3.4 DBPROPSET_ROWSET 80
5.3.5 DBPROPSET_SESSION 81
5.4 The OLE DB Interfaces 82
5.4.1 The Data Source Object 82
5.4.2 The Session Object 82
5.4.3 The Command Object 82
5.4.4 The Multiple Results Object 83
5.4.5 The Rowset Object 83
5.4.6 The Row Object 83
5.4.7 The Error Object 83
5.4.8 The Custom Error Object 84
5.5 Examples 85
5.5.1 Connection String 85
5.5.2 ADO Visual Basic Example 85
5.5.3 ADO.NET C# Example 85
5.5.4 ADO.NET C++ Code Snippet for Calling Stored Procedure 86
6. XA Interface 89
6.1 XA Interface 90
6.1.1 XA Glossary 90
6.1.2 XA Structure 91
6.1.3 XA and 2PC (Two-Phase Commit) 92
6.1.4 xa_switch_t Structure 92
6.1.5 The XA Library 93
6.2 The XA Interface 94
6.2.1 XA Functions 94
6.3 Using XA 99
6.3.1 Executing ODBC/XA 99
6.3.2 Executing APRE/XA 100
6.3.3 Executing JDBC/XA 101
6.3.4 XA Transaction Control 102
6.3.5 Changing an Existing Application into a TPM Application 104
6.4 Limitations when using XA 106
6.4.1 Limitations on Use of SQL 106

6.4.2 Limitations related to Transaction Branches 107
6.4.3 No Support for Association Migration 107
6.4.4 No Support for Asynchronous Calls 107
6.4.5 No Support for Dynamic Registration 107
6.4.6 Server Shutdown 107
6.5 JDBC Distributed Transactions 109
6.5.1 JTA (Java Transaction API) and Application Server 109
6.5.2 XA Components 109
6.5.3 Error Handling 112
6.5.4 Making XA Settings in Application Servers 112
6.5.5 Example 116
6.6 How to Solve Application Problems Using XA 120
6.6.1 Checking XA Tracking Information 120
6.6.2 Processing In-doubt Transactions 120
6.6.3 Checking Heuristically Completed Transactions 121
7. The iLoader API 123
7.1 Overview of the iLoader API 124
7.2 Using the iLoader API 125
7.2.1 Header Files 125
7.2.2 Libraries 125
7.2.3 Samples 125
7.3 iLoader API Data Structures 126
7.3.1 The iLoader Handle 126
7.3.2 Error Structure 126
7.3.3 Log Structure 126
7.3.4 Option Structure 128
7.3.5 iLoader API Enumerators 130
7.4 The iLoader API 131
7.4.1 altibase_iloader_datain 131
7.4.2 altibase_iloader_dataout 133
7.4.3 altibase_iloader_final 135
7.4.4 altibase_iloader_formout 136
7.4.5 altibase_iloader_init 138
7.4.6 altibase_iloader_options_init 140
7.4.7 CallbackFunctionName 140
8. The CheckServer API 145
8.1 Overview of the CheckServer API 146
8.1.1 Restrictions 146
8.2 Using the CheckServer API 147
8.2.1 Header File 147
8.2.2 The CheckServer Libraries 147
8.2.3 Samples 147
8.3 CheckServer API Data Structure 148
8.3.1 The CheckServer Handle 148
8.4 The CheckServer API 149
8.4.1 altibase_check_server 149
8.4.2 altibase_check_server_final 150
8.4.3 altibase_check_server_init 151
8.4.4 altibase_check_server_cancel 152

Preface

About This Manual

This manual explains how to use the ALTIBASE HDB API.

Audience

This manual has been prepared for the following users of ALTIBASE HDB:

- Database administrators
- Application developers
- Programmers

It is recommended that those reading this manual possess the following background knowledge:

- Basic knowledge in the use of computers, operating systems, and operating system utilities
- Experience in using relational databases and an understanding of database concepts
- Computer programming experience

Software Environment

This manual has been prepared assuming that ALTIBASE HDB 6 will be used as the database server.

Organization

This manual is organized as follows:

- Chapter2: The PHP Interface

  This chapter explains how to integrate PHP pages with ALTIBASE HDB using ODBC functionality of PHP.

- Chapter3: PERL DBD DBI

  This chapter explains how to use the Perl DBD (Database Driver) and DBI (Database Interface) with ALTIBASE HDB. It describes in detail how to install Perl and the ALTIBASE HDB Perl DBD and how to check the status of the ALTIBASE HDB Perl DBD.

- Chapter4: .NET Data Provider

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

- Chapter6: XA Interface

  This chapter introduces the data structures and functions that are needed to use the XA functions supported by ALTIBASE HDB and provides basic procedures for using ODBC, APRE and JDBC in an XA environment..

- Chapter7: The iLoader API

This chapter introduces the ALTIBASE HDB iLoader API, which is an application programming interface that lets you create applications that use function calls to download data from, or upload data to, an Altibase database server.

- Chapter8: The CheckServer API

  This chapter introduces the ALTIBASE HDB CheckServer API, which is an application programming interface for creating applications that use function calls to monitor whether the ALTIBASE HDB server has terminated abnormally.

Documentation Conventions

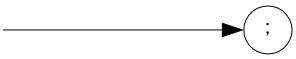
This section describes the conventions used in this manual. Understanding these conventions will make it easier to find information in this manual and in the other manuals in the series.

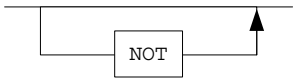
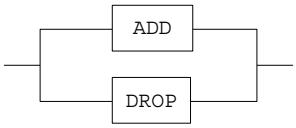
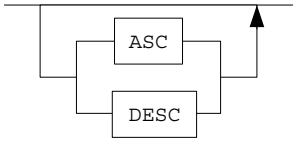
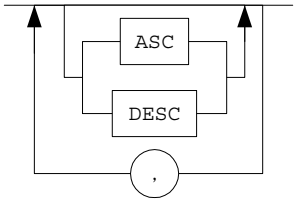
There are two sets of conventions:

- Syntax Diagram Conventions
- Sample Code Conventions

Syntax Diagram Conventions

In this manual, the syntax of commands is described using diagrams composed of the following elements:

Element	Description
	Indicates the start of a command. If a syntactic element starts with an arrow, it is not a complete command.
	Indicates that the command continues to the next line. If a syntactic element ends with this symbol, it is not a complete command.
	Indicates that the command continues from the previous line. If a syntactic element starts with this symbol, it is not a complete command.
	Indicates the end of a statement.
	Indicates a mandatory element.

Element	Description
	Indicates an optional element.
	Indicates a mandatory element comprised of options. One, and only one, option must be specified.
	Indicates an optional element comprised of options.
	Indicates an optional element in which multiple elements may be specified. A comma must precede all but the first element.

Sample Code Conventions

The code examples explain SQL statements, stored procedures, iSQL statements, and other command line syntax.

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

Convention	Meaning	Example
[]	Indicates an optional item.	`VARCHAR [(size)] [[FIXED |] VARIABLE]`
{}	Indicates a mandatory field for which one or more items must be selected.	`{ ENABLE | DISABLE | COMPILE }`
|	A delimiter between optional or mandatory arguments.	`{ ENABLE | DISABLE | COMPILE }` `[ ENABLE | DISABLE | COMPILE ]`

Convention	Meaning	Example

.
.
.
	Indicates that the previous argument is repeated, or that sample code has been omitted.	`iSQL> select e_lastname from employees;`
		`E_LASTNAME`
		`------------------------`
		`Moon`
		`Davenport`
		`Kobain`
		`.`
		`.`
		`.`
		`20 rows selected.`

Other symbols	Symbols other than those shown above are part of the actual code.	`EXEC :p1 := 1;` `acc NUMBER(11,2);`
Italics	Statement elements in italics indicate variables and special values specified by the user.	`SELECT * FROM table_name;` `CONNECT` *`userID/password`*`;`
Lower Case Letters	Indicate program elements set by the user, such as table names, column names, file names, etc.	`SELECT e_lastname FROM employees;`
Upper Case Letters	Keywords and all elements provided by the system appear in upper case.	`DESC SYSTEM_.SYS_INDICES_;`

Related Reading

For additional technical information, please refer to the following manuals:

- ALTIBASE HDB Installation Guide
- ALTIBASE HDB Administrator's Manual
- ALTIBASE HDB Replication Manual
- ALTIBASE HDB Precompiler User's Manual
- ALTIBASE HDB ODBC Reference
- ALTIBASE HDB iSQL User's Manual
- ALTIBASE HDB Utilities Manual
- ALTIBASE HDB Error Message Reference

Online Manuals

Online versions of our manuals (PDF or HTML) are available from Altibase's Customer Support site (http://support.altibase.com/).

Altibase Welcomes Your Comments

Please feel free to send us your comments and suggestions regarding this manual. Your comments and suggestions are important to us, and 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 that you are using
- Any comments that you have about the manual
- Your full name, address, and phone number

For immediate assistance with technical issues, please contact Altibase's Customer Support site (http://support.altibase.com/).

We always appreciate your comments and suggestions.

1 JDBC

This chapter describes how to install the JDBC driver of ALTIBASE HDB, connect to ALTIBASE HDB using JDBC, handle national character sets, configure a connection pool, and enable failover in a JDBC environment.

1.1 Installation

1. Download the Client package of ALTIBASE HDB from the Altibase Customer Support (http://support.altibase.com) and install it.

2. Copy the JDBC driver to the resource directory of each WAS (Web Application Server). If necessary, edit the configuration file for each WAS.

1.1.1 Checking the JDBC Driver Version

The following shows how to check the version of the JDBC and JDK driver that are currently being used.

```
$ java -jar $ALTIBASE_HOME/lib/Altibase.jar
JDBC Driver Info : Altibase Ver = 6.1.1.1 for JavaVM v1.4, CMP:5.6.2, $Revi-
sion: 14502 $ Mar 12 2012 17:45:32
```

1.1.2 JAVA Application

Add the path and the filename “Altibase.jar” to the CLASSPATH environment variable, as shown in the following example:

```
export CLASSPATH=$CLASSPATH:$ALTIBASE_HOME/lib/Altibase.jar
```

1.1.3 Tomcat

Copy the Altibase.jar file to the $TOMCAT_HOME/common/lib directory ($TOMCAT_HOME is the home directory for the Apache Tomcat installation), so that all web applications will be able to use the JDBC driver of ALTIBASE HDB.

1.1.4 WebLogic

Copy the “Altibase.jar” file to the $WL_HOME/lib/ folder ($WL_HOME is the home directory for the Weblogic installation). In the startWebLogic.sh script, add the path to the Altibase.jar file to CLASSPATH.

1.1.5 Jeus

Copy the “Altibase.jar” file to the /$JEUS_HOME/lib/datasource/ folder. ($JEUS_HOME is the home directory for the Jeus installation).

1.2 Using JDBC to Connect to ALTIBASE HDB

1.2.1 Connection Sequence

1. Loading the Driver

   Load the JDBC driver of ALTIBASE HDB.

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

   Here, there is no need to use the DriverManager to create and register an instance of the driver. When the driver is loaded using the call to Class.forName() as shown above, an instance of the driver is created and registered automatically.

   Loading the driver in this way makes it possible to connect to ALTIBASE HDB.

2. Connecting to a Database

   When connecting, use the database URL format that is supported by the JDBC driver of ALTIBASE HDB.

   The Database URL format for JDBC Driver of ALTIBASE HDB when connecting via TCP/IP is:

   ```
   jdbc:Altibase://server_ip:server_port/dbname
   ```

   server_ip can be a host name, an IPv4 address, or an IPv6 address. For more information about IPv6, please refer to Internet Protocol Version 6 (IPv6) Support.

1.2.1.1 Example

The following is a sample JDBC program that illustrates how to connect to an Altibase database. It can be found at $ALTIBASE_HOME/sample/JDBC/SimpleSQL/SimpleSQL.java.

```
import java.util.Properties;
import java.sql.*;

class SimpleSQL
{
    public static void main(String args[]) {

        Properties        sProps   = new Properties();
        Connection        sCon     = null;
        Statement         sStmt    = null;
        PreparedStatement sPreStmt = null;
        ResultSet         sRS;

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

        String sPort     = args[0];
        String sURL      = "jdbc:Altibase://127.0.0.1:" + sPort + "/mydb";
        String sUser     = "SYS";
        String sPassword = "MANAGER";
```

```
sProps.put( "user",     sUser);
sProps.put( "password", sPassword);
// sProps.put( "encoding", sEncoding );

/* Deploy Altibase's JDBC Driver  */
try
{
    Class.forName("Altibase.jdbc.driver.AltibaseDriver");
}
catch ( Exception e )
{
    System.out.println("Can't register Altibase Driver");
    System.out.println( "ERROR MESSAGE : " + e.getMessage() );
    System.exit(-1);
}

/* Initialize environment */
try
{
    sCon = DriverManager.getConnection( sURL, sProps );
    sStmt = sCon.createStatement();
}
catch ( Exception e )
{
    System.out.println( "ERROR MESSAGE : " + e.getMessage() );
    e.printStackTrace();
}

try
{
    sStmt.execute( "DROP TABLE TEST_EMP_TBL" );
}
catch ( SQLException e )
{
}

try
{
   sStmt.execute( "CREATE TABLE TEST_EMP_TBL " +
                  "( EMP_FIRST VARCHAR(20), " +
                  "EMP_LAST VARCHAR(20), " +
                  "EMP_NO INTEGER )" );

    sPreStmt = sCon.prepareStatement( "INSERT INTO TEST_EMP_TBL " +
                                      "VALUES( ?, ?, ? )" );

    sPreStmt.setString( 1, "Susan" );
    sPreStmt.setString( 2, "Davenport" );
    sPreStmt.setInt(    3, 2 );
    sPreStmt.execute();

    sPreStmt.setString( 1, "Ken" );
    sPreStmt.setString( 2, "Kobain" );
    sPreStmt.setInt(    3, 3 );
    sPreStmt.execute();

    sPreStmt.setString( 1, "Aaron" );
    sPreStmt.setString( 2, "Foster" );
    sPreStmt.setInt(    3, 4 );
    sPreStmt.execute();

    sPreStmt.setString( 1, "Farhad" );
    sPreStmt.setString( 2, "Ghorbani" );
    sPreStmt.setInt(    3, 5 );
    sPreStmt.execute();
```

```
            sPreStmt.setString( 1, "Ryu" );
            sPreStmt.setString( 2, "Momoi" );
            sPreStmt.setInt(    3, 6 );
            sPreStmt.execute();

            sRS = sStmt.executeQuery( "SELECT EMP_FIRST, EMP_LAST," +
                                      " EMP_NO FROM TEST_EMP_TBL " );

            /* Fetch all data */
            while( sRS.next() )
            {
                System.out.println( "  EmpName : " + sRS.getString(1) +
                                    " " + sRS.getString(2) );
                System.out.println( "  EmpNO   : " + sRS.getInt(3) );
            }

            /* Finalize process */
            sStmt.close();
            sPreStmt.close();
            sCon.close();
        }
        catch ( SQLException e )
        {
            System.out.println( "ERROR CODE    : " + e.getErrorCode() );
            System.out.println( "ERROR MESSAGE : " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

The sample program is executed as follows:

```
$ javac SimpleSQL.java
$ java SimpleSQL 20300
  EmpName : Susan Davenport
  EmpNO   : 2
  EmpName : Ken Kobain
  EmpNO   : 3
  EmpName : Aaron Foster
  EmpNO   : 4
  EmpName : Farhad Ghorbani
  EmpNO   : 5
  EmpName : Ryu Momoi
  EmpNO   : 6
```

1.3 Internet Protocol Version 6 (IPv6) Support

This release of the ALTIBASE HDB JDBC Driver supports the use of IPv6 addresses in the JDBC URL, as well as the use of host names that resolve to IPv6 addresses.

1.3.1 Using IPv6 Addresses

To specify an IPv6 address as a URL, the address must be enclosed in square brackets ("[]"). For example, when `localhost` (meaning the same computer) is specified using an IP address, brackets are not used to denote the IPv4 address `127.0.0.1`, whereas they are used to denote the IPv6 address `[::1]`. For more information about IPv6 address notation, please refer to the *ALTIBASE HDB Administrator's Manual*.

The PREFER_IPV6 property determines whether to first attempt to resolve a host name to an IPv4 address or an IPv6 address when a host name is given for the *server_ip* property.
If this property is set to TRUE and a host name is given for the *server_ip* property, the client application will first attempt to resolve the host name to an IPv6 address. If this property is omitted, or if it is set to FALSE, the client application will first attempt to resolve the host name to an IPv4 address. That is, the default behavior is to attempt to resolve the host name to an IPv4 address.

If the client application fails to connect using the preferred IP address type, it then attempts to connect using the other IP address type.

1.3.1.1 Example

```
Connection sCon = null;
Properties sProps = new Properties();

sProps.put( "user", "SYS");
sProps.put( "password", "MANAGER");
sProps.put( "PREFER_IPV6", "FALSE");

String sURL = "jdbc:Altibase://localhost:20300/mydb";
Connection sCon = DriverManager.getConnection( sURL, sProps );
```

1.3.2 Notes

java.net.preferIPv4Stack

In order to connect using an IPv6 address, the java.net.preferIPv4Stack property on the client must be set to FALSE. If this property is set to TRUE, the client application will not be able to connect to a database server using an IPv6 address.

```
Ex)
$ java -Djava.net.preferIPv4Stack=false sample [::1]
```

java.net.preferIPv6Addresses

Setting the java.net.preferIPv6Addresses property to TRUE or FALSE on the client has no effect in ALTIBASE HDB JDBC.

Windows

The use of IPv6 addresses with the ALTIBASE HDB JDBC driver, which is implemented using the java NIO library, is not supported on Windows systems due to an NIO channels bug. For more information about this problem, please visit http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6230761.

Note however that the use of IPv6 addresses is supported in JDK 7 and later versions in Windows Vista and later versions.

1.4 Handling National Character Sets

1.4.1 Character Set Conversion

When the JBDC driver of ALTIBASE HDB connects to a database server, the driver obtains information about the character set that is in use on the database server, and then internally converts strings in java programs from UTF16 to the character set being used on the server.

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

1.4.2 Settings in JSP

Place a line of text that indicates the national character set at the top of JSP pages. In the following example, the Korean character set “euc-kr” is specified:

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

The following command is usually executed in order for Korean characters to be properly handled in JSP and Servlet programs:

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

1.5 Using National Character Sets in SQL with JDBC

This section describes how to handle national character types, that is, Unicode types such as the NCHAR and NVARCHAR character types, in JDBC.

1.5.1 Querying and Changing Data

In JDBC, NCHAR and NVARCHAR type data can be queried and updated in the same way that CHAR and VARCHAR type data are. That is, methods such as getString() and setString() can be used normally.

1.5.2 Using Character-String Constants

How to use character-string constants with national character types in SQL statements is explained below.

- When connecting to the server, set the value of the NcharLiteralReplace property to TRUE.
- Place the upper-case character “N” in front of national character type character-string constants in SQL statements.

1.5.2.1 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'AB가나')");
ResultSet sRS = sStmt.executeQuery( "select * from t1 where c1 like N'%가나
%'");
```

1.6 Setting up a Connection Pool

A connection pool can be set up and managed in either of the following two ways:

- Using ABConnectionPoolDataSource: When connection pooling is used in a WAS (Web Application Server), set this class to the JDBC connection pooling configuration option on the WAS.
- Using ABPoolingDataSource: In situations where a WAS is not being used, use this class to manage a connection pool and obtain a PooledConnection from it. These tasks are performed by the JDBC Driver.

This section includes the following topics:

- Setting up a WAS (Web Application Server)
- Setting up the JDBC Driver

1.6.1 Setting up a WAS (Web Application Server)

Some Web Application Servers support the creation of a connection pool using a GUI interface. However, in this section, only the configuration method in which the configuration text files are directly edited will be described. Because different Web Application Servers support the use of different configuration methods, it will be necessary to check which methods are supported in your environment.

This section explains how to create a connection pool using the following Web Application Servers:

- Tomcat 4.x
- WebLogic 6.x
- Jeus 3.x

1.6.1.1 Tomcat 4.x

This explanation assumes that the jakarta commons-dbcp package is being used.

Procedure

1. Install the jakarta commons-dbcp package. For more information on how to install the jakarta commons-dbcp package, please visit http://jakarta.apache.org.
2. When the 'jakarta commons-dbcp' package has been successfully installed, the following resource must 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 number of dB connections in pool -->
 <parameter>
 <name>maxActive</name>
```

```
 <value> 5</value>
 </parameter>
 <!--Maximum number of idle dB connections to retain in pool-->
 <parameter>
 <name>maxIdle</name>
 <value>3</value>
 </parameter>
 <!--Maximum time to wait for a dB connection to become available (unit :
milliseconds)-->
 <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>
```

3. Append the following to 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>
```

1.6.1.2 WebLogic 6.x

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

Procedure

```
<!------ 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 Properties -->
Properties="user=SYS;password=MANAGER;encoding=KSC5601;portNum-
ber=20300;databaseName=mydb;serverName=192.168.1.1"
 Targets="myserver"
 TestTableName="dual"
 URL="jdbc:Altibase://192.168.1.1:20300/mydb"
/>
```

1.6.1.3 Jeus 3.x

Set up the connection pool by editing the <DataSource> element in the "$JEUS_HOME/config/JeusMain.xml" file.

Procedure

```
<!------ JeusMain.xml ----->
<DataSource>
 <Database>
 <Vendor>others</Vendor>
 <ExportName>Alti_XA_DB</ExportName>
<!--Sets Data Source Class -->
<DataSourceClassName>Altibase.jdbc.driver.ABConnectionPoolDataSource</Data-
SourceClassName>
 <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>
 <!--Sets the Idle connection check interval -->
 <MaxIdleTime>300</MaxIdleTime>
 <ResizingPeriod>60</ResizingPeriod>
 <!--Sets the Maximum DB operation timeout -->
 <OperationTimeout>500</OperationTimeout>
 </ConnectionPool>
</Database>
</DataSource>
```

1.6.2 Setting up the JDBC Driver

The following sample program shows how to set up a connection pool using the JDBC driver and the ABPoolingDataSource class. This section also explains how to run the sample program.

1.6.2.1 Example

```
import java.sql.*;
```

```
import Altibase.jdbc.driver.ABPoolingDataSource;
class poolTest
{
 public static void main(String args[])
 {
  int    sMinPoolSize = 3;
  int    sMaxPoolSize = 5;
  int    sConnCount   = 5;
  String sURL = "jdbc:Altibase://localhost:" + args[0] + "/mydb";
  String sSql = "SELECT COUNT(*) FROM V$SESSION WHERE client_type like
'JDBC'";
  ABPoolingDataSource mConnPool = null;
  mConnPool = new ABPoolingDataSource();
  Connection sTestConn[] = new Connection[sConnCount];
  PreparedStatement sStmt = null;
  ResultSet         sRs   = null;
  try
  {
   mConnPool.setUrl(sURL);
   mConnPool.setUser("SYS");
   mConnPool.setPassword("MANAGER");
   mConnPool.setMinPoolSize(sMinPoolSize);
   mConnPool.setMaxPoolSize(sMaxPoolSize);
  } catch ( Exception e )
  {
   e.printStackTrace();
  }
  try
  {
   for( int i=0; i<sConnCount; i++)
   {
    sTestConn[i] = mConnPool.getConnection();
    if ( sTestConn != null )
    {
     System.out.println("SUCCESS : Connection" );
     sStmt = sTestConn[i].prepareStatement(sSql);
     sRs   = sStmt.executeQuery();
     while(sRs.next())
     {
System.out.println("PooledConnection Count : " + sRs.getString(1));
     }
     sStmt.close();
    }
    else
    {
     System.out.println("ERROR : Connection" );
    }
   }
   for(int i=0; i<sConnCount; i++)
   {
    sTestConn[i].close();
   }
  }
  catch ( Exception e )
  {
   e.printStackTrace();
  }
 }
}
```

The sample program is executed as follows:

```
$ javac poolTest.java
$ java poolTest 20300
```

```
SUCCESS : Connection
PooledConnection Count : 3
SUCCESS : Connection
PooledConnection Count : 3
SUCCESS : Connection
PooledConnection Count : 3
SUCCESS : Connection
PooledConnection Count : 4
SUCCESS : Connection
PooledConnection Count : 5
```

1.7 JDBC 2.1 Core API and JDBC 2.0 Optional Package API

This section indicates which functions are currently supported by each version and package of the ALTIBASE HDB JDBC driver (“O”), which functions are to be supported in the future (“Δ”), and which functions are unsupported (“X”).

1.7.1 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

1.7.2 java.sql Package Support Status

ALTIBASE HDB JDBC Driver support status in JDBC 3.0

1.7.2.1 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

1.7.2.2 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()	O
Int	getHoldability()	Δ
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)	O
PreparedStatement	prepareStatement(String sql, int autoGeneratedKeys)	X
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)	Δ
PreparedStatement	prepareStatement(String sql, String[] columnNames)	X
Void	releaseSavepoint(Savepoint savepoint)	O

Return	Function Name	Support
Void	rollback()	O
Void	rollback(Savepoint savepoint)	O
Void	setAutoCommit(boolean autoCommit)	O
Void	setCatalog(String catalog)	X
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

1.7.2.3 DatabaseMetaData

Return Type	Function Name	Support
Boolean	allProceduresAreCallable()	O
Boolean	allTablesAreSelectable()	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()	O
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
Connection	getConnection()	O

Return Type	Function Name	Support
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
int	getMaxCursorNameLength()	O

Return Type	Function Name	Support
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
ResultSet	getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	O

Return Type	Function Name	Support
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
boolean	supportsAlterTableWithAddColumn()	O
boolean	supportsAlterTableWithDropColumn()	O

Return Type	Function Name	Support
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	supportsDataDefinitionAndDataManipulationTransac-tions()	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
boolean	supportsMixedCaseIdentifiers()	O
boolean	supportsMixedCaseQuotedIdentifiers()	O

Return Type	Function Name	Support
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
boolean	supportsSubqueriesInQuantifieds()	O
boolean	supportsTableCorrelationNames()	O

Return Type	Function Name	Support
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

1.7.2.4 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
InputStream	getBinaryStream(int columnIndex)	O
InputStream	getBinaryStream(String columnName)	O
Blob	getBlob(int i)	O

Return	Function Name	Support
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)	O
Clob	getClob(String colName)	O
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
Object	getObject(int columnIndex)	O
Object	getObject(int i, Map map)	X
Object	getObject(String columnName)	O

Return	Function Name	Support
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
boolean	last()	O
void	moveToCurrentRow()	O
void	moveToInsertRow()	X

Return	Function Name	Support
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)	Δ
void	updateAsciiStream(String columnName, InputStream x, int length)	Δ
void	updateBigDecimal(int columnIndex, BigDecimal x)	X
void	updateBigDecimal(String columnName, BigDecimal x)	X
void	updateBinaryStream(int columnIndex, InputStream x, int length)	Δ
void	updateBinaryStream(String columnName, Input-Stream x, int length)	Δ
void	updateBlob(int columnIndex, Blob x)	Δ
void	updateBlob(String columnName, Blob x)	Δ
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
void	updateCharacterStream(int columnIndex, Reader x, int length)	Δ
void	updateCharacterStream(String columnName, Reader reader, int length)	Δ

Return	Function Name	Support
void	updateClob(int columnIndex, Clob x)	Δ
void	updateClob(String columnName, Clob x)	Δ
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
void	updateTimestamp(String columnName, Timestamp x)	X
boolean	wasNull()	O

1.7.2.5 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

1.7.2.6 Statement

Return Type	Function Name	Support
void	addBatch(String sql)	O
void	cancel()	O
void	clearBatch()	O
void	clearWarnings()	O

Return Type	Function Name	Support
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
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

Return Type	Function Name	Support
void	setMaxRows(int max)	O
void	setQueryTimeout(int seconds)	O

1.7.2.7 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()	O
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
void	setClob(int i, Clob x)	O
void	setDate(int parameterIndex, Date x)	O
void	setDate(int parameterIndex, Date x, Calendar cal)	Δ
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

Return Type	Function Name	Support
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)	Δ
void	setUnicodeStream(int parameterIndex, InputStream x, int length)	X
void	setURL(int parameterIndex, URL x)	X

1.7.2.8 CallableStatement

Return Type	Function Name	Support
Array	getArray(int i)	X
Array	getArray(String parameterName)	X
BigDecimal	getBigDecimal(int parameterIndex)	O
BigDecimal	getBigDecimal(int parameterIndex, int scale)	O
BigDecimal	getBigDecimal(String parameterName)	O
Blob	getBlob(int i)	O
Blob	getBlob(String parameterName)	O
boolean	getBoolean(int parameterIndex)	O
boolean	getBoolean(String parameterName)	O
byte	getByte(int parameterIndex)	O

Return Type	Function Name	Support
byte	getByte(String parameterName)	O
byte[]	getBytes(int parameterIndex)	O
byte[]	getBytes(String parameterName)	O
Clob	getClob(int i)	O
Clob	getClob(String parameterName)	O
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)	O
float	getFloat(int parameterIndex)	O
float	getFloat(String parameterName)	O
int	getInt(int parameterIndex)	O
int	getInt(String parameterName)	O
long	getLong(int parameterIndex)	O
long	getLong(String parameterName)	O
Object	getObject(int parameterIndex)	O
Object	getObject(int i, Map map)	Δ
Object	getObject(String parameterName)	O
Object	getObject(String parameterName, Map map)	X
Ref	getRef(int i)	X
Ref	getRef(String parameterName)	X
short	getShort(int parameterIndex)	O
short	getShort(String parameterName)	O
String	getString(int parameterIndex)	O
String	getString(String parameterName)	O
Time	getTime(int parameterIndex)	O
Time	getTime(int parameterIndex, Calendar cal)	Δ
Time	getTime(String parameterName)	O
Time	getTime(String parameterName, Calendar cal)	Δ

Return Type	Function Name	Support
Timestamp	getTimestamp(int parameterIndex)	O
Timestamp	getTimestamp(int parameterIndex, Calendar cal)	Δ
Timestamp	getTimestamp(String parameterName)	O
Timestamp	getTimestamp(String parameterName, Calendar cal)	Δ
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)	O
void	registerOutParameter(String parameterName, int sql-Type, int scale)	O
void	registerOutParameter(String parameterName, int sql-Type, String typeName)	Δ
void	setAsciiStream(String parameterName, InputStream x, int length)	Δ
void	setBigDecimal(String parameterName, BigDecimal x)	O
void	setBinaryStream(String parameterName, InputStream x, int length)	Δ
void	setBoolean(String parameterName, Boolean x)	O
void	setByte(String parameterName, byte x)	O
void	setBytes(String parameterName, byte[] x)	O
void	setCharacterStream(String parameterName, Reader reader, int length)	Δ
void	setDate(String parameterName, Date x)	O
void	setDate(String parameterName, Date x, Calendar cal)	Δ
void	setDouble(String parameterName, double x)	O
void	setFloat(String parameterName, float x)	O
void	setInt(String parameterName, int x)	O
void	setLong(String parameterName, long x)	O
void	setNull(String parameterName, int sqlType)	O

Return Type	Function Name	Support
void	setNull(String parameterName, int sqlType, String typeName)	Δ
void	setObject(String parameterName, Object x)	O
void	setObject(String parameterName, Object x, int targetSqlType)	O
void	setObject(String parameterName, Object x, int targetSqlType, int scale)	O
void	setShort(String parameterName, short x)	O
void	setString(String parameterName, String x)	O
void	setTime(String parameterName, Time x)	O
void	setTime(String parameterName, Time x, Calendar cal)	Δ
void	setTimestamp(String parameterName, Timestamp x)	O
void	setTimestamp(String parameterName, Timestamp x, Calendar cal)	Δ
void	setURL(String parameterName, URL val)	X
boolean	wasNull()	O

1.7.2.9 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
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

1.7.2.10 Clob

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

1.7.2.11 SavePoint

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

1.7.3 javax.sql package Support Status

ALTIBASE HDB JDBC support status in JDBC 3.0

1.7.3.1 ConnectionPoolDataSource

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

1.7.3.2 DataSource

Return Type	Function Name	Support
Connection	getConnection()	O
Connection	getConnection(String username, String password)	O
int	getLoginTimeout()	Δ
PrintWriter	getLogWriter()	O

Return Type	Function Name	Support
void	setLoginTimeout(int seconds)	Δ
void	setLogWriter(PrintWriter out)	O

1.7.3.3 PooledConnection

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

1.7.3.4 XAConnection

Return Type	Function Name	Support
XAResource	getXAResource()	O

1.7.3.5 XADataSource

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

1.7.4 javax.transaction.xa package Support Status

1.7.4.1 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

1.7.4.2 Xid

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

1.7.5 Data Size Consideration

There is an upper limit to the amount of data that can be uploaded using setByte() to upload data. When uploading more than 64kB of data, use a BLOB stream.

1.8 JDBC Connection Failover

In an environment with multiple simultaneously operating ALTIBASE HDB servers, if one server fails, or if a network problem or the like occurs, the JDBC driver of ALTIBASE HDB may become unable to provide service.

If this happens, the clients that were connected to the server that failed are able to sense the outage and automatically connect to another server.

For information on how to use the Fail-Over functionality within JDBC applications, please refer to Chapter 4 in Replication Manual.

2 The PHP Interface

This chapter explains how to integrate PHP pages with ALTIBASE HDB using PHP's ODBC functionality.

2.1 About the PHP Module of ALTIBASE HDB

1. The following data types are supported for use with ALTIBASE HDB and PHP:

    - `resource`
    - `int`
    - `bool`
    - `double`
    - `float`
    - `string`
    - `array`
    - `HashTable`

2. The port number in db.php, the sample program, must be changed so that it matches the port number that is actually being used on the ALTIBASE HDB server.

2.2 Installing the ODBC Manager for Integration with PHP

In order to integrate ALTIBASE HDB with a PHP server, the ODBC Manager must be installed. This section describes how to install the ODBC Manager in Unix, Linux, and Windows environments.

2.2.1 The ODBC Manager in Unix and Linux

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

1. Download unixODBC.

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

2. Install unixODBC.

   After downloading the unixODBC source files, it is necessary to compile unixODBC. Move the source files to the desired location and run the following commands from a command prompt:

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

3. Configure the unixODBC environment.

   a. Set the $ODBCSYSINI environment variable. Set its value to the same value as the $HOME environment variable for the user that was used to install ALTIBASE HDB.

   ```
   export ODBCSYSINI=~
   ```

   b. Add an environment variable indicating the path where the unixODBC Driver Manager is installed, as shown below. Depending on the platform and whether the OS is a 32-bit or 64-bit OS, the library path will be one of LD_LIBRARY_PATH, LD_LIBRARY_PATH_64 or SHLIB_PATH.

   In the following example, unixODBC is installed at /usr/local/odbcDriverManager32.

   ```
   export LD_LIBRARY_PATH=/usr/local/odbcDriverManager32/
   lib:$LD_LIBRARY_PATH
   ```

   In the following example, unixODBC is installed at /usr/local/odbcDriverManager64.

   ```
   export LD_LIBRARY_PATH=/usr/local/odbcDriverManager64/
   lib:$LD_LIBRARY_PATH
   ```

4. Create the following two files in the $ODBCSYSINI path:

   - odbc.ini
   - odbcinst.ini

   odbcinst.ini must be an empty file having a size of 0 bytes.

   In odbc.ini, specify the DSN name, the path and directory where the ODBC driver of ALTIBASE HDB is installed, and the server address and port number.

   Here are some sample odbc.ini file contents:

   ```
   [Altibase]
   Driver = /home/altibase/altibase_home/lib/libaltibase_odbc.so
   ```

```
Server = 127.0.0.1
Port = 20300
```

2.2.2 The ODBC Manager in Windows

On Windows systems, ODBC data sources are managed using the ODBC Data Source Administrator, which comes preinstalled. When ALTIBASE HDB is installed, the ODBC driver of ALTIBASE HDB is automatically registered in the ODBC Data Source Administrator.

2.3 PHP Functions for ODBC Connectivity

ALTIBASE HDB supports all standard ODBC functions, and thus all ODBC functions that are typically used in PHP pages can be used with ALTIBASE HDB without any special considerations. For a detailed explanation of the ODBC functions that can be used with PHP, please refer to the official PHP online documentation, which can be found at:

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

2.3.1 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);
}
?>
```

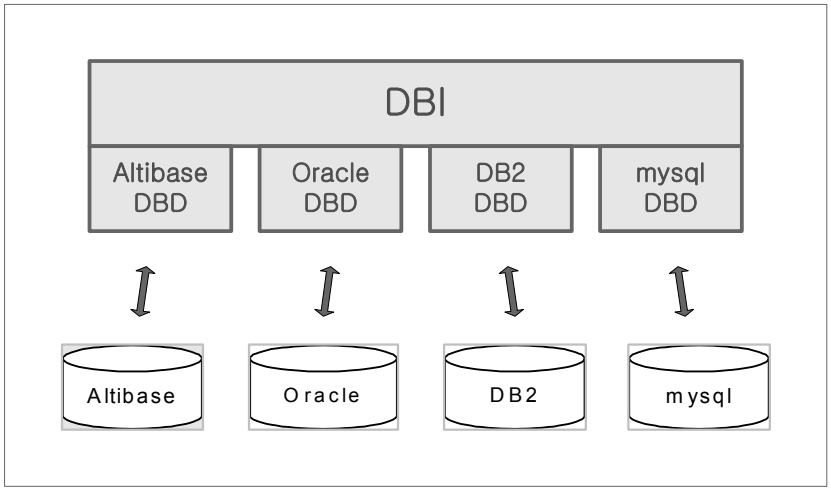
3 PERL DBD DBI

This chapter explains how to use the Perl DBD (Database Driver) and DBI (Database Interface) with ALTIBASE HDB. It describes in detail how to install Perl and the ALTIBASE HDB Perl DBD and how to check the status of the ALTIBASE HDB Perl DBD.

3.1 Overview of the Perl DBD and DBI

A Perl DBI (Database Interface) is a database API written in the form of a PERL5 module. This module defines a series of methods and attributes for accessing a database for which a DBD (Database Driver) has been created. This DBD is also available in the form of a PERL5 module.

The DBI enables an application to access various databases via respective DBDs. The entity that actually handles communication with the database is the DBD. The DBD driver of ALTIBASE HDB is called DBD::altibase. The available attributes and methods are classified as either database attributes and methods or statement attributes and methods.



3.2 Perl Package Installation

3.2.1 The Perl Package Installation Procedure

1. Download the Perl package that is suitable for the OS on which the server is running. (For example, the Perl package for SunOS can be downloaded from http://www.sunfreeware.com.)

   a. Uncompress the package (e.g. perl-5.8.5.tar.gz) in the desired directory.
   ```
   gzip -cd perl-5.8.5.tar.gz | tar xvf –
   ```
   b. Execute the `configure` command in the directory in which the Perl package was uncompressed.
   ```
   ./configure
   ```
   c. Execute the `make` command in the directory in which the Perl package was uncompressed.
   ```
   make
   ```
   d. From the directory in which the Perl package was decompressed, install the package using the root account.
   ```
   make install
   ```

2. Download the Event package (e.g. Event-1.00.tar.gz) to install the Event module.

   a. Uncompress Event-1.00.tar.gz in the directory in which it is desired to install the Event module.
   ```
   gzip -cd Event-1.00.tar.gz | tar xvf –
   ```
   b. Execute the `configure` command in the directory in which the Event package was uncompressed.
   ```
   ./configure
   ```
   c. Execute the `make` command in the directory in which the Event package was uncompressed.
   ```
   make
   ```
   d. From the directory in which the Event package was decompressed, install the module using the root account.
   ```
   make install
   ```

3.3 Installing the ALTIBASE HDB DBD

3.3.1 The ALTIBASE HDB PERL DBD Installation Procedure

3.3.1.1 Check the Perl Installation

Execute `perl` on the command line with the `-V` option to check the value of the dlext configuration variable. It must be `sl` on HP-UX and `so` on other platforms. If this value is not correct, it will be necessary to reinstall Perl.

3.3.1.2 Install the Perl DBI

Install the Perl DBI package, which is a prerequisite for compiling the Perl DBD.

- Method 1) Using the root account:

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

- Method 2) If the above method does not work, download the package via ftp from the address shown below, compile it, and install it using the following commands.

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

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

3.3.1.3 Download and Uncompress the ALTIBASE HDB Perl DBD Files

Visit http://support.altibase.com and download the file from the location shown below. Because the ALTIBASE HDB Perl DBD is a 32-bit driver, the ALTIBASE HDB 32-bit client package or 32-bit server package must already be installed on the system on which the DBD is being installed. Additionally, the value of the $ALTIBASE_HOME environment variable must be correctly set.

```
/download_back/altibase/PERL-DBD/altibase-perlDBD-YYYYMMDD.tar.gz
```

Uncompress the file.

```
gzip -cd altibase-perlDBD-YYYYMMDD.tar.gz | tar xvf -
```

3.3.1.4 Make the ALTIBASE HDB Perl DBD

Execute `make` on the install.mk file, which was included in the compressed file downloaded in the previous step, to create the Makefile which will be used to create the DBD.

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

Execute `make` again without any arguments to specify that Makefile is to be used as the source. This creates the shared library that is used as the Perl DBD of ALTIBASE HDB. On HP-UX, the resultant filename is altibase.sl. On platforms other than HP-UX, the resultant filename is altibase.so, and it is cre-

ated in blib/arch/auto/DBD/altibase.

```
make
```

Execute `make install` as root. This installs the Perl DBD of ALTIBASE HDB in Perl.

```
make install
```

In this example, the ALTIBASE HDB DBD can be seen in the folder shown below:

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

3.3.1.5 Make Additional Environment Settings

Set the $LD_PRELOAD environment variable. This task must be performed on HP-UX and certain other platforms. On these platforms, if this environment variable is not set, an error message will be displayed.

3.3.1.6 Test the ALTIBASE HDB Perl DBD Installation

After starting the ALTIBASE HDB server, change the values of the DSN and PORT_NO keyword/value pairs in the test.pl file to suitable values.

```
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});
```

Finally, test the Perl installation by executing Perl on the test.pl file.

```
perl test.pl
```

4 .NET Data Provider

This chapter describes how to use Microsoft's ADO.NET interface with ALTIBASE HDB.

4.1 An Overview of the .NET Data Provider

4.1.1 Overview

The ALTIBASE HDB 5 .NET Data Provider is an implementation of Microsoft's ADO.NET interface for use with ALTIBASE HDB. That is, the ALTIBASE HDB 5 .NET Data Provider is required in order for applications based on the .NET Framework to be able to access ALTIBASE HDB.

The ADO.NET interface uses the .NET Framework to access data sources such as DBMS. The .NET Framework already supports the ODBC and OLEDB interfaces and provides methods for accessing SQL Server and Oracle.

The role of a .NET Framework-based Data Provider is to access data sources, execute commands and return the results. Developers use the ADO.NET DataSet class to process results returned in this way and to write data to the data sources.

Connecting to a database is time consuming; connection pooling reduces this time by managing connections in a pool. Available connections are assigned to requests and returned to the pool after use, and idle connections are removed.

The biggest advantage of the ADO.NET interface is that when ALTIBASE HDB is upgraded, the Data Provider and client applications can continue to be used without having to change them, as long as the CLI library is also upgraded to match the new version of ALTIBASE HDB. For more information on ADO.NET, please refer to the Microsoft website (http://www.msdn.com).

4.1.2 Requirements

- .NET Framework
  - The .NET Framework must be installed. Additionally, the version of the .NET Framework must be compatible with the version of ADO.NET that is being used. For example, if ADO.NET 1.0 is being used, the version of the .NET Framework must be version 1.1 or a subsequent release. If using ADO.NET 2.0, the version of the .NET Framework must be version 2.0 SP1 or a subsequent release.
  - Different .NET Data Provider library files are provided for different versions of ADO.NET, and thus it is essential that you use a version of the library that is suitable for the version of ADO.NET to be used.
  - The .NET Data Provider included in the ALTIBASE HDB package is recommended for use on .NET Framework 3.5 SP1 or above. You can request for the Data Provider for .NET Framework 2.0 at Altibase's Customer Support site (http://support.altibase.com/).
- Altibase CLI Library

  Because the .NET Data Provider uses CLI libraries to connect to databases, the CLI library of ALTIBASE HDB must be installed. This is provided in the ALTIBASE HDB package (server or client package) by default.

4.1.3 Considerations

- The dependency of the .NET Data Provider on CLI libraries means that it is essential that the

version of the odbccli_sl.dll library is correct for the ALTIBASE HDB server with which it is associated.

- A different .NET Data Provider is built for each version of the .NET Framework, so be sure to use the version of the .NET Data Provider that is intended for use with the version of the .NET Framework that you are using.
- When using the ColumnName property with classes such as DataReader and CommandBuilder, remember that it is case-sensitive. When creating a table, any column names that are not placed inside double quotation marks will be changed to all upper-case letters. Columns created in this way can still be used normally as long as they are referenced by their actual names, that is, in upper-case letters.
- Data loss may occur when using the DataReader.GetValue() function to retrieve data from NUMBER, NUMERIC, FLOAT and DECIMAL type columns. The reason for this is that this function converts numeric data to the .NET System.Decimal type, whose capacity is less than those of the above data types.
- The ALTIBASE .NET Data Provider does not support the execution of multiple queries. To execute a multiple number of queries simultaneously, the user should use stored procedures.
- For further information about considerations for connection pooling, please refer to Connection Pooling > Considerations.
- For further information about considerations for distributed transactions, please refer to DTC Considerations.

4.2 Using the ALTIBASE HDB .NET Data Provider

The ALTIBASE HDB .NET Data Provider can be found in the "lib" subdirectory in the directory in which ALTIBASE HDB is installed (which can be found using the environment variable %ALTIBASE_HOME%). The complete path and filename are as follows:

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

4.2.1 Compiling .NET Applications

An application that uses the ALTIBASE HDB .NET Data Provider can be compiled in one of two ways:

4.2.1.1 Compilation on the Command Line

When compiling an application on the command line, include a reference to the DLL as shown below:

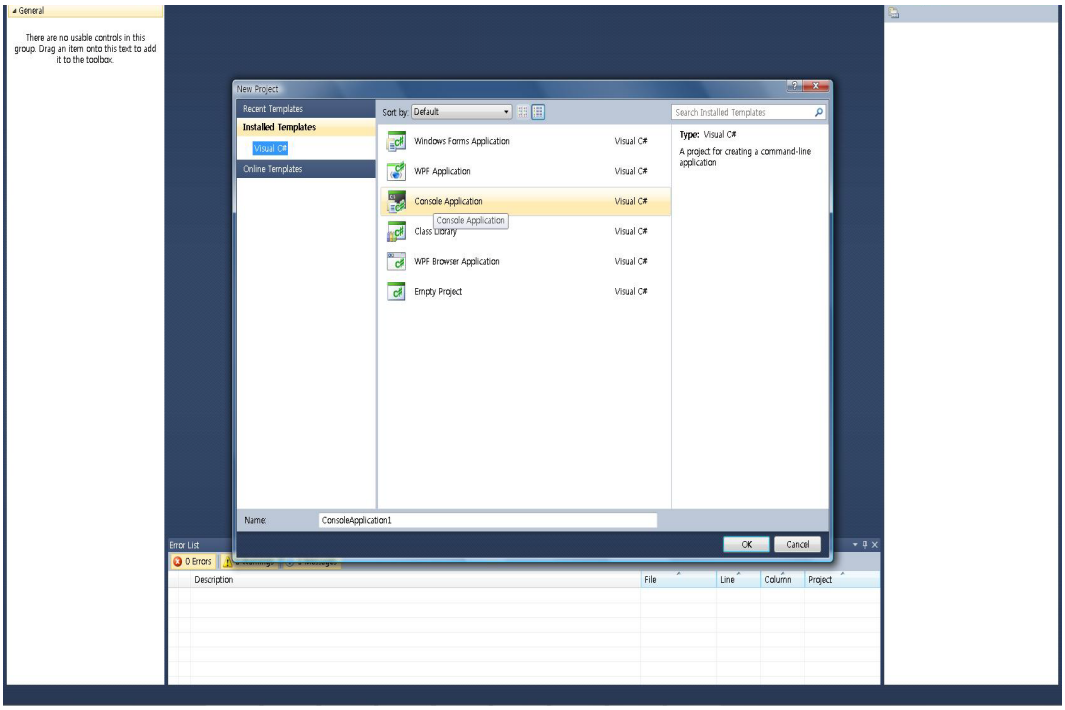
```
csc /r:%ALTIBASE_HOME%\lib\Altibase.Data.AltibaseClient.dll program_name.cs
```

4.2.1.2 Compilation in an IDE Environment

The following example shows how to register the ALTIBASE HDB .NET Data Provider in an IDE (Integrated Development Environment).

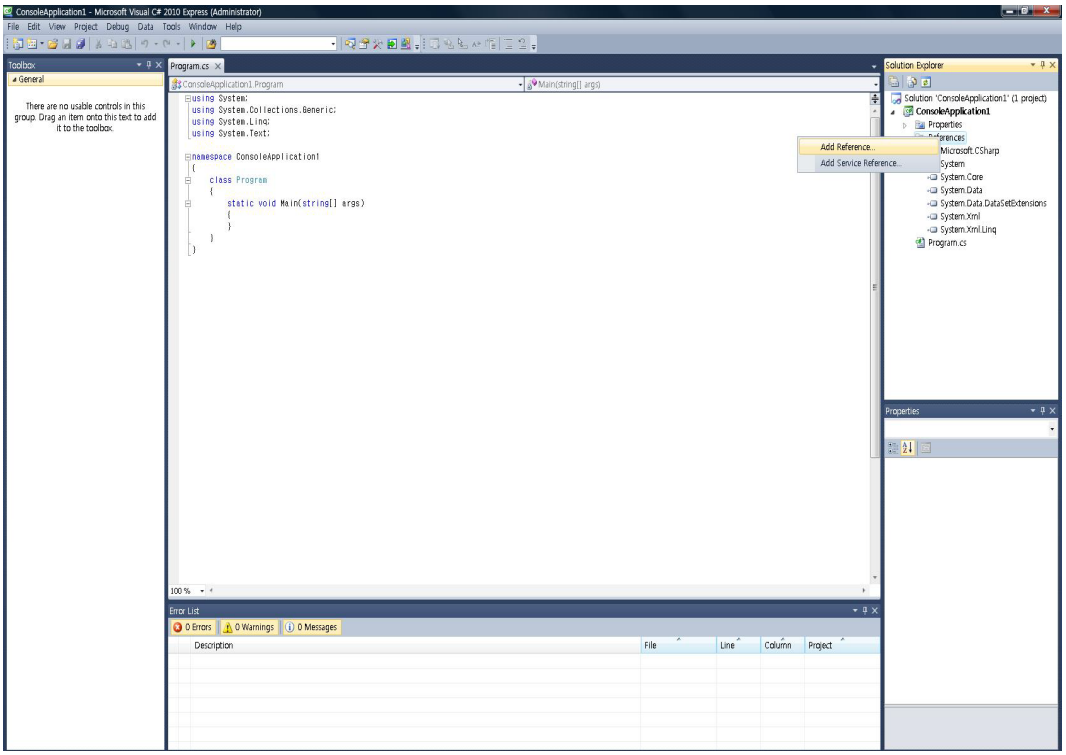
1. Open a new project [Figure 4-1].

Figure 4-1 Opening a New Project



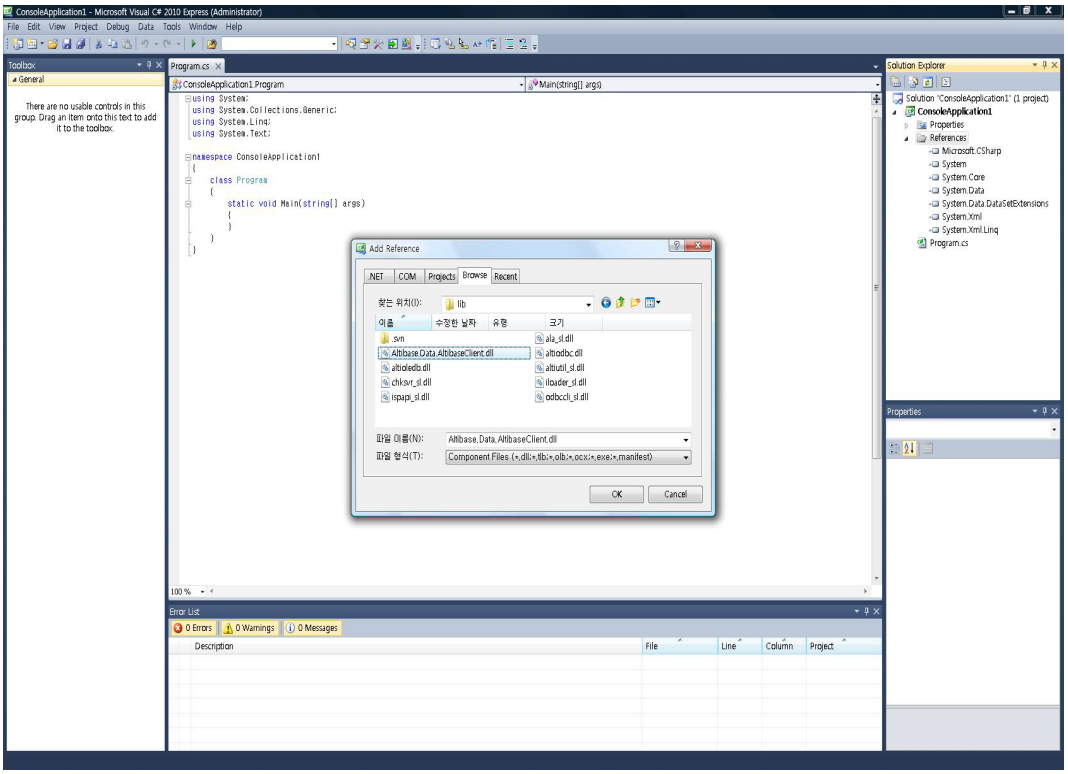
2. To register the ALTIBASE HDB .NET Data Provider [Figure 4-2], right-click on “References” and click “Add Reference…”.

Figure 4-2 Add a Reference to the DLL



3. Locate and register Altibase.Data.AltibaseClient.dll in the “lib” subdirectory in the directory in which ALTIBASE HDB is installed. The installation directory of ALTIBASE HDB can be found using the environment variable %ALTIBASE_HOME%.

Figure 4-3 Register ALTIBASE HDB .NET Data Provider with Project



4. Build the project and execute the resulting executable file.

4.2.2 Array Binding

The ALTIBASE HDB .NET Data Provider supports array binding, that is, binding data that are in the form of arrays to parameters. Because this reduces the use of network resources when compared to the usual binding method, an improvement in speed can be expected when processing multiple rows.

At present, array binding is supported for input parameters only; the binding of output and input/output parameters is not supported.

The procedure for binding arrays is as follows:

1. Ensure that all of the variables to be bound are in array form. The size of each array must be greater than or equal to the setting of the ArrayBindCount property in the AltibaseCommand class.

2. Bind the array variable(s) to the parameter(s). If the column to be bound is a CHAR, VARCHAR or BLOB type column, the value of the ArrayBindSize property in the AltibaseParameter class must be set to the same size as the largest of the elements in the array.

3. Set the value of the ArrayBindCount property in the AltibaseCommand class.

   For example, to insert 100 values at a time, set the value of the ArrayBindCount property to 100.

4. Execute SQL statements using the bound array(s).

4.2.2.1 Considerations

When binding arrays, it is important to keep the following in mind:

- The valid range of the ArrayBindCount property is from 1 to 65535. No performance benefit is realized by creating very large arrays, so this property should be set appropriately for the amount of data being handled.
- If the length of the data for a single element of a CHAR, VARCHAR or BLOB type array exceeds the value of ArrayBindSize, an error occurs.
- When working with the NCHAR and NVARCHAR (i.e. Unicode character) data types, the value of the ArrayBindSize property is the number of characters, not bytes.
- For arrays of BLOB type data, handle the array within the application as an object, and each array element as a byte[].

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

- Array binding is not supported for CLOB, BYTE, NIBBLE, BIT, VARBIT and GEOMETRY type arrays.

4.2.2.2 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");

        // table: create table t1 (c1 int, c2 varchar(12));

        // 3 records
        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;

        //=====================================================
        // 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.Ansi-
String);
        prm2.Direction = ParameterDirection.Input;
        prm2.Value = c2;
        prm2.ArrayBindSize = 12; // max element size in bytes

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

        //=================================================
        // execute INSERT
        //=================================================
        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();
    }
}
```

4.2.3 Declaring the ALTIBASE HDB .NET Data Provider

In order to use the ALTIBASE HDB .NET Data Provider classes, it is first necessary to declare the ALTIBASE HDB .NET Data Provider as shown below:

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

4.2.4 Processing Transactions

Transactions can be processed using the transaction processing interface provided with ADO.NET.

To process transactions using the ADO.NET interface, use the AltibaseConnection.BeginTransaction() method and the AltibaseTransaction class.

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

// Commencing a transaction
AltibaseTransaction sTrans = sConn.BeginTransaction();

AltibaseCommand sCmd = sConn.CreateCommand();
// Processing the transaction
// TODO
```

```
// Terminating the transaction
s.sTrans.Commit();
```

4.2.5 Connection Pooling

An application connects to the database server through a series of steps which is time-consuming and repetitive. ADO.NET provides connection pooling to minimize the number of times a connection is opened and closed.

The ownership of connections is maintained in the connection pool. When the pooler receives a request for a connection, it checks whether there is an available connection. If so, it is assigned; if not, a new connection is created and then assigned. When a connection is closed, it is not removed, but returned to the pool.

The size of a connection pool depends on the `Max Pool Size` parameter. The default value is 100.

4.2.5.1 Creating a Pool

When a connection is opened, it is pooled in an existing connection pool that matches the algorithm of the connection string; a connection pool is created if no connection pools match the algorithm of the connection string. The case-sensitivity, properties, and number/position of blank spaces in a connection string are also evaluated when finding a matching connection pool. The number of connection pools automatically created corresponds to the value set for the `Min Pool Size` property; on omission, no connection pools are created because the default value is 0.

4.2.5.2 Assigning and Adding Connections

When a connection is requested, the connection string is evaluated for a matching connection pool. If there is a match, a connection is assigned; if not, a connection pool is created and then a connection is assigned. The number of connection pools allowed to be created corresponds to the value set for the `Max Pool Size` property (default value: 100). When the number of connection pools reaches the limit, the connection waits for a connection pool which exceeds the `Connection Life Time` value, to throw an exception and be removed. When a connection is closed, it is returned to the connection pool for future use.

4.2.5.3 Removing Connections

If an application does not close or remove a connection, the pooler checks and removes unnecessary connections on a regular basis. A connection is removed if it is idle for the amount of time set by the `Connection Life Time` property; the `Min Pool Size` property determines the number of idle connections to be retained.

A connection that throws an exception is also removed. If the user selects a connection that has been closed and returned to a connection pool, an exception can be thrown at actual connection because it has not been evaluated as to whether it is a match.

4.2.5.4 Deleting Connection Pools

The `ClearPool` and `ClearAllPools` methods of the `AltibaseConnection` class allow the user to delete connection pools. `ClearPool` deletes specified connection pools, whereas `Clear-`

`AllPools` deletes all connection pools.

4.2.5.5 Restrictions

The `Min Pool Size` property sets the number of connection pools to be created automatically; this number cannot exceed the number of connections allowed to concurrently connect to the server. The `Pool Size` property sets the number of connection pools the client is able to connect to.

4.2.6 Connection Properties

4.2.6.1 application name

Default Value	.NET Altibase Data Provider
Range	Random string
Mandatory	No
Setting Range	N/A
Description	Information about the client application that is connected. This value is printed under the `CLIENT_APP_INFO` column of `V$SESSION`.

4.2.6.2 connection life time

Default Value	0
Range	A numeric value within the unsigned integer range $[1\text{-}2^{31}]$
Mandatory	No
Setting Range	N/A
Description	The amount of time after which an idle connection is to be removed from the connection pool. The unit is seconds; the default value 0 indicates infinity.

4.2.6.3 connection timeout

Default Value	15
Range	A numeric value within the unsigned integer range $[1\text{-}2^{31}]$

Mandatory	No
Setting Range	N/A
Description	The amount of time to wait for a connection. The unit is seconds; the default value 0 indicates infinity.

4.2.6.4 data source

Default Value	
Range	
Mandatory	No
Setting Range	N/A
Description	The data source name. This value depends on the following conditions. • If the server value has been omitted, the data source value is ignored. • If the server value has been omitted, but an ODBC data source with the same value as the data source exists, the Data Source Name (DSN) is used • If the server value has been omitted and an ODBC data source with the same value as the data source does not exist, the server IP or host name is used.

4.2.6.5 encoding

Default Value	
Range	
Mandatory	No
Setting Range	N/A
Description	The character set for the client.

4.2.6.6 enlist

Default Value	True
Range	[True | False]
Mandatory	No
Setting Range	N/A

Description	The transaction context of the thread (the `Transaction.Current` property). In other words, whether or not to join the ambient transaction in which code is currently being executed.

4.2.6.7 max pool size

Default Value	100
Range	A numeric value within the unsigned integer range $[1\text{-}2^{31}]$
Mandatory	No
Setting Range	N/A
Description	The maximum number of connection pools that can be created.

4.2.6.8 min pool size

Default Value	0
Range	A numeric value within the unsigned integer range $[1\text{-}2^{31}]$
Mandatory	No
Setting Range	N/A
Description	The minimum number of connection pools allowed to remain in the pooler. When a connection pool is created, as many pools as set for this property are created. Pools that stay idle for longer than `connection life time` are removed; but the pooler retains as many pools as set for this property.

4.2.6.9 nchar literal replace

Default Value	False
Range	[True | False]
Mandatory	No
Setting Range	Session
Description	Whether or not to use a string constant in the national character set in a SQL statement.

4.2.6.10 password

Default Value	
Range	The database user's password
Mandatory	Yes
Setting Range	N/A
Description	The user password of the database server to connect to.

4.2.6.11 persist security info

Default Value	False
Range	[True | False]
Mandatory	No
Setting Range	N/A
Description	Whether or not to include the password when retrieving a connection string.

4.2.6.12 pooling

Default Value	True
Range	[True | False]
Mandatory	No
Setting Range	N/A
Description	Whether or not to use connection pooling.

4.2.6.13 port

Default Value	20300
Range	[0~65535]
Mandatory	No
Setting Range	N/A
Description	The ALTIBASE HDB server port number to connect to.

4.2.6.14 prefer ipv6

Default Value	False
Range	[True | False]
Mandatory	No
Setting Range	N/A
Description	Whether or not IPv6 has higher priority than IPv4. If false, an IPv4 address is used to connect to the database server.

4.2.6.15 server

Default Value	localhost
Range	
Mandatory	No
Setting Range	N/A
Description	The IP address or host name of the ALTIBASE HDB server to connect to.

4.2.6.16 transaction timeout

Default Value	The value set for the server
Range	A numeric value within the unsigned integer range [1-2^{31}]
Mandatory	No
Setting Range	N/A
Description	The amount of time (in seconds) after which to automatically terminate the execution of an update operation. The unit is seconds; the default value 0 indicates infinity.

4.2.6.17 user id

Default Value	
Range	The database user ID

Mandatory	Yes
Setting Range	N/A
Description	The user ID of the database server to connect to.

4.2.7 Schema

In ALTIBASE HDB, the use of the ADO.NET GetSchema() method can of course be used to return general schema information, such as information about the MetaDataCollections, DataSourceInformation, DataTypes, Restrictions and ReservedWords collections. It can additionally be used to query the following Altibase-specific schema information pertaining to meta tables and performance views:

Table 4-1 Altibase-Specific Meta Table and Performance View Schema supported for Use with GetSchema()

Schema	Meta Table or Performance View	Description
Users	SYS_USERS_	A meta table containing information about users
Tables	SYS_TABLES_	A meta table containing information about all kinds of tables
Views	SYS_VIEWS_	A meta table containing information about views
Sequences	V$SEQ	A performance view containing information related to sequences
Synonyms	SYS_SYNONYMS_	A meta table containing information about synonyms
Indexes	SYS_INDICES_	A meta table containing information about indexes
Columns	SYS_COLUMNS_	A meta table containing information about columns
Constraints	SYS_CONSTRAINTS_	A meta table containing information about constraints
Procedures	SYS_PROCEDURES_	A meta table containing information about stored procedures and functions
ProcedureParameters	SYS_PROC_PARAS_	A meta table containing information about the parameters for stored procedures and functions

Schema	Meta Table or Performance View	Description
IndexColumns	SYS_INDEX_COLUMNS_	A meta table containing information about index columns
ConstraintColumns	SYS_CONSTRAINT_COLUMNS_	A meta table containing information about constraint columns
Triggers	SYS_TRIGGERS_	A meta table containing information about triggers

For more information about the meta tables and performance views provided in ALTIBASE HDB, please refer to the chapter in the *ALTIBASE HDB General Reference* that explains the data dictionary.

4.2.8 ALTIBASE HDB .NET Data Provider Classes

The ALTIBASE HDB .NET Data Provider provides functions for connecting to Altibase databases, executing queries, and retrieving results. These functions are based on the four classes shown in Table 4-2. For detailed information on the functionality of the methods in each class, please refer to the Microsoft ADO.NET documentation.

Table 4-2 ALTIBASE HDB .NET Data Provider Classes for Connecting to Databases, Executing Queries and Retrieving Results

Class	Description
AltibaseConnection	This class is used to make connection settings and establish a connection with a database so that transactions can be executed.
AltibaseCommand	This class is used to execute a query in a database and set parameters.
AltibaseDataReader	Once a command has been executed on a database, this class is used to retrieve and output the results.
AltibaseDataAdapter	This class is used to write data to an instance of a DataSet class and update a database with the data stored therein.

The ALTIBASE HDB .NET Data Provider provides the following classes for handling exceptions, managing stored procedures, and processing transactions:

Table 4-3 ALTIBASE HDB .NET Data Provider Classes for Processing Transactions and Handling Exceptions

Class	Description
AltibaseException	This class is thrown when ALTIBASE HDB server returns an error, and can be used to handle database errors and client errors received via the .NET Framework.

Class	Description
AltibaseParameter	This class is used to define input and output parameters for use with SQL commands and stored procedures.
AltibaseTransaction	This class supports the execution of transaction-related commands in the database.

4.2.9 ALTIBASE HDB .NET Data Provider Data Types

The AltibaseDbType class is used when declaring the data type of table columns and parameters. Table 4-4 shows the relationships between the data types provided by ALTIBASE HDB/the AltibaseDbType class and the .NET Framework.

Table 4-4 The Relationships between ALTIBASE HDB and .NET Data Types

AltibaseDbType	ALTIBASE HDB Column Type	.NET Framework
BigInt	BIGINT	Int64
BitArray	BIT	BitArray
Blob	BLOB	Byte[]
Binary	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
NibbleArray	NIBBLE	NibbleArray
Number	NUMBER	Decimal
Numeric	NUMERIC	Decimal
NVarChar	NVARCHAR	String
Real	REAL	Float
SmallInt	SMALLINT	Int16

AltibaseDbType	ALTIBASE HDB Column Type	.NET Framework
VarBitArray	VARBIT	BitArray
VarChar	VARCHAR	String

To use a character string constant in a SQL statement when working with a national character set, add the character "N" in front of the character string.

4.2.10 Considerations

The following are considerations for using the Altibase Entity Provider.

- When creating a BIT or VARBIT column in the database, its size must be 1.
- When creating a NUMERIC or NUMBER column in the database, the precision must be between 1 and 38, and the scale must be between 0 and the precision value.
- The NIBBLE, BLOB and CLOB data types are unsupported. Instead, you can use BYTE for NIBBLE and BLOB, and VARCHAR for CLOB.

4.3 .NET Data Provider Interface

4.3.1 Unsupported Interfaces

The following is a list of interfaces unsupported by the Altibase .NET Data Provider.

Class	Identification	Component
AltibaseConnection	M	ChangeDatabase
AltibaseCommand	M	Cancel
	P	CommandTimeout
	P	CommandType
AltibaseDataReader	M	GetData
	P	Depth
AltibaseParameter	M	Clone
AltibaseParameterCollection	M	AddRange
M: Method P: Property A: All the class		

Class	Identification	Component
AltibaseFactory	M	CreateDataSourceEnumerator
	M	CreatePermission
AltibaseDataReader	M	GetDbDataReader
	M	GetProviderSpecificFieldType
	M	GetProviderSpecificValue
	M	GetProviderSpecificValues
	P	HasRows
	P	VisibleFieldCount

Class	Identification	Component
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	

Methods that are derived from base constructors in abstract classes that are inherited are the same as the corresponding base methods. The members that provide base constructors are as follows:

- CreateCommandBuilder
- CreateConnectionStringBuilder
- CreateDataSourceEnumerator
- CreatePermission
- GetProviderSpecificFieldType
- GetProviderSpecificValue
- GetProviderSpecificValues
- VisibleFieldCount
- GetBatchedRecordsAffected(int commandIdentifier, out int recordsAffected, out Exception error)

All members that are not based on base constructors will raise a NotImplementedException.

4.4 .NET Data Provider Examples

4.4.1 Connection String

The following connection string establishes a connection to ALTIBASE HDB from .NET applications.

```
Server=127.0.0.1;PORT=20091;User=sys;Password=manager
```

4.4.2 Simple Example with DDL and DML

Use the AltibaseConnection class to connect to an Altibase database, create the *test_goods* table, insert some data, and then query the data.

```
using Altibase.Data.AltibaseClient;

class ConnectionTest
{
    static void Main(string[] args)
    {
        string sConnectionString =
"Server=127.0.0.1;PORT=20091;User=sys;Password=manager";
        AltibaseConnection conn = new AltibaseConnection(sConnectionString);

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

            try
            {
                command.ExecuteNonQuery(); // This executes a query
            }
            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 a query

            command.CommandText =
            "insert into test_goods values
              ('A111100001','IM-300','AC0001',1000,78000)";
            command.ExecuteNonQuery();//This executes a query

            command.CommandText =
            "insert into test_goods values
              ('A111100002','IM-310','DD0001',100,98000)";
            command.ExecuteNonQuery();//This executes a query

            command.CommandText =
            "insert into test_goods values
              ('B111100001','NT-H5000','AC0002',780,35800)";
            command.ExecuteNonQuery();//This executes a query
```

```
            command.CommandText = "select * from test_goods";
            AltibaseDataReader dr = command.ExecuteReader();
            Console.WriteLine(" GNO GNAME LOCATION STOCK PRICE ");
            Console.WriteLine(
"=============================================================================
========");

            while (dr.Read())
            {
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    Console.Write("\t{0}", dr[i]);
                }
                // This outputs the retrieved data
                Console.WriteLine();
            }
        }
        catch (Exception ex)
        {
          Console.WriteLine(ex.ToString());
        }
        conn.Close(); // This closes the connection to the 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
```

4.4.3 Bulkcopying

Use the AltibaseBulkCopy to copy data from BULKCOPY_SOURCE table to BULKCOPY_DESTINATION table

```
using System;
using System.Data;
using Altibase.Data.AltibaseClient;
class Program
{
    static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            Console.WriteLine("\t[NOT PASSED]: Invalid argument");
            return;
        }

        string connectionString = GetConnectionString(args);
        using (AltibaseConnection sourceConnection =
                new AltibaseConnection(connectionString))
        {
            sourceConnection.Open();

            // Perform an initial count on the destination table.
            AltibaseCommand commandRowCount = new AltibaseCommand(
                "SELECT COUNT(*) FROM BULKCOPY_DESTINATION;",
                sourceConnection);
```

```
            long countStart = System.Convert.ToInt32(
                commandRowCount.ExecuteScalar());

            Console.WriteLine("Starting row count = {0}", countStart);

             // Get data from the source table as a AltibaseDataReader.
            AltibaseCommand commandSourceData = new AltibaseCommand(
                "SELECT A1, A2, A3, A4 FROM BULKCOPY_SOURCE;",
                 sourceConnection);

            AltibaseDataReader reader =
                commandSourceData.ExecuteReader();

            // Open the destination connection. In the real world you would
            // not use AltibaseBulkCopy to move data from one table to the
other
            // in the same database. This is for demonstration purposes only.
            using (AltibaseConnection destinationConnection =
                       new AltibaseConnection(connectionString))
            {
                destinationConnection.Open();

                // Set up the bulk copy object.
                // Note that the column positions in the source
                // data reader match the column positions in
                // the destination table so there is no need to
                // map columns.
                using (AltibaseBulkCopy bulkCopy =
                           new AltibaseBulkCopy(destinationConnection))
                {
                    bulkCopy.DestinationTableName =
                        "BULKCOPY_DESTINATION";

                    try
                    {
                        // Write from the source to the destination.
                        bulkCopy.WriteToServer(reader);
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(ex.Message);
                    }
                    finally
                    {
                        // Close the AltibaseDataReader. The AltibaseBulkCopy
                        // object is automatically closed at the end
                        // of the using block.
                        reader.Close();
                    }
                }

                // Perform a final count on the destination
                // table to see how many rows were added.
                long countEnd = System.Convert.ToInt32(
                    commandRowCount.ExecuteScalar());

                Console.WriteLine("Ending row count = {0}", countEnd);
                Console.WriteLine("{0} rows were added.", countEnd - count-
Start);
            }
        }
    }
    private static string GetConnectionString(string[] args)
    // To avoid storing the sourceConnection string in your code,
    // you can retrieve it from a configuration file.
```

```
    {
        return "Server=" + args[0] + ";" + "PORT=" + args[1] + ";" +
"User=sys;Password=manager";
    }
}
```

4.4.4 Connection Pooling

The following example uses connection pooling in ADO.NET.

```
static void Main(string[] sArgs)
{
    AltibaseConnection cn = new AltibaseConnection();
    AltibaseConnection cn2 = new AltibaseConnection();
    AltibaseCommand cmd = new AltibaseCommand();
    string cnStr = "Server=127.0.0.1;Port=20300;User=user;Password=pwd;Pool-
ing=true;Min Pool Size=0;Max Pool Size=10";

    cn.ConnectionString = cnStr;
    cn.Open();

    Console.WriteLine("Successfully Connected.");

    cmd.Connection = cn;
    cmd.CommandText = "SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD HH:MI:SS') FROM

DUAL";

    AltibaseDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
        Console.WriteLine(dr[0].ToString());
    }
    dr.Close();

    cn.Close();

    cn2.ConnectionString = cnStr;
    cn2.Open(); // Pooled Connection
    cn2.Close();
}
```

5 OLE DB

This chapter describes current support of ALTIBASE HDB for OLE DB. It explains how to install and uninstall OLE DB and how to use OLE DB to access data in a variety of environments, such as ADO, ADO.NET, etc. This chapter also explains the data types and properties that are supported in ALTIBASE HDB for use with OLE DB.

5.1 Overview of OLE DB

OLE DB (Object Linking and Embedding, Database) is an open interface developed by Microsoft for accessing a wide variety of data sources.

Although a universal database access API already exists in the form of ODBC, it is limited in its ability to support new ways of designing and building data-dependent applications.

Its successor, OLE DB, supports open database access in a way that provides greater functionality. OLE DB uses a COM-based programming interface to support access to all types of data, including relational, non-relational and hierarchical data.

5.2 Installation

This section covers how to install and uninstall OLE DB. Additionally, after OLE DB is installed, there are a number of items to check before it is possible to use the ODBC driver. These items are set forth here. This section also explains how to use OLE DB to access an Altibase database in various operating environments and how to map OLE DB data types to ALTIBASE HDB data types.

- Installing and Uninstalling OLE DB
- Verifying the OLE DB Installation
- Creating a Connection String
- ALTIBASE HDB and OLE DB Data Types

5.2.1 Installing and Uninstalling OLE DB

The OLE DB provider of ALTIBASE HDB is not installed automatically when ALTIBASE HDB is installed. Therefore, it needs to be installed manually by running the command shown below. This command registers the altioledb.dll dynamic link library, in which the OLE DB provider of ALTIBASE HDB is implemented. This library file is located at $ALTIBASE_HOME/lib.

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

Use the following command to uninstall the OLE DB provider of ALTIBASE HDB.

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

5.2.2 Verifying the OLE DB Installation

To verify the installation of the ALTIBASE HDB OLE DB provider, follow the steps outlined below:

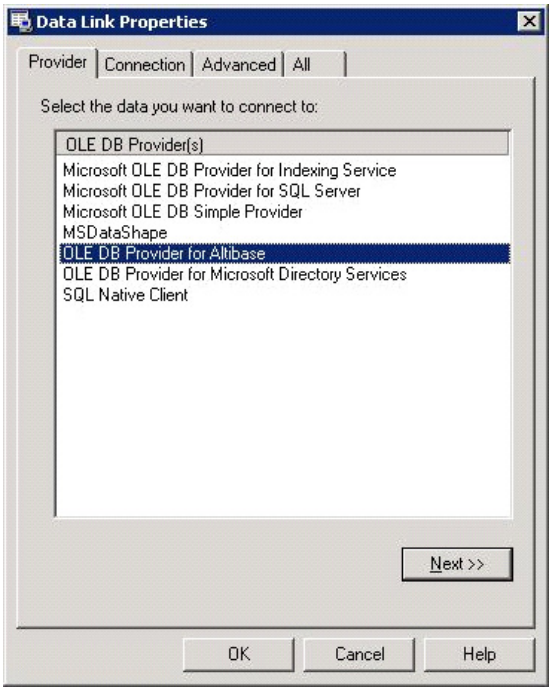
To verify that the OLE DB provider of ALTIBASE HDB has been installed:

1. Get all connection properties pertaining to the ALTIBASE HDB server to which to connect, including the server IP address, the port number, and the user account of ALTIBASE HDB.

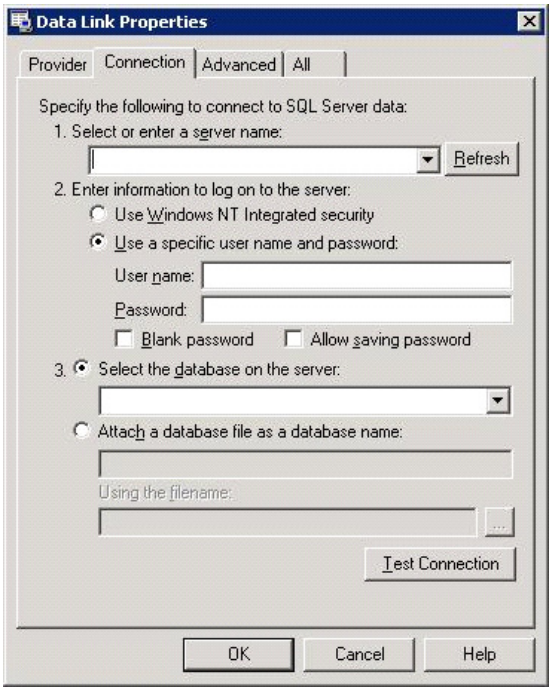
2. Create an OLE DB Universal Data Link (UDL) file.

   - Go to the folder where the UDL file is to be created.
   - Create a new text document in this folder.
   - The name of the text file must have the udl file name extension (i.e. the file must be named *.udl). Therefore, rename the empty file that was created in the previous step as desired, making sure that you change the extension from .txt to .udl.
   - Double-clicking this file should cause the panel of data access properties for the .udl file to appear.

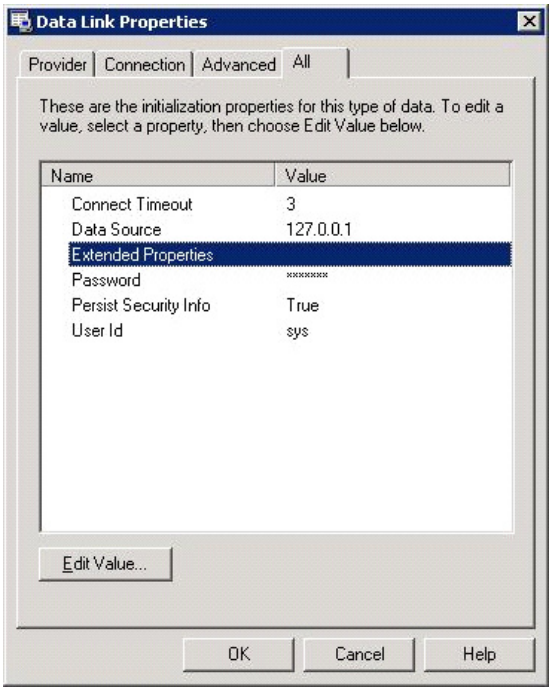
3. Go to the “Provider” tab and select the provider to use. As you can see, there is a wide variety of OLE DB providers to choose from. In this example, we will use the OLE DB Provider for ALTIBASE HDB. Then, click the “Next” button.



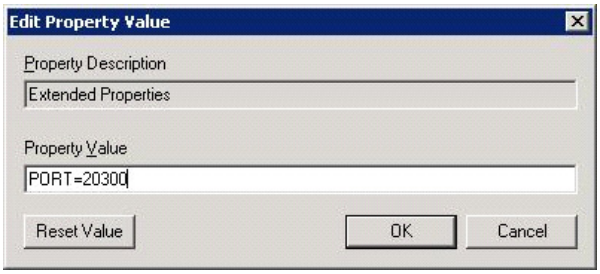
4. Go to the "Connection" tab and add the data source name and login information.



5. Now click on the "All" tab. It is a good idea at this point to change the values of the initialization properties for the OLE DB provider of ALTIBASE HDB. Select "Extended Properties" and then click the "Edit Value" button to set the property value.



6. The “Extended Properties” value has the following format: “keyword=value;keyword=value”. When specifying multiple keywords, they are delimited using a semicolon (“;”). In this example, the port number is set to 20300 (“PORT=20300”).

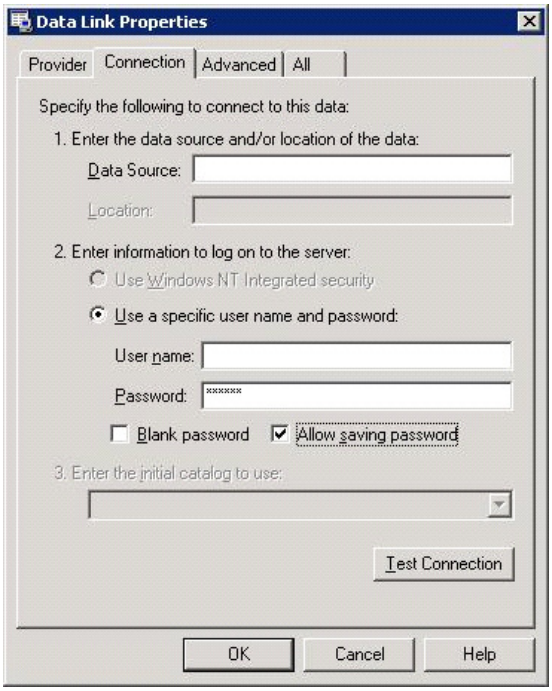


7. The settings are now complete. Go back to the “Connection” tab and click on the “Test Connection” button to ensure that the connection works as expected. The following box will appear if the connection has been correctly configured.



8. Close this message box and check the “Allow saving password” checkbox in the “Connection” tab. Then click on the “OK” button. At this point, another dialog box will appear asking you whether you want to save your password. Saving your password is useful for subsequent connection testing. However, for security reasons, saving your password in this way in a produc-

tion environment is not recommended. Because the UDL file is an unencrypted text file, it could be opened and read by malicious users.





5.2.3 Creating a Connection String

This section explains how to create and use OLE DB connection strings for accessing an Altibase database in an ADO or ADO.net environment. It includes the following topics:

- Typical Connection Strings
- ADO Connection Strings
- ADO.net Connection Strings

5.2.3.1 Typical Connection Strings

A typical connection string consists of these keywords:

Keyword	Description
Provider	This specifies the OLE DB Provider of ALTIBASE HDB. In order to access ALTIBASE HDB, this must always be set to Altibase.OLEDB.
Data Source	This specifies the database path or the name of an ODBC DSN, in which information about the database path is stored.
User ID	This specifies the user name.
Password	This specifies the user password.
Extended Properties	This is used to specify multiple keywords, other than the Provider, User ID and Password. It follows this format: "keyword=value;keyword=value" As shown, multiple keywords are delimited using semicolons (";").
File Name	This specifies the name of a data link file (*.udl) containing preset connection information.

For example, if the user account is the sys account, the password is "manager" and the IP and port number of the server are 192.168.2.1:20300, a typical connection string including all of this information would look like this:

```
Provider=Altibase.OLEDB;Data Source=192.168.2.1;User Id=sys;Password=man-
ager;Extended Properties="PORT=20300;NLS_USE=US7ASCII"
```

If the connection information of ALTIBASE HDB is stored in an ODBC DSN, it is possible to simply refer to the information in this DSN. This obviates the need to specify a large number of keyword/value pairs in the connection string, and also make the connection string simpler and shorter. In this example, altibase_odbc is the name of the DSN containing the connection information. A typical connection string containing this information looks like this:

```
Provider=Altibase.OLEDB;Data Source=altibase_odbc;
```

This connection string can also be used to establish an OLE DB connection with ALTIBASE HDB via ADO or ADO.net, as explained below:

5.2.3.2 ADO Connection Strings

This section contains sample ADO connection strings used for connecting to ALTIBASE HDB via ADO. As explained above, the connection string can be modified so that it simply refers to a DSN rather than containing a large number of keyword/value pairs. In this example, altibase_odbc is used as the name of the DSN containing the connection information.

As above, the data provider must be set to Altibase.OLEDB.

An ADO connection string containing this information looks something like this:

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

If a udl file exists, it can be used to open an ADO connection. To accomplish this, modify the connection string to specify the name of the udl file using the "File Name" keyword.

```
Dim con As New ADODB.Connection
con.ConnectionString = "File Name=conn_test.udl"
con.Open
```

5.2.3.3 ADO.net Connection Strings

A sample ADO.net connection string used for connecting to ALTIBASE HDB via ADO.net is shown below.

```
using System.Data.OleDb // Using OLE DB
OleDbConnection connection = new OleDbConnection();
connection.ConnectionString = "Provider=Altibase.OLEDB;Data
Source=127.0.0.1;User ID=sys;Password=manager;Extended Proper-
ties=\"PORT=20300\"";
connection.Open(); // Opening an ADO.net connection
```

5.2.4 ALTIBASE HDB and OLE DB Data Types

The following table shows the OLE DB data types to which data types of ALTIBASE HDB are mapped when using OLE DB to access a data source of ALTIBASE HDB:

ALTIBASE HDB	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
NUMERIC	DBTYPE_STR
DECIMAL	DBTYPE_STR
REAL	DBTYPE_R4
SMALLINT	DBTYPE_I2
VARBIT	DBTYPE_BOOL
VARCHAR	DBTYPE_STR

- The ALTIBASE HDB data types BLOB, CLOB, GEOMETRY, and NIBBLE have no corresponding OLE DB types, and thus cannot be accessed via OLE DB.
- The BIT and VARBIT ALTIBASE HDB types are converted to the BOOL type in OLE DB.

The following table shows the ALTIBASE HDB data types to which OLE DB data types are mapped when using OLE DB to write to an Altibase database:

OLE DB	ALTIBASE HDB
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

Note: To use a character string constant in a SQL statement when working with a national character set, add the character "N" in front of the character string.

5.3 Properties

This section lists the properties in each property set that are currently supported by ALTIBASE HDB.

Property Set	Description
DBPROPSET_DATASOURCEINFO	These properties constitute a set of static information about the data source.
DBPROPSET_DBINIT	These properties are used to initialize the data source.
DBPROPSET_OGIS_SPATIAL_OPS	These properties describe OpenGIS spatial data.
DBPROPSET_ROWSET	These properties describe rowsets.
DBPROPSET_SESSION	These properties describe sessions.

The specific properties within each property set that are currently supported by ALTIBASE HDB are listed in the tables that follow.

5.3.1 DBPROPSET_DATASOURCEINFO

The following table summarizes the OLE DB properties in the DBPROPSET_ DATASOURCEINFO property set that are currently supported by ALTIBASE HDB.

Property	State	Initial Value
DBPROP_ACTIVESESSIONS	Read-Only	0
DBPROP_BYREFACCESSORS	Read-Only	VARIANT_FALSE
DBPROP_CONNECTIONSTATUS	Read-Only	DBPROPVAL_CS_COMMUNICATIONFAILURE
DBPROP_DATASOURCENAME	Read-Only	" ", This should be set at run time.
DBPROP_DATASOURCEREADONLY	Read-Only	VARIANT_TRUE
DBPROP_DBMSNAME	Read-Only	" ", This should be set at run time.
DBPROP_DBMSVER	Read-Only	" ", This should be set at run time.
DBPROP_DSOTHREADMODEL	Read-Only	DBPROPVAL_RT_FREETHREAD
DBPROP_MAXTABLESINSELECT	Read-Only	32
DBPROP_MULTIPLERESULTS	Read-Only	DBPROPVAL_MR_SUPPORTED
DBPROP_MULTIPLESTORAGEOBJECTS	Read-Only	VARIANT_TRUE
DBPROP_OLEOBJECTS	Read-Only	DBPROPVAL_OO_BLOB | DBPROPVAL_OO_SINGLETON

Property	State	Initial Value
DBPROP_OUTPUTPARAMETERAVAILABILITY	Read-Only	DBPROPVAL_OA_ATEXECUTE
DBPROP_PROVIDERFRIENDLYNAME	Read-Only	" ", This should be set at run time.
DBPROP_PROVIDEROLEDBVER	Read-Only	" ", This should be set at run time.
DBPROP_STRUCTUREDSTORAGE	Read-Only	DBPROPVAL_SS_ISEQUENTIALSTREAM
DBPROP_SUPPORTEDTXNISOLEVELS	Read-Only	DBPROPVAL_TI_READCOMMITED
DBPROP_USERNAME	Read-Only	" ", This should be set at run time.
DBPROP_SQLSUPPORT	Read-Only	DBPROPVAL_SQL_ODBC_MINIMUM | DBPROPVAL_SQL_ANSI92_ENTRY | DBPROPVAL_SQL_ESCAPECLAUSES

5.3.2 DBPROPSET_DBINIT

The following table summarizes the OLE DB properties in the DBPROPSET_DBINIT property set that are currently supported by ALTIBASE HDB.

Property	State	Initial Value
DBPROP_AUTH_PASSWORD	Read / Write	" "
DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO	Read / Write	VARIANT_FALSE
DBPROP_AUTH_USERID	Read / Write	" "
DBPROP_INIT_DATASOURCE	Read / Write	" "
DBPROP_INIT_HWND	Read / Write	NULL
DBPROP_INIT_OLEDBSERVICES	Read / Write	DBPROPVAL_OS_ENABLEALL
DBPROP_INIT_PROMPT	Read / Write	DBPROMPT_NOPROMPT
DBPROP_INIT_PROVIDERSTRING	Read / Write	" "
DBPROP_INIT_TIMEOUT	Read / Write	3

5.3.3 DBPROPSET_OGIS_SPATIAL_OPS

The following table summarizes the OLE DB properties in the DBPROPSET_OGIS_SPATIAL_OPS property set that are currently supported by ALTIBASE HDB.

Property	State	Initial Value
DBPROP_OGIS_TOUCHES	Read-Only	VARIANT_TRUE
DBPROP_OGIS_WITHIN	Read-Only	VARIANT_TRUE
DBPROP_OGIS_CONTAINS	Read-Only	VARIANT_TRUE
DBPROP_OGIS_CROSSES	Read-Only	VARIANT_TRUE
DBPROP_OGIS_OVERLAPS	Read-Only	VARIANT_TRUE
DBPROP_OGIS_DISJOINT	Read-Only	VARIANT_TRUE
DBPROP_OGIS_INTERSECTS	Read-Only	VARIANT_TRUE
DBPROP_OGIS_ENVELOPE_INTERSECTS	Read-Only	VARIANT_TRUE
DBPROP_OGIS_INDEX_INTERSECTS	Read-Only	VARIANT_TRUE

5.3.4 DBPROPSET_ROWSET

The following table summarizes the OLE DB properties in the DBPROPSET_ROWSET property set that are currently supported by ALTIBASE HDB.

Property	State	Initial Value
DBPROP_ACCESSORDER	Read-Only	DBPROPVAL_AO_RANDOM
DBPROP_BLOCKINGSTORAGEOBJECTS	Read-Only	VARIANT_TRUE
DBPROP_CANHOLDROWS	Read-Only	VARIANT_FALSE
DBPROP_CANFETCHBACKWARDS	Read-Only	VARIANT_FALSE
DBPROP_CANSCROLLBACKWARDS	Read-Only	VARIANT_FALSE
DBPROP_CLIENTCURSOR	Read-Only	VARIANT_TRUE
DBPROP_COMMANDTIMEOUT	Read-Only	30
DBPROP_IAccessor	Read-Only	VARIANT_TRUE
DBPROP_IColumnsInfo	Read-Only	VARIANT_TRUE
DBPROP_IColumnsRowset	Read-Only	VARIANT_TRUE
DBPROP_IConvertType	Read-Only	VARIANT_TRUE
DBPROP_IGetSession	Read-Only	VARIANT_TRUE
DBPROP_IMultipleResults	Read-Only	VARIANT_FALSE
DBPROP_IRowset	Read-Only	VARIANT_TRUE
DBPROP_IRowsetChange	Read-Only	VARIANT_FALSE

Property	State	Initial Value
DBPROP_IRowsetIdentity	Read-Only	VARIANT_TRUE
DBPROP_IRowsetInfo	Read-Only	VARIANT_TRUE
DBPROP_ISequentialStream	Read-Only	VARIANT_TRUE
DBPROP_ISupportErrorInfo	Read-Only	VARIANT_TRUE
DBPROP_LITERALIDENTITY	Read-Only	VARIANT_TRUE
DBPROP_MAXROWS	Read-Only	0
DBPROP_OTHERINSERT	Read-Only	VARIANT_FALSE
DBPROP_OTHERUPDATEDELETE	Read-Only	VARIANT_FALSE
DBPROP_OWNINSERT	Read-Only	VARIANT_FALSE
DBPROP_OWNUPDATEDELETE	Read-Only	VARIANT_FALSE
DBPROP_REMOVEDELETED	Read-Only	VARIANT_FALSE
DBPROP_SERVERCURSOR	Read-Only	VARIANT_FALSE
DBPROP_UPDATABILITY	Read-Only	VARIANT_FALSE

5.3.5 DBPROPSET_SESSION

The following tables summarizes the OLE DB properties in the DBPROPSET_SESSION property set that are currently supported by ALTIBASE HDB.

Property	State	Initial Value
DBPROP_SESS_AUTOCOMMITISOLEVELS	Read-Only	ISOLATIONLEVEL_READCOMMITTED

5.4 The OLE DB Interfaces

This sections discusses the use of objects and their interfaces. The following objects are the OLE DB interfaces that are currently supported by ALTIBASE HDB:

- Data Source Object
- Session Object
- Command Object
- Multiple Results Object
- Rowset Object
- Row Object
- Error Object
- Custom Error Object

5.4.1 The Data Source Object

The data source object is necessary in order to connect to an Altibase database. The data source object cotype is defined as follows:

```
CoType TDataSource {
 interface IDBCreateSession;
 interface IDBInitialize;
 interface IDBProperties;
 interface IPersist;
 interface IDBInfo;
 interface ISupportErrorInfo;
}
```

5.4.2 The Session Object

The session object cotype is defined as follows:

```
CoType TSession {
 interface IDBGetDataSource;
 interface IOpenRowset;
 interface ISessionProperties;
 interface IDBCreateCommand;
 interface IDBSchemaRowset;
 interface ISupportErrorInfo;
 interface ITransactionLocal;
}
```

5.4.3 The Command Object

The command object cotype is defined as follows:

```
CoType TCommand {
```

```
 interface IAccessor;
 interface IColumnsInfo;
 interface ICommandPrepare;
 interface ICommandProperties;
 interface ICommandText;
 interface ICommandWithParameters;
 interface IConvertType;
 interface ISupportErrorInfo;
}
```

5.4.4 The Multiple Results Object

The multiple results object cotype is defined as follows:

```
CoType TMultipleResults {
 interface IMultipleResults;
 interface ISupportErrorInfo;
}
```

5.4.5 The Rowset Object

The rowset object cotype is defined as follows:

```
CoType TRowset {
 interface IAccessor;
 interface IColumnsInfo;
 interface IColumnsRowset;
 interface IConverType;
 interface IRowset;
 interface IRowsetChange;
 interface IRowsetInfo;
 interface IRowsetIdentity;
 interface IRowsetLocate;
 interface IRowsetRefresh;
 interface IRowsetScroll;
 interface IRowsetUpdate;
 interface ISupportErrorInfo;
}
```

5.4.6 The Row Object

The row object cotype is defined as follows:

```
CoType TRow {
 interface IColumnsInfo;
 interface IConvertType;
 interface IGetSession;
 interface IRow;
 interface ISupportErrorInfo;
}
```

5.4.7 The Error Object

The error object cotype is defined as follows:

```
CoType TError {
 interface IErrorInfo;
}
```

5.4.8 The Custom Error Object

The custom error object cotype is defined as follows:

```
CoType TCustomError {
 interface ISQLErrorInfo;
}
```

5.5 Examples

5.5.1 Connection String

The following connection string establishes a connection to ALTIBASE HDB from ADO or ADO.NET applications.

```
Server=127.0.0.1;PORT=20091;User=sys;Password=manager
```

5.5.2 ADO Visual Basic Example

```
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;Data Source=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
```

5.5.3 ADO.NET C# Example

```
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 specifies the connection string
            sConnection = new OleDbConnection("Provider=Altibase.OLEDB;Data
Source=" + args[0]);

            try
            {
                sConnection.Open(); // This creates a connection to the DB
                Console.WriteLine("Connection Success");
                sCommand = sConnection.CreateCommand();
                Console.WriteLine("CreateCommand Success");

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

                // This outputs data
                while (sDataReader.Read())
                {
                    // This iterates through the columns in the result set
                    for (int i = 0; i < sDataReader.FieldCount; i++)
                    {
                        Console.WriteLine("GetDataTypeName : " + sDa-
taReader.GetDataTypeName(i));
                        Console.WriteLine("IsDBNull : " + sDataReader.IsDB-
Null(i));
                        Console.WriteLine("Value : " + sDataReader.Get-
Value(i));
                    }
                }
                sDataReader.Close();

                // This closes the connection to the DB
                sConnection.Close();
                Console.WriteLine("Close Success");
            }
            catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

5.5.4 ADO.NET C++ Code Snippet for Calling Stored Procedure

```
string szConnectionString = "Provider=Altibase.OLEDB;Data
Source=127.0.0.1;User ID=sys;Password=manager;Extended Proper-
ties='PORT=20300;NLS_USE=KO16KSC5601'";

OleDbConnection conn = new OleDbConnection(szConnectionString);
conn.Open();

string szQuery = "upT1";
```

```
OleDbCommand comm = new OleDbCommand(szQuery, conn);
comm.CommandType = CommandType.StoredProcedure;

OleDbParameter param1 = comm.Parameters.Add("@p_Seq", OleDbType.Integer);
param1.Value = 4;

comm.ExecuteNonQuery();

conn.Close();
conn.Dispose();
```

6 XA Interface

This chapter explains the general concept of distributed transactions, introduces the XA standard, and describes the XA interface. It explains how to use a global transaction manager to access ALTIBASE HDB via ODBC, JDBC and APRE, and specifies the support for the XA within ALTIBASE HDB. It also describes the limitations of the XA distributed transaction processing model and how to deal with problems that can arise in applications.

6.1 XA Interface

XA is a standard interface that is used for processing distributed transactions (also known as “global transactions”). It was proposed by The Open Group (formerly X/Open).

A distributed transaction (also known as a “global transaction”) is a transaction that spans two or more nodes connected via a network. The database systems provide the resources for the transaction, while a TM (Transaction Manager) creates and manages the global transaction, which oversees all operations performed on these resources. The XA standard thus enables distributed applications to share resources provided by multiple database servers, and makes global transactions possible.

6.1.1 XA Glossary

- Application (AP)

  An application defines the necessary transactions and the operations that constitute a transaction. An application can be written using, for example, the precompiler or the ODBC CLI.

- Global Transaction

  This refers to multiple transactions that are managed as a single transaction by a TM. It is essentially a distributed transaction.

- Heuristic Completion

  In some situations, when an RM does not receive an expected command pertaining to an in-doubt transaction, such as a COMMIT command or the like, the RM proceeds to commit or roll back the transaction of its own accord. Completion of transactions in this way is referred to as “Heuristic Commit” or “Heuristic Rollback”, or collectively as “Heuristic Completion”. Typical causes are network failure and transaction timeouts.

- In-doubt Transaction

  An “in-doubt transaction” is a transaction branch that has been prepared on an RM (i.e. DBMS) and for which a commit or rollback message has not yet been received. It is also known as a “pending transaction”.

- Resource Manager (RM)

  A Resource Manager (RM) controls a resource that is accessed by an XA transaction. It must be possible to restore the resource to its original state in the event of a failure. An RM can be, for example, a relational database, a transactional queue, or a file system.

- Transaction Branch

  A transaction branch is essentially a sub-transaction that is part of the global transaction. It is executed on one of the Resource Managers (see above) participating in the global transaction. There is a one-to-one relationship between a transaction branch and a so-called “XID” (i.e. a Transaction ID in XA parlance).

- Transaction Manager (TM)

  A transaction manager provides an API that defines a transaction. It is responsible for committing and rolling back transactions and performing recovery. The TM has a two-phase commit

engine to ensure that all of the RMs are consistent with each other.

- Transaction Processing Monitor (TPM)

  A Transaction Processing Monitor (TPM) coordinates the flow of transaction requests from one or more APs (see above) for resources managed by one or more RMs (see above). The RMs can be heterogeneous, and can be distributed across a network.

  The TPM completes distributed transactions by coordinating commit and rollback operations. The TM portion of the TPM is responsible for determining the timing of distributed commit and rollback operations, that is, the TPM is responsible for controlling two-phase commit.

  Because the TM manages distributed commit and rollback operations, it must be aware of, and able to communicate directly with, all RMs. The TM uses the XA interface for this. In the case of ALTIBASE HDB, the TM uses XA library functions of ALTIBASE HDB to control transaction processing by ALTIBASE HDB.

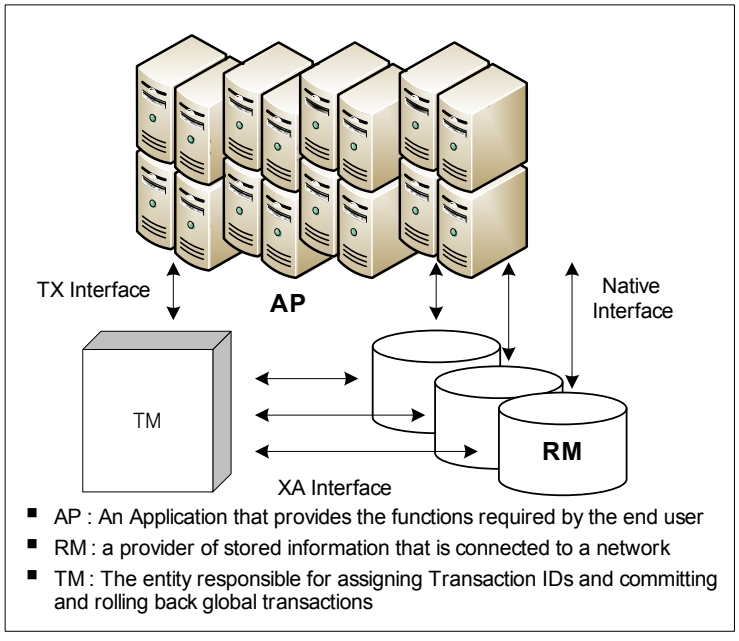
- TX Interface

  An AP (see above) controls a transaction through the TM using the TX interface. An AP does not use the XA interface directly. APs are not aware of the operations of individual transaction branches, and application threads do not participate directly in transaction branch tasks. The branches of a global transaction are managed on behalf of APs by the TM. APs merely request the TM to commit or roll back entire global transactions.

6.1.2 XA Structure

As shown in the following diagram, the entities involved in a distributed transaction include one or more APs (Applications), the TM (Transaction Manager), and one or more RMs (Resource Managers).

Figure 6-1 XA Structure



If an AP announces the start of a distributed transaction to the TM using TX interface, the TM determines which RMs (databases) are involved in the distributed transaction. The TM internally generates XIDs to identify the transaction branches that are to be executed in respective RMs, and then calls XA interface with the XIDs to the RMs.

The RMs (i.e. database nodes) then start to process the transaction branches corresponding to the respective XIDs.

To terminate the transaction, the AP calls the TM via the TX interface. The TM then uses the XA interface to instruct the RMs on which the branches of the distributed transaction are running to either commit or roll back their respective transaction branches.

6.1.3 XA and 2PC (Two-Phase Commit)

The XA Interface of ALTIBASE HDB supports 2PC (Two-Phase Commit) transaction processing. Two-phase commit consists of separate prepare and commit steps.

In the prepare step, which is the first step of 2PC, the TM queries all database nodes (RMs) participating in a distributed transaction to determine whether it is possible to commit the transaction. If an individual RM is able to commit the transaction branch that has been assigned to it, it sends a message to the TM indicating that it is in a “prepare” state. If, however, an RM is not able to commit its transaction branch, it sends a corresponding message to the TM so that the transaction can be rolled back.

In the commit step, which is the second step of 2PC, the TM waits until it has received “prepare” acknowledgements from all RMs. If it receives such acknowledgements from all RMs, it sends an instruction to all RMs to commit the transaction. However, if there is even one RM that has not sent a “prepare” acknowledgement, the TM sends an instruction to all RMs to roll back the transaction.

6.1.4 xa_switch_t Structure

Every RM has a switch that contains various information about the RM, including its entry points. This information is used by the TM. The structure of an RM's switch is known as `xa_switch_t`.

In ALTIBASE HDB, the name of the `xa_switch_t` is `altibase_xa_switch`. Its structure is as follows:

```
struct xa_switch_t {
 char name[RMNAMESZ];        /* name of resource manager */
 long flags;                 /* resource manager specific options */
 long version;               /* must be 0 */

 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*/
};
```

6.1.5 The XA Library

No additional library is required in order for applications to connect to ALTIBASE HDB using the ALTIBASE HDB XA. The required functionality is included in the odbccli library. All that is needed in order to use the XA-related functionality with ALTIBASE HDB is to link the XA-dependent applications with the libodbccli.a library file.

6.2 The XA Interface

The XA Interface is the two-way interface that sits between the TM and the RMs.

This interface consists of `xa_` routines, which the TM uses to control RMs so that it can execute global transactions, and `ax_` routines, which allow the RMs to make requests to the TM.

Note: *Because ALTIBASE HDB does not support dynamic registration, each RM (Altibase database) must be called with xa_start before the start of a transaction.*

6.2.1 XA Functions

In ALTIBASE HDB, the XA-related functions are provided in `altibase_xa_switch`, which is ALTIBASE HDB's implementation of `xa_switch_t`.

Table 6-1 XA Interface

XA Interface	Description
xa_open	This is used to connect to an RM.
xa_close	This is used to close a connection with an RM.
xa_start	This is used to start a new transaction branch or restart an existing one, and to link the branch to a given XID.
xa_end	This is used to end an association with a transaction branch.
xa_rollback	This is used to roll back a transaction branch corresponding to a given XID.
xa_prepare	This is used to prepare a transaction branch to be committed.
xa_commit	This is used to commit a transaction branch.
xa_recover	This is used to show a list of XIDs corresponding to transactions that have been prepared, heuristically committed, or heuristically rolled back.
xa_forget	This is used to instruct an RM to discard information about a heuristically completed transaction branch.

6.2.1.1 xa_open

This is used to connect to an RM.

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

`xa_info` is a null-terminated character string that contains information about the server. Its maximum length is 256 bytes. It has the same format as the parameters to the `SQLDriverConnect` function, and has the additional parameters XA_NAME and XA_LOG_DIR. For detailed information about the other parameters, please refer to the description of the `SQLDriverConnect` function in the *ODBC Reference*.

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

Table 6-2 Additional XA Interface Parameters

XA Parameter	Description
XA_NAME	This is the name that is used by the ALTIBASE HDB Precompiler to identify the connection. If this value is omitted when writing an application with the ALTIBASE HDB Precompiler, the default connection will be used. If a name is specified here, it can be used in the AT clause of a subsequently executed SQL statement. This value is specified in this way: `XA_NAME=conn1`
XA_LOG_DIR	This is used to specify the directory in which information about ALTIBASE HDB XA library errors is logged. If the $ALTIBASE_HOME environment variable has been set, then the default value of XA_LOG_DIR is $ALTIBASE_HOME/trc. If $ALTIBASE_HOME has not been set, the default is the current directory.

`rmid` is used to specify an identifier for the server to be accessed. This can be set to any arbitrary value.

If `flags` is not set to any other value, it must be set to the following value:

- TMNOFLAGS

6.2.1.2 xa_close

This terminates the connection to the specified RM and returns XA_OK.

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

`xa_info` is a null-terminated character string that contains information about the server. Its maximum length is 256 bytes.

Note: *XA_OK is returned even if `xa_close` is executed on a connection that is already closed.*

`flags` has no specific purpose in this function, and must be set to the following value:

- TMNOFLAGS

6.2.1.3 xa_start

This is used to start the execution of a transaction branch. XID is the identifier of a global transaction.

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

`flags` can be set to one or more of the following values. In order to specify multiple flags, delimit them with a single vertical bar ("|").

- TMRESUME

  This is used to resume execution of a previously suspended transaction branch.

- TMNOWAIT

  If the execution of `xa_start` is blocked, this specifies that XA_RETRY is to be returned with-

out waiting.

- TMASYNC

  This specifies that the transaction branch is to be executed in asynchronous mode (not supported in ALTIBASE HDB).

- TMNOFLAGS

  If `flags` is not set to any other value, it must be set to this value.

- TMJOIN

  This specifies that the transaction branch is to be connected to an existing transaction branch.

6.2.1.4 xa_end

This is used to terminate the execution of a transaction branch.

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

`flags` can be set to one or more of the following values:

- TMSUSPEND

  This specifies that execution of the transaction branch is to be merely suspended, rather than permanently terminated. Execution of this transaction branch can be resumed later using `xa_start` with the TMRESUME flag.

- TMSUCCESS

  This is used to specify successful termination of a transaction branch. It can't be used together with TMSUSPEND or TMFAIL.

- TMFAIL

  This is used to specify abnormal termination of a transaction branch. The status of the transaction branch becomes “rollback only”. It can't be used together with TMSUSPEND or TMSUCCESS.

6.2.1.5 xa_rollback

This is used to roll back the operations performed by the transaction branch.

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

`flags` can be set to one of the following values:

- TMASYNC

  This specifies that the transaction branch is to be rolled back in asynchronous mode (not supported in ALTIBASE HDB).

- TMNOFLAGS

  If `flags` is not set to TMASYNC, it must be set to this value.

6.2.1.6 xa_prepare

When using the two-phase commit protocol, this is executed before committing or rolling back a transaction.

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

`flags` can be set to one of the following values:

- TMASYNC

  (not supported in ALTIBASE HDB)

- TMNOFLAGS

  If `flags` is not set to TMASYNC, it must be set to this value.

`xa_prepare` can return the following values:

- XA_RDONLY

  This is returned when the transaction doesn't change any of the data on the RM (i.e. DBMS). The transaction does not need to be committed or rolled back.

- XA_OK

  This is returned when the prepare task is performed normally.

6.2.1.7 xa_commit

This is used to commit a particular transaction branch.

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

`flags` can be set to either of the following values:

- TMONEPHASE

  This is set to specify one-phase commit.

- TMNOFLAGS

  If `flags` is not set to any other value, it must be set to this value.

6.2.1.8 xa_recover

This obtains a list of the XIDs corresponding to branch transactions that are in a prepared state on an ALTIBASE HDB server.

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

The return value indicates the number of XIDs that were recovered. The count parameter is used to set the maximum number of XIDs that fit into the *xids* array.

`flags` can be set to one or more of the following values:

- TMSTARTRSCAN

  For more information, please refer to the XA Specification documentation.

- TMENDRSCAN

  For more information, please refer to the XA Specification documentation.

- TMNOFLAGS

  XIDs are returned starting at the current cursor position.

6.2.1.9 xa_forget

This instructs the ALTIBASE HDB server (i.e. the RM) to stop managing a heuristically completed transaction branch.

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

`flags` has no specific purpose in this function, and must be set to the following value:

- TMNOFLAGS

6.2.1.10 xa_complete

When operating in asynchronous mode, this is used to determine whether to keep waiting for an operation to terminate. This is not supported in ALTIBASE HDB, and thus an error will always be returned.

6.3 Using XA

This section describes the basic procedures for using ODBC, APRE and JDBC in an XA environment.

6.3.1 Executing ODBC/XA

- xa_open

  This is used to connect to the specified server.

- SQLAllocHandle

  This is used to create the connection and environment handles for connecting via ODBC.

- SQLSetConnectAttr

  This is used to associate the connection handle with the XA connection.

- SQLConnect

  `SQLConnect` does not actually establish a physical connection between the TM and the RM, because that is accomplished by calling `xa_open`. `SQLConnect` merely changes the internal state of the connection within ODBC. This is a necessary step in order to be able to perform DML operations using ODBC.

- xa_start

  This is used to commence execution of the transaction branch corresponding to the given XID.

- executing SQL statements

  Here, operations such as `SQLPrepare` and `SQLExecute` are performed. If an attempt is made to execute a commit statement here, the server will return an error message.

- xa_end

  This is used to terminate execution of a transaction branch.

- xa_prepare

  This is used to prepare a transaction branch for commit.

- xa_commit

  This is used to commit a transaction.

- SQLDisconnect

  This is used to switch the internal state to disconnected. However, the physical connection established by XA remains active.

- xa_close

  This is used to close the XA connection.

6.3.1.1 SQLSetConnectAttr

Calling SQLSetConnectAttr enables an XA connection to use an ODBC connection, so that an application can access a distributed transaction via ODBC.

The following parameters are provided to enable an XA connection to be configured using SQLSetConnectAttr:

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

- fAttr=ALTIBASE_XA_RMID

  Setting the fAttr parameter to ALTIBASE_XA_RMID enables the connection specified using the hdbc parameter to use a specified XA connection. Detailed information about the XA connection is set by specifying a pointer for the vParam parameter, which is described below.

- vParam

  This must be set to the rmid value that was specified when a connection was established using xa_open.

  To establish an XA connection with a server without specifying a value for rmid, use one of the following settings for the fAttr parameter:

  fAttr=SQL_ATTR_ENLIST_IN_DTC or SQL_ATTR_ENLIST_IN_XA

  Setting the fAttr parameter to SQL_ATTR_ENLIST_IN_DTC or SQL_ATTR_ENLIST_IN_XA associates the current database connection with the last XA connection.

6.3.2 Executing APRE/XA

6.3.2.1 How to Author an Application depending on the Setting of XA_NAME in xa_open

In XA applications, a cursor is valid only for a single transaction. This means that a cursor must be opened after the start of execution of a transaction, and must be closed before the transaction is completed (i.e. committed or rolled back).

How to Author an Application when Using the Default Connection

If it is desired to use the default connection, the XA_NAME keyword must not be present in xa_info, which is the character string parameter of xa_open that contains the connection information. An example of xa_info without XA_NAME is shown below:

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

It is therefore not possible to use the AT clause when executing SQL queries. The following query is acceptable because it does not contain an AT clause:

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

How to Author an Application when Using XA_NAME to Specify One or More Connections

If it is desired to specify a connection when using APRE to author an application, the XA_NAME keyword and a corresponding value must be present in the `xa_info` connection character string parameter of `xa_open`. An example of a valid keyword/value pair is shown below:

```
XA_NAME=conn1
```

It is possible to write an application that uses a default connection and one or more additional connections specified using XA_NAME. This is accomplished as shown below.

If, for example, the names of the connections specified using XA_NAME are conn1 and conn2, the value of open_string in the TM (Transaction Manager) environment settings would be as follows:

```
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
```

This permits the application to execute SQL statements that contain the AT clause, thereby accessing multiple servers, as shown below:

```
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;
```

6.3.3 Executing JDBC/XA

The XA classes that are defined by the jdbc driver of ALTIBASE HDB are as shown below:

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

The ABXADataSource class is the only one that the user accesses directly. The user does not need to directly access the other classes, as they are implemented in the JTA interface class.

1. Create an ABXADataSource Object

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

2. Create an XAConnection Object

   Create an XAConnection object by calling the `getXAConnection` method in the XADataSource class.

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

3. Create an XAResource Object

   Create an XAResource object by calling the `getXAResource` method in the XAConnection class.

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

4. Create a Connection Object

Create a connection object to use for executing SQL statements by calling the `getConnection` method in the XAConnection class.

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

5. Use the XAResource Object to Execute XA Functions

   XA functions such as `xa_start` and `xa_end` can be executed using the methods in the XAResource class.

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

6. Execute SQL Statements using the Connection Object

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

6.3.4 XA Transaction Control

This section describes how to control transactions in an ALTIBASE HDB XA environment.

When using the XA library, the SQL COMMIT and ROLLBACK statements are not used to commit and roll back transactions. Instead, the users must use the TX interface that is provided by the TM in application programs, as shown below.

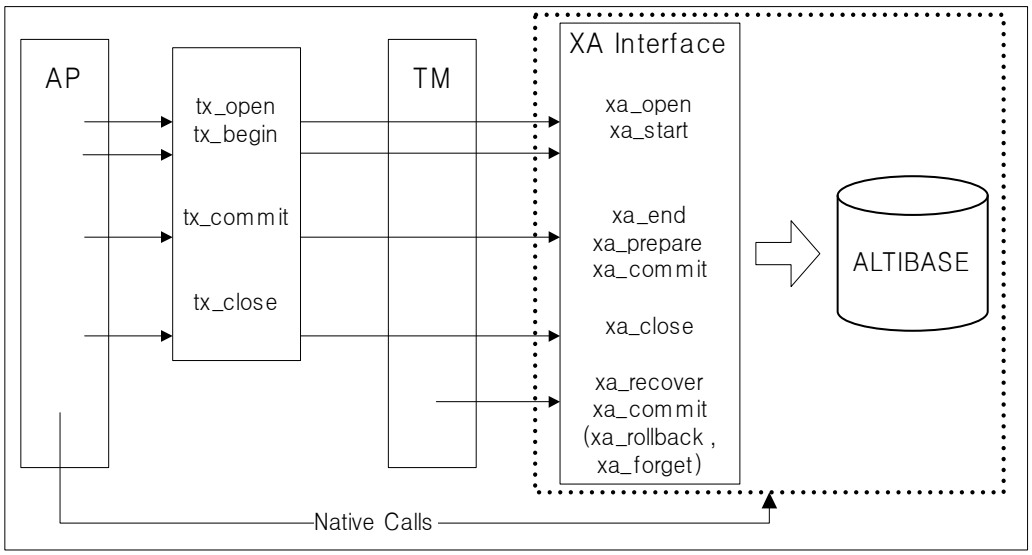
The TM typically controls a transaction using the XA interface.

Table 6-3 The TX Interface

TX Interface Entry Point	Description
tx_open	This logs on to an RM.
tx_close	This logs off from an RM.
tx_begin	This starts execution of a new transaction.
tx_commit	This commits a transaction.
tx_rollback	This rolls back a transaction.

The process of calling the TX and XA interfaces is as shown in the following diagram:

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



A TPM (Transaction Processing Monitor) application has a client/server structure in which a client requests a service provided by an application server. Service is divided into logical work units. When ALTIBASE HDB is used as the RM, a logical work unit typically consists of a set of SQL statements.

6.3.4.1 Example

In the following example, it is assumed that the application server has already logged on to the TPM system.

Starting a Transaction on an Application Server

The following example shows the start of a transaction on an application server.

```
Ex) Client:
tpm_service("SERVICE1");

Ex) Server:
SERVICE1()
{
  <get service specific data>
  tx_begin();
  EXEC SQL UPDATE....;

  tpm_service("SERVICE2");
  tx_commit();
  <return service status back to the client>
}
```

Starting a Transaction on a Client

The following example shows the start of a transaction on a client.

```
Ex) Client:
tx_begin();
```

```
tpm_service("SERVICE1");
tmp_service("SERVICE2");
tx_commit();

Ex) 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>
}
```

6.3.5 Changing an Existing Application into a TPM Application

To change an existing application (Precompiler or ODBCCLI application) into a TPM (Transaction Processing Monitor) application that uses the XA library of ALTIBASE HDB, follow the procedure outlined below:

1. Convert the application into one that incorporates a "service" framework.

   Here, the term "framework" means one in which a client requests a service from an application server. In some TPMs, the `tx_open` and `tx_close` functions must be explicitly called, while in other TPMs, the logging on and off takes place implicitly.

2. General connection statements must be changed into a TPM-compatible form. For example, when working with APRE, replace the `EXEC SQL CONNECT` statement with a call to `tx_open`, and when working with ODBCCLI, replace the `SQLDriverConnect` statement with calls to both `tx_open` and `SQLConnect`.

   The ODBCCLI `SQLDriverConnect` statement must be replaced by both `tx_open` and `SQLConnect`. Although the actual connection is achieved using `tx_open`, the `SQLConnect` task is necessary in order for it to be possible to make a connection internally within ODBC. For more detailed information, please refer to 6.3.1 Executing ODBC/XA, which outlines the required tasks in sequence.

3. Disconnection statements must also be changed into a TPM-compatible form. Replace the `EXEC SQL DISCONNECT` statement (when working with APRE) or the `SQLDisconnect` statement (when working with ODBCCLI) with a call to `tx_close`.

4. Commit and rollback statements must also be changed into a TPM-compatible form. When working with APRE, replace the `EXEC SQL COMMIT` statement with a call to `tx_commit` and the `EXEC SQL ROLLBACK` statement with a call to `tx_rollback`. When working with ODBCCLI, replace the `SQLEndTran` statement with a call to either `tx_commit` or `tx_rollback`, as appropriate. Use `tx_begin` to initiate the execution of a transaction.

5. Before terminating a transaction, the application must exit the state in which it is ready to fetch records. That is, after fetching data using a cursor and before ending the transaction, the `CLOSE RELEASE` statement must be used to close the cursor and free all associated resources.

ALTIBASE HDB Statement	TPM Functions
CONNECT	tx_open
Implicit commencement of transaction	tx_begin
Executing SQL statements	Service that executes the SQL statements
COMMIT	tx_commit
ROLLBACK	tx_rollback
DISCONNECT	tx_close
SET TRANSACTION READ ONLY	Not allowed

6.4 Limitations when using XA

The use of XA is limited in the following ways:

- Limitations on Use of SQL
- Limitations related to Transaction Branches
- No Support for Association Migration
- No Support for Asynchronous Calls
- No Support for Dynamic Registration

6.4.1 Limitations on Use of SQL

6.4.1.1 Rollback and Commit

Because global transactions are managed by the TM, the ALTIBASE HDB transaction control statements `COMMIT` and `ROLLBACK` must not be used within an XA application to control global transactions.

Instead, `tx_commit` and `tx_rollback` must be used to complete global transactions. This means that the EXEC SQL ROLLBACK and EXEC SQL COMMIT statements can't be used within applications authored using APRE. Similarly, `SQLEndTran` can't be used within an ODBCCLI application to commit or roll back a transaction.

6.4.1.2 DDL

Because DDL SQL statements are implicitly committed, they can't be used within XA applications of ALTIBASE HDB.

6.4.1.3 The AUTOCOMMIT Session Property

Because global transactions execute in non-autocommit mode, the AUTOCOMMIT property can't be changed using the `ALTER SESSION SET AUTOCOMMIT = TRUE` statement.

6.4.1.4 SET TRANSACTION

The ALTIBASE HDB `SET TRANSACTION { READ ONLY | READ WRITE | ISOLATION LEVEL ... }` data control statement can't be used within an XA application of ALTIBASE HDB.

6.4.1.5 Connection or Disconnection with EXEC SQL Statements

The `EXEC SQL CONNECT` and `EXEC SQL DISCONNECT` statements can't be used to establish or terminate connections in applications authored using APRE.

6.4.2 Limitations related to Transaction Branches

Multiple application threads participate in the execution of a single global transaction. These threads have either tightly-coupled or loosely-coupled relationships between them.

Threads that have a tightly-coupled relationship share a common resource. In addition, an RM handles a pair of coupled threads as a single entity. The RM must ensure that tightly-coupled threads do not reach a resource deadlock in a transaction branch. However, there is no need to provide this guarantee for loosely-coupled threads. The RM handles loosely-coupled transaction branches as though they were different global transactions.

Relationship between XID and Thread

If the TM assigns a new XID (branch qualifier) to a thread, this thread has a loosely-coupled relationship with the other threads in the same branch. The RM handles this thread as though it were a separate global transaction.

In contrast, if the TM joins a branch with an XID, that is, assigns an existing XID (branch qualifier) to a thread, the thread has a tightly-coupled relationship with the other threads sharing this branch. The RM regards tightly-coupled threads as one object, and must guarantee that a resource deadlock does not occur between tightly-coupled threads.

6.4.3 No Support for Association Migration

Association migration (in which the TM associates a suspended branch with another branch and resumes its execution) isn't supported in ALTIBASE HDB.

6.4.4 No Support for Asynchronous Calls

Asynchronous XA calls are not supported in XA applications of ALTIBASE HDB.

6.4.5 No Support for Dynamic Registration

The ALTIBASE HDB server does not support dynamic registration. Only static registration is supported. In so-called “dynamic registration”, an RM registers a global transaction with the TM before it starts executing a transaction branch.

In static registration, it is necessary to use `xa_start` to tell an RM that a transaction has commenced.

6.4.6 Server Shutdown

6.4.6.1 Abnormal Server Shutdown

Suppose that the server terminates abnormally or that the `shutdown abort` command is executed on the server, and that there are one or more transactions that are in a prepared state at that time. When the server is subsequently restarted, recovery tasks will be performed, after which it will be possible to execute these transactions using the `xa_recover` statement.

6.4.6.2 Normal Server Shutdown

If the server is shut down normally using the `shutdown immediate` or `shutdown normal` command while there are one or more transactions in a prepared state, ALTIBASE HDB aborts these transactions in order to shut down. Recovery tasks are then performed when the server is subsequently restarted, and these prepared transactions are restored to their previous state. In contrast, if there are no prepared transactions when the server is shut down normally, recovery tasks will not be performed when the server is restarted.

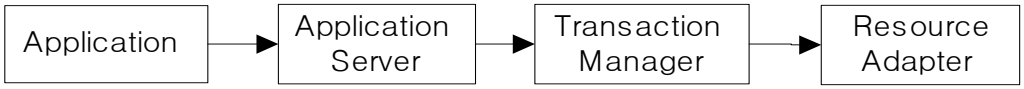
6.5 JDBC Distributed Transactions

ALTIBASE HDB JDBC can be used to implement distributed transactions, as it complies with the OpenXA standards related to connection pooling and distributed transaction processing, as set forth in the JDBC 2.0 extension API. The jdbc driver package of ALTIBASE HDB includes classes for realizing all of the distributed transaction processing functionality in accordance with the XA standard.

6.5.1 JTA (Java Transaction API) and Application Server

The method by which an application processes a distributed transaction through an application server is shown in the following figure:

Figure 6-3 Distributed Transaction Processing



The application server supports the use of XAConnections that make it possible to connect to respective resources.

An application connects to an application server, establishes a connection, and executes queries. The application server manages the transaction using the TM (Transaction Manager). The TM accesses required resources using a Resource Adapter provided by the DBMS vendor.

When the resource to which a connection is to be established using the Resource Adapter is a DBMS, the JDBC driver package can be used as the Resource Adapter. A Resource Adapter has 4 classes, namely the ResourceFactory, Transactional Resource (XAConnection), Connection, and XAResource classes.

The ResourceFactory class is used to create an XAConnection object. In the case of JDBC, the factory that is used to create XAConnection objects is XADataSource. The application server obtains an XAConnection object (for connecting to a DBMS) from the XADataSource factory. The application server then obtains an instance of a connection object (java.sql.Connection), to be used by the application, and an instance of a XAResource object, to be used by the TM, from the XAConnection object.

6.5.2 XA Components

In this section, the standard XA interfaces provided in the JDBC 2.0 Optional Package are explained, along with the ALTIBASE HDB classes in which they are implemented.

6.5.2.1 XADataSource Interface

The javax.sql.XADataSource interface is a factory for creating XAConnection objects. This interface's `getXAConnection` method returns an XAConnection object.

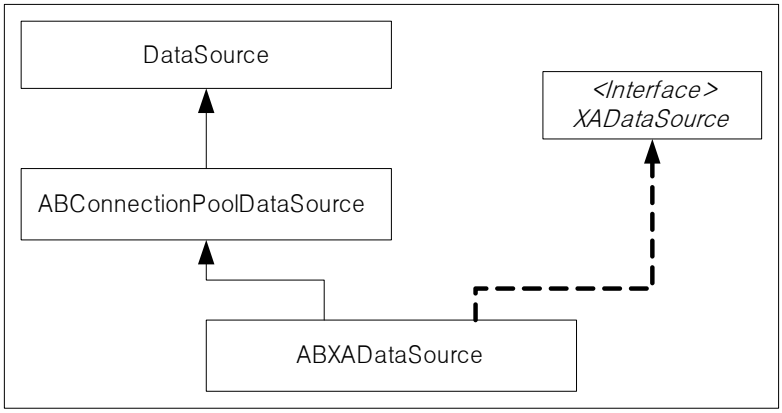
```
public interface XADataSource
{
  XAConnection getXAConnection() throws SQLException;
```

```
  XAConnection getXAConnection(String user, String password)
    throws SQLException;
 ...
}
```

Altibase.jdbc.driver.ABXADataSource is the class in which the XADataSource interface is implemented, and is included in the JDBC driver provided by ALTIBASE HDB. It is derived from the Altibase.jdbc.driver.ABConnectionPoolDataSource class. The ABConnectionPoolDataSource class is in turn derived from the Altibase.jdbc.driver.DataSource class.

Therefore, the ABXADataSource class includes all the connection properties that the DataSource and ABConnectionPoolDataSource classes have.

Figure 6-4 ABXADataSource Class



The `getXAConnection` method of the ABXADataSource class returns an instance of the XAConnection type. Because this is actually an instance of the ABPooledConnection class, the ABPooledConnection class is the implementation of the XAConnection interface.

An XA data source can be registered in the Java Naming and Directory Interface (JNDI) and used.

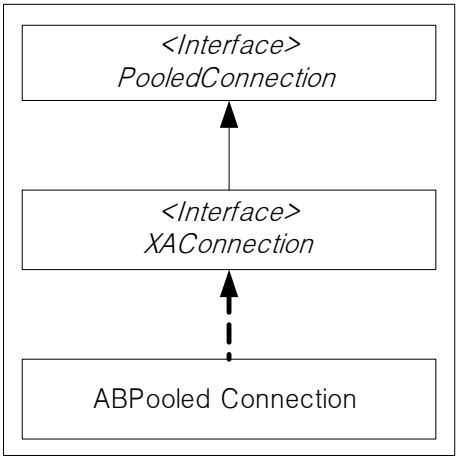
6.5.2.2 XAConnection Interface

The XAConnection interface is a child interface of the PooledConnection interface. It includes the `getConnection`, close, `addConnectionEventListener` and `removeConnectionEventListener` methods.

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

An XAConnection instance establishes a physical connection to a database. It is used to manage a distributed transaction, and to obtain an XAResource object that plays a role in managing the distributed transaction. In the JDBC driver of ALTIBASE HDB, the instance of the Altibase.jdbc.driver.ABPooledConnection class is the actual instance of the XAConnection type. The `getXAResource` method of the ABPooledConnection class returns an instance of the ABXAResource object. The `getConnection` method returns an instance of the ABConnection object.

Figure 6-5 ABPooledConnection Class



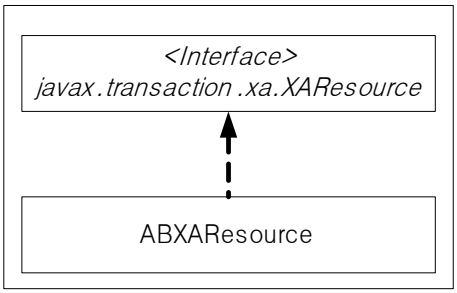
The ABConnection instance returned by the `getConnection` method acts as a temporary handle for the physical database connection. It acts like a normal connection until the transaction branch starts participating in the global transaction. At the moment that the transaction branch participates in the global transaction, AUTOCOMMIT becomes false. After the global transaction terminates, AUTOCOMMIT is restored to its original state, that is, its state prior to the start of the global transaction.

Each time an XAConnection instance's `getConnection` method is called, it returns a new instance of a Connection object. At this time, if any previous connection instance that was created by the same XAConnection instance still exists, it is closed. It is nevertheless advisable to explicitly close a previous Connection instance before opening a new one. Calling the `close` method of an XAConnection instance closes the physical connection to the database.

6.5.2.3 XAResource Interface

The TM uses instances of the ABXAResource object to coordinate all of the transaction branches. An instance of the Altibse.jdbc.driver.ABXAResource type is an instance of the class in which the javax.transaction.xa.XAResource interface is implemented.

Figure 6-6 ABXAResource Class



Whenever the `getXAResource` method of the ABPooledConnection class is called, the JDBC driver of ALTIBASE HDB creates and returns an instance of the ABXAResource class, and associates the ABXAResource instance with a Connection instance. This is the Connection object that is used by the transaction branch.

The ABXAResource class has several methods for controlling a transaction branch of a distributed transaction. A TM receives an instance of the ABXAResource class from a middle-tier component such as an application server. The ABXAResource class exposes the following methods:

```
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)
```

For more detailed information, please refer to the description of the javax.transaction.xa.XAResource class in the Java API Specifications.

6.5.2.4 The Xid interface

The TM creates instances of the Xid interface and uses them to coordinate the branches of a distributed transaction. Each transaction branch is assigned a unique transaction ID, which includes the following information:

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

In ALTIBASE HDB, the javax.transaction.xa.Xid interface is implemented as the XID class in the Altibase.jdbc.driver package.

Note: *Altibase.jdbc.driver.XID does not need to be used to make ABXAResource calls. Any class in which the javax.transaction.xa.Xid interface is implemented can be used for this.*

6.5.3 Error Handling

When errors occur, XA-related methods throw the ABXAException class. The ABXAException class is a subclass of the javax.transaction.xa.XAException class.

6.5.4 Making XA Settings in Application Servers

6.5.4.1 Making XA Settings in WebLogic

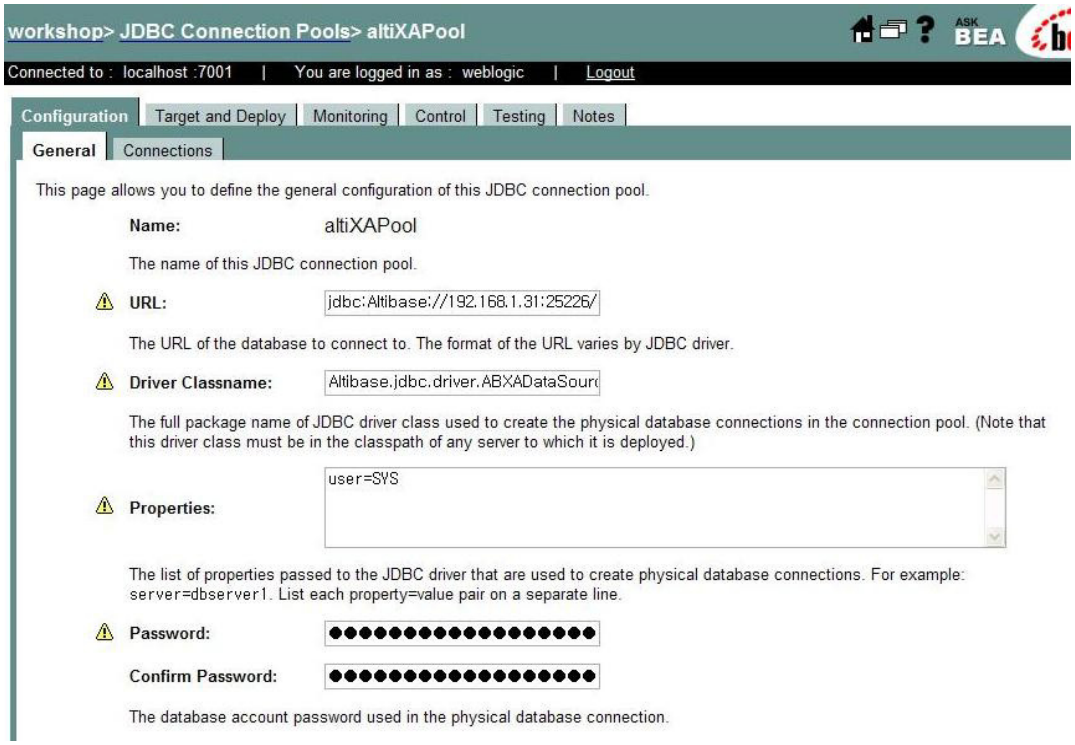
1. In the WebLogic console, expand Services -> JDBC -> Connection Pools -> Configure a new JDBC Connection Pool, and then enter the required JDBC connection information. (The required information is shown in [Figure 6-7].)

Table 6-4 Connection Information for Non-XA and XA Environments

	NON-XA	XA
URL	jdbc:Altibase://[ip]:[port]/dbname	jdbc:Altibase://[ip]:[port]/dbname

	NON-XA	XA
Driver Classname	Altibase.jdbc.driver.AltibaseDriver	Altibase.jdbc.driver.ABXADataSource
Properties	User=[username]	User=[username]

Figure 6-7 Entering JDBC Connection Information

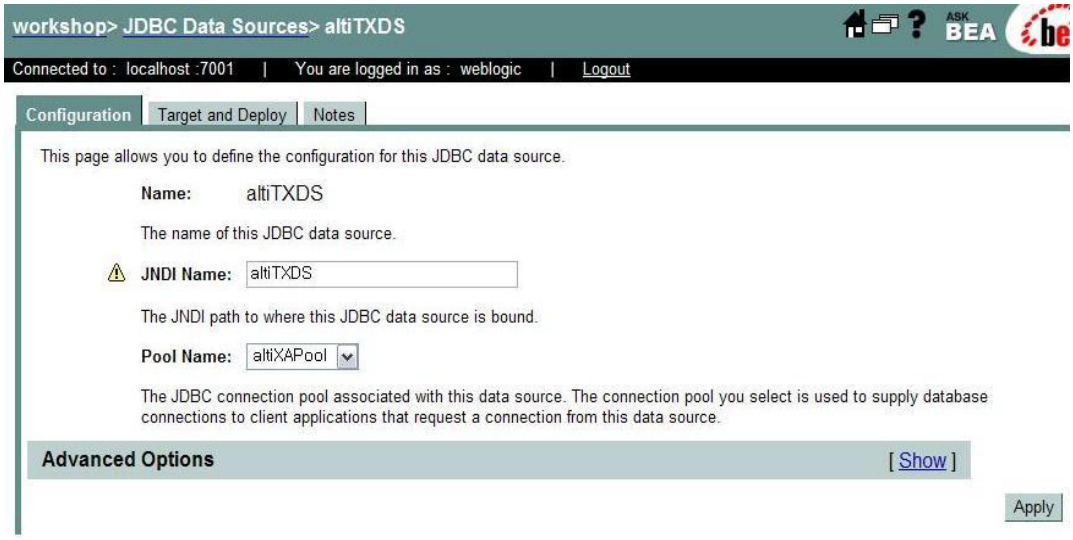


2. Create a DataSource using the newly created Connection Pool. Expand Services->JDBC->Data Sources and choose Configure a new JDBC Data Source.

   Enter the Name and JNDI Name and check “Honor Global Transactions”. For “Pool Name”, enter the name of the pool created in the first step in the window shown in [Figure 6-8].

   Note: *In versions of WebLogic prior to version 8.1, a new DataSource is created by expanding Services->JDBC->XA Data Sources.*

Figure 6-8 Creating a Data Source



6.5.4.2 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.WLInitialContextFactory");
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.UserTransaction");

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

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

6.5.4.3 Making XA Settings in JEUS

Here is how to make the basic settings to create a JDBC data source in JEUS.

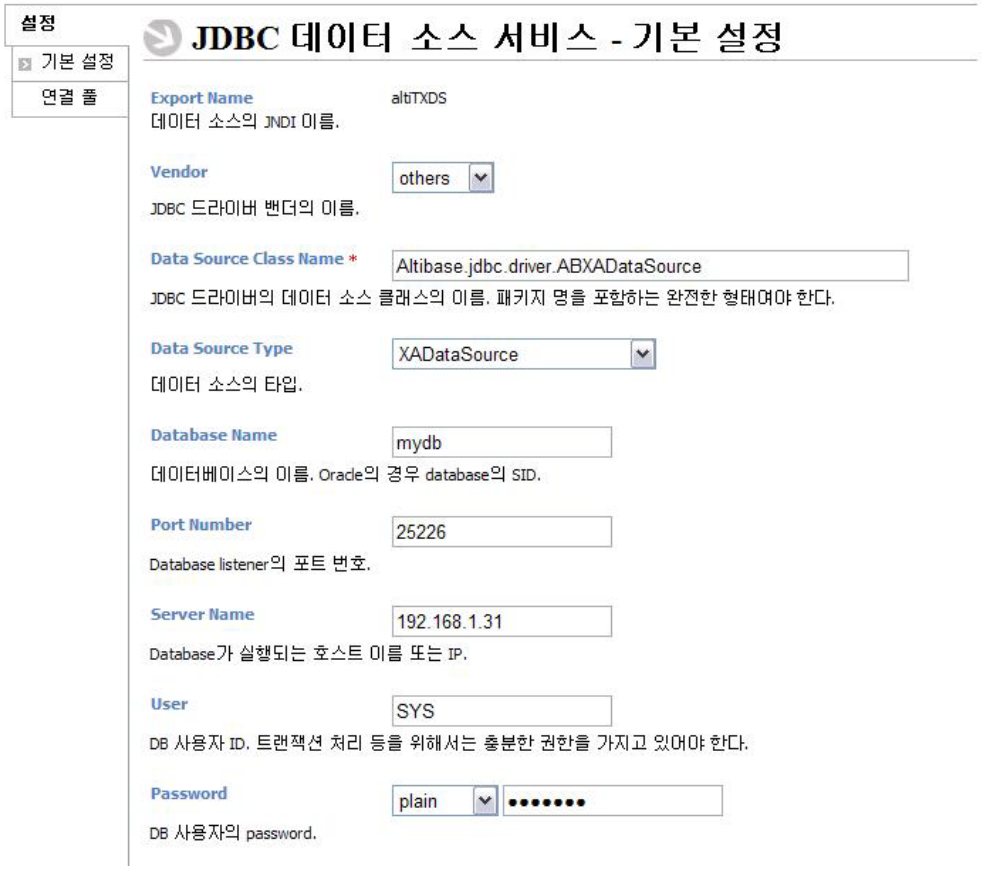
1. Under “JEUS Manager Resource(s)”, choose “JDBC” and then select “Create New JDBC Data Source”.

2. Enter the following information in the basic setup window that appears.

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

3. Enter appropriate values for Database Name, Port Number, Server Name, User and Password.

Figure 6-9 Setting a Data Source in JEUS



6.5.4.4 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/UserTransaction");
```

```
// 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");
```

6.5.5 Example

The following example illustrates how to implement distributed transactions using ALTIBASE HDB XA.

In this example, the operations are executed in the following order:

1. Start transaction branch #1.
2. Start transaction branch #2.
3. Execute DML operations on branch #1.
4. Execute DML operations on branch #2.
5. End transaction branch #1.
6. End transaction branch #2.
7. Prepare branch #1.
8. Prepare branch #2.
9. Commit branch #1.
10. 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.DataSource();
      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.DataSource();
ads2.setUrl(URL2);
ads2.setUser("SYS");
ads2.setPassword("MANAGER");
Connection connb = ads2.getConnection();

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

// Prepare a statement to create a 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 using 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();
  }
}
```

6.6 How to Solve Application Problems Using XA

This section explains how to determine the cause of any XA-related errors that may arise.

6.6.1 Checking XA Tracking Information

The XA library of ALTIBASE HDB records information that is useful for tracing errors in a trace file. If you open this file, you can check information such as error codes and messages. For example, if `xa_open` fails, checking the tracking information will help you determine whether the reason was that the `xa_info` character string has errors in it, or whether it was because the TP Manager couldn't find an ALTIBASE HDB server, or whether the attempt to log on to the ALTIBASE HDB server failed.

6.6.1.1 XA Trace File Name and Location

`altibase_xa<`*`XA_NAME`*`><`*`date`*`>.log`

- *XA_NAME* : This is the connection name specified in the `xa_info` character string field XA_NAME=value. If XA_NAME is not specified in the `xa_info` character string, it will be NULL.
- *date* : This is the date specified in the trace file (YYYYMMDD).

If the $ALTIBASE_HOME environment variable has been set, this trace file will be created in $ALTIBASE_HOME/trc. If the $ALTIBASE_HOME environment variable has not been set, the trace file will be created in the current directory.

6.6.1.2 Example

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

"104744" is the time the log was recorded (HHMISS), "19381" is the Process ID (PID), and "1" is the Resource Manager ID.

ulxXaOpen is the module name, XAER_RMERR is the XA error code, [ERR-4102E] is the error code returned by the ALTIBASE HDB server, and "invalid password" is the error message returned by the ALTIBASE HDB server.

6.6.2 Processing In-doubt Transactions

The TM is responsible for providing functionality for detecting problems that give rise to in-doubt and pending transactions and automatically completing in-doubt transactions. The RM in which the in-doubt or pending transaction is taking place maintains a lock on all resources associated with the prepared transaction until the transaction has been completed and it receives an instruction to commit the transaction.

However, if another transaction requires the data locked by an in-doubt transaction, or if a transaction remains in an in-doubt or pending status for an excessive amount of time, it will be necessary for the DBA to manually handle the transaction.

ALTIBASE HDB provides the V$DBA_2PC_PENDING performance view, which displays information about the state of in-doubt transactions so that they can be dealt with. For more information about this and other performance views, please refer to the *ALTIBASE HDB General Reference*.

To manually process such transactions, the DBA can forcefully commit or rollback transaction as shown below:

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

6.6.2.1 Example

This example shows how to check the state of in-doubt transactions and manually commit a transaction as desired.

```
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.
```

6.6.3 Checking Heuristically Completed Transactions

It is possible to check information about heuristically completed transactions. A so-called "heuristically completed transaction" is a transaction that is either committed or rolled back at the discretion of the RM after the RM fails to receive a transaction completion instruction (such as commit or rollback) from the TM for whatever reason.

If an in-doubt transaction is forcibly committed or rolled back, it is said to be a heuristically committed or heuristically rolled back transaction. Information about this transaction will be visible in the SYS_XA_HEURISTIC_TRANS_ meta table.

To delete this information, call `xa_forget` after the execution of `xa_recover`, or execute `remove_xid`().

6.6.3.1 Example

After the DBA commits an in-doubt transaction, information about the transaction is visible in the SYS_XA_HEURISTIC_TRANS_ meta table.

```
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.
```

7 The iLoader API

7.1 Overview of the iLoader API

The ALTIBASE HDB iLoader API is an application programming interface that lets you create applications that use function calls to download data from, or upload data to, an Altibase database server. Data are downloaded or uploaded in units of tables. The iLoader API provides the same functionality as the iLoader utility. For more information about the iLoader utility, please refer to the *ALTIBASE HDB iLoader User's Manual*.

The following table summarizes the functions available in the iLoader API.

Function Name	Purpose
altibase_iloader_init	Allocates an iLoader handle
altibase_iloader_final	Frees a handle and all associated resources
altibase_iloader_options_init	Initializes the option structure to its default values
altibase_iloader_formout	Creates a table format file (FORM file)
altibase_iloader_dataout	Downloads data from a table in a database and writes the data to a file
altibase_iloader_datain	Uploads data into a table in a database
CallbackFunctionName	A user-defined callback function

7.2 Using the iLoader API

7.2.1 Header Files

```
$ALTIBASE_HOME/include/iloaderApi.h
```

7.2.2 Libraries

The iLoader API library files that are required in order to develop an application that uses the iLoader API reside in the `$ALTIBASE_HOME/lib` directory. The iLoader API applications must always link with the following libraries:

- UNIX

  ```
  libiloader.a, libodbccli.a
  ```

- Windows

  ```
  iloader.lib, odbccli.lib
  ```

7.2.3 Samples

Sample iLoader API applications can be found in the `$ALTIBASE_HOME/sample/ILOADERAPI` directory.

7.3 iLoader API Data Structures

This section describes the C types that are provided for use in applications written using the iLoader API.

These types are:

- The iLoader Handle
- Error Structure
- Log Structure
- Option Structure
- iLoader API Enumerators

7.3.1 The iLoader Handle

The iLoader handle is an opaque data structure that is defined by the iLoader API library. It is used to store information pertaining to the behavior of applications that use the iLoader API.

- ALTIBASE_ILOADER_HANDLE

  This is an iLoader handle. The iLoader handle is primarily used when downloading data, uploading data and creating FORM files.

  The iLoader handle is allocated with `altibase_iloader_init()` and freed with `altibase_iloader_final()`.

7.3.2 Error Structure

- ALTIBASE_ILOADER_ERROR

  This structure is used to store information for diagnosing errors that occur during the execution of an application that was written using the iLoader API.

  This structure is defined as follows:

```
typedef struct ALTIBASE_ILOADER_ERROR
{
    int   errorCode;              /* Error Code */
    char *errorState;             /* SQLSTATE Code */
    char *errorMessage;           /* Error Message */
} ALTIBASE_ILOADER_ERROR;
```

7.3.3 Log Structure

The iLoader API provides the following two structures for use in logging the progress of an iLoader task:

- ALTIBASE_ILOADER_LOG

  This structure is passed to an application's callback function every time an error occurs during the execution of an iLoader task.

  It is also passed to a callback function upon completion of an iLoader task. At this time, the `record`, `recordData`, `recordColCount`, and `errorMgr` members are not passed.

  The purpose of this structure is to report errors that occur during the execution of an iLoader task, and to provide information about the results of execution of a task.

  This structure is defined as follows:

```
typedef struct ALTIBASE_ILOADER_LOG
{
    char                        tableName[50];
    int                         totalCount;
    int                         loadCount;
    int                         errorCount;
    int                         record;
    char                       **recordData;
    int                         recordColCount;
    ALTIBASE_ILOADER_ERROR      errorMgr;
} ALTIBASE_ILOADER_LOG;
```

Member	Description
`tableName`	This is the name of the table being uploaded or downloaded.
`totalCount`	This is the total number of rows for which an upload or download attempt has been made.
`loadCount`	This is the total number of rows that have been successfully uploaded or downloaded.
`errorCount`	This is the number of rows that could not be uploaded or downloaded due to the occurrence of an error. Note that when an error occurs, this count does not include the current error. That is, it is a count of all errors preceding the current error.
`record`	When an error occurs, this indicates the position of the record that could not be uploaded or downloaded.
`recordData`	When an error occurs, this is the data stored in the record that could not be uploaded or downloaded.
`recordColCount`	When an error occurs, this is the number of columns in the record that could not be uploaded or downloaded.
`errorMgr`	When an error occurs, this is an error structure that contains information about the error.

- ALTIBASE_ILOADER_STATISTIC_LOG

This structure is periodically passed to an application's callback function during the execution of an iLoader task. The frequency with which it is passed is determined by the setRowFrequency option in the Option Structure.

This structure is used to pass statistics about the execution of an iLoader task. These statistics are: the time at which the task started, the total number of rows to be uploaded or downloaded, the number of rows that have been successfully uploaded or downloaded, and the number of rows that could not be uploaded or downloaded due to the occurrence of an error.

This structure is defined as follows:

```
typedef struct ALTIBASE_ILOADER_STATISTIC_LOG
{
    char                        tableName[50];
    time_t                      startTime;
    int                         totalCount;
    int                         loadCount;
    int                         errorCount;
} ALTIBASE_ILOADER_STATISTIC_LOG;
```

Member	Description
tableName	The name of the table being uploaded or downloaded.
startTime	This is the time at which the upload or download task started.
totalCount	This is the total number of rows to be uploaded. This member is not used when downloading data.
loadCount	This is the total number of rows that have been successfully uploaded or downloaded.
errorCount	This is the number of rows that could not be uploaded or downloaded due to the occurrence of an error.

7.3.4 Option Structure

- ALTIBASE_ILOADER_OPTION_V1

  Most of the members of the ALTIBASE_ILOADER_OPTION_V1 structure correspond to iLoader options. The corresponding option is noted in the comment following each member. For detailed information on the iLoader options, please refer to the *ALTIBASE HDB iLoader User's Manual*.

  The definitions of the iloBool, iloLoadMode, iloDirectMode and ALTIBASE_ILOADER_LOG_TYPE enumerators can be found iLoader API Enumerators.

  This structure is defined as follows:

```
typedef struct ALTIBASE_ILOADER_OPTIONS_V1
{
    int             version;
    char            loginID[128 * 2];       /* -u login_id */
    char            password[128];          /* -p password */
    char            serverName[128];        /* -s server_name */
    int             portNum;                /* -port port_no */
    char            NLS[128];               /* -nls_use characterset */
    char            DBName[128];
    char            tableOwner[50];
    char            tableName[50];          /* -T table_name */
    char            formFile[1024];         /* -f formatfile */
    char            dataFile[32][1024];     /* -d datafile */
    int             dataFileNum;
    int             firstRow;               /* -F firstrow */
    int             lastRow;                /* -L lastrow */
    char            fieldTerm[11];          /* -t field_term */
    char            rowTerm[11];            /* -r row_term */
    char            enclosingChar[11];      /* -e enclosing_term */
    iloBool         useLobFile;             /* -lob use_lob_file */
    iloBool         useSeparateFile;        /* -lob use_separate_file */
    char            lobFileSize[11];        /* -lob log_file_size */
    char            lobIndicator[11];       /* -lob lob_indicator*/
    iloBool         replication;            /* -replication true/false */
    iloLoadMode     loadModeType;           /* -mode mode_type */
    char            bad[1024];              /* -bad bad_file */
    char            log[1024];              /* -log log_file */
    int             splitRowCount;          /* -split n */
    int             errorCount;             /* -errors count */
    int             arrayCount;             /* -array array_size */
    int             commitUnit;             /* -commit commit_unit */
    iloBool         atomic;                 /* -atomic */
    iloDirectMode   directLog;              /* -direct log/nolog */
    int             parallelCount;          /* -parallel count */
    int             readSize;               /* -readSize size */
    iloBool         informix;
    iloBool         flock;
    iloBool         mssql;
    iloBool         getTotalRowCount;
    int             setRowFrequency;
} ALTIBASE_ILOADER_OPTIONS_V1;
```

Member	Description
`version`	This must be set to the same value as the `version` argument that is passed by `altibase_iloader_options_init()`.
`tableOwner`	This is used to specify the name of the table owner.
`loadModeType`	ILO_APPEND: This is the same as APPEND, one of the possible values for the iLoader `-mode` option. ILO_REPLACE: This is the same as REPLACE, one of the possible values for the iLoader `-mode` option. ILO_TRUNCATE: This is the same as TRUNCATE, one of the possible values for the iLoader `-mode` option. The default value is ILO_APPEND.

Member	Description
atomic	This is used to specify whether to use Atomic Array INSERT. It can be either ILO_TRUE or ILO_FALSE. The default value is ILO_FALSE.
directLog	This is used to specify whether to use direct-path INSERT. If it is set to ILO_DIRECT_NONE, Direct-Path INSERT is not used. If it is set to ILO_DIRECT_LOG, Direct-Path INSERT is executed in logging mode. If it is set to ILO_DIRECT_NOLOG, Direct-Path INSERT is executed in nologging mode. The default value is ILO_DIRECT_NONE.
dataFileNum	This is used to specify the number of datafiles for the dataFile member in this structure.
getTotalRowCount	This specifies whether to get the total number of rows in the datafiles and set the totalCount member in the ALTIBASE_ILOADER_STATISTIC_LOG structure to this number when uploading data. It can be either ILO_TRUE or ILO_FALSE. The default value is ILO_FALSE.
setRowFrequency	The user callback function is called every time the number of rows specified here is uploaded or downloaded. The default value is 0. If this value is set to 0, the callback function is never called.

7.3.5 iLoader API Enumerators

```
typedef enum
{
    ILO_FALSE = 0,                  /* false */
    ILO_TRUE  = 1                   /* true */
} iloBool;
typedef enum
{
    ILO_APPEND,
    ILO_REPLACE,
    ILO_TRUNCATE
} iloLoadMode;
typedef enum
{
    ILO_DIRECT_NONE,
    ILO_DIRECT_LOG,
    ILO_DIRECT_NOLOG
} iloDirectMode;
typedef enum
{
    ILO_LOG,
    ILO_STATISTIC_LOG
} ALTIBASE_ILOADER_LOG_TYPE;
```

7.4 The iLoader API

This section describes each of the functions in the iLoader API.

The following information is provided for each function.

- The name and purpose of the function
- The function syntax
- A list of arguments for the function
- The function's return values
- Diagnostics for the function
- Notes related to use of the function
- A list of related functions
- An example of use of the function in code

7.4.1 altibase_iloader_datain

This function is used to upload data into a database table.

7.4.1.1 Syntax

```
int altibase_iloader_datain ( ALTIBASE_ILOADER_HANDLE * handle,
                              int                       version
                              void *                    options
                              ALTIBASE_ILOADER_CALLBACK logCallback,
                              ALTIBASE_ILOADER_ERROR *  error );
```

7.4.1.2 Arguments

Argument	In/Out	Description
handle	Input	This is the pointer to the iLoader handle.
version	Input	This is the version of the iLoader API.
options	Input	This is the pointer to the option structure.
logCallback	Input	This is the name of a log callback function. It may be a user-defined function. It can be NULL.
error	Output	This is the pointer to the error structure in which to return information for diagnosing errors. For more information about the error structure, please refer to iLoader API Data Structures.

7.4.1.3 Return Values

`ALTIBASE_ILO_SUCCESS, ALTIBASE_ILO_ERROR, or ALTIBASE_ILO_WARNING`

If the overall upload operation succeeded but one or more errors occurred, `ALTIBASE_ILO_WARNING` is returned.

7.4.1.4 Diagnostics

When `altibase_iloader_datain()` returns either `ALTIBASE_ILO_ERROR` or `ALTIBASE_ILO_WARNING`, the associated error information is returned in `error`.

7.4.1.5 Description

`altibase_iloader_datain()` is used to upload data from a file into a database table.

The value of the *version* argument must be `ALTIBASE_ILOADER_V1`.

If a user-defined log callback function is being used, specify the name of the function in *logCallback*. Set *logCallback* to NULL when not using a user-defined log callback function.

7.4.1.6 Related Functions

`altibase_iloader_init`

`altibase_iloader_options_init`

`altibase_iloader_formout`

`altibase_iloader_final`

7.4.1.7 Example

The following example shows how to specify a format file and a data file when uploading data, both when using the iLoader utility directly and when calling the iLoader API from within an application.

- Uploading Data using the iLoader Utility

```
iloader in -s 127.0.0.1 -u sys -p manager -f t1.fmt -d t1.dat
```

- Uploading Data Using an Application that Calls the iLoader API

```
int main()
{
    ALTIBASE_ILOADER_HANDLE      handle = ALTIBASE_ILOADER_NULL_HANDLE;
    ALTIBASE_ILOADER_OPTIONS_V1  opt;
    ALTIBASE_ILOADER_ERROR       err;
    int rc;

    /* Allocate an ILOADER handle */
    rc = altibase_iloader_init(&handle);

    if ( rc != ALTIBASE_ILO_SUCCESS )
    {
```

```
            printf("altibase_iloader_init() failed: %d\n", rc);
            return 1;
        }

        /* Initialize an option structure */
        altibase_iloader_options_init(ALTIBASE_ILOADER_V1, &opt);

        strcpy(opt.serverName, "127.0.0.1");
        strcpy(opt.loginID, "sys");
        strcpy(opt.password, "manager");
        strcpy(opt.formFile, "t1.fmt");
        strcpy(opt.dataFile[0], "t1.dat");
        opt.dataFileNum = 1;

        /* Upload data */
        rc = altibase_iloader_datain(&handle, ALTIBASE_ILOADER_V1, &opt,
    NULL, &err);

        if ( rc == ALTIBASE_ILO_SUCCESS )
        {
            printf("SUCCESS\n");
        }
        else
        {
            printf("ERR-%05X [%s] %s\n",
                    err.errorCode,
                    err.errorState,
                    err.errorMessage);
        }

        if ( handle != ALTIBASE_ILOADER_NULL_HANDLE )
        {
            altibase_iloader_final(&handle);
        }

        return 0;
    }
```

7.4.2 altibase_iloader_dataout

This function is used to download data from a database table and write the data to a file.

7.4.2.1 Syntax

```
int altibase_iloader_dataout ( ALTIBASE_ILOADER_HANDLE * handle,
                               int                       version
                               void *                    options
                               ALTIBASE_ILOADER_CALLBACK logCallback,
                               ALTIBASE_ILOADER_ERROR *  error );
```

7.4.2.2 Arguments

Argument	In/Out	Description
handle	Input	This is the pointer to the iLoader handle.

Argument	In/Out	Description
version	Input	This is the version of the iLoader API.
options	Input	This is the pointer to the option structure.
logCallback	Input	This is the name of a log callback function. It may be a user-defined function.
error	Output	This is the pointer to the diagnostic error structure in which to return information for diagnosing errors. For more information about this structure, please refer to iLoader API Data Structures.

7.4.2.3 Return Values

`ALTIBASE_ILO_SUCCESS or ALTIBASE_ILO_ERROR`

7.4.2.4 Diagnostics

When `altibase_iloader_dataout()` returns `ALTIBASE_ILO_ERROR`, the associated error information is returned in *error*.

7.4.2.5 Description

`altibase_iloader_dataout()` is used to download data from a database table and write the data to a file.

The value of the *version* argument must be `ALTIBASE_ILOADER_V1`.

If a user-defined log callback function is being used, specify the name of the function in *logCallback*. Set *logCallback* to NULL when not using a user-defined log callback function.

7.4.2.6 Related Functions

`altibase_iloader_init`

`altibase_iloader_options_init`

`altibase_iloader_formout`

`altibase_iloader_final`

7.4.2.7 Example

The following example shows how to specify a format file and a data file when downloading data, both when using the iLoader utility directly and when calling the iLoader API from within an application.

- Downloading Data Using the iLoader Utility

```
iloader out -s 127.0.0.1 -u sys -p manager -f t1.fmt -d t1.dat
```

- Downloading Data Using an Application that Calls the iLoader API

```
int main()
{
    ALTIBASE_ILOADER_HANDLE      handle = ALTIBASE_ILOADER_NULL_HANDLE;
    ALTIBASE_ILOADER_OPTIONS_V1  opt;
    ALTIBASE_ILOADER_ERROR       err;
    int rc;

    /* Allocate an iLoader handle */
    rc = altibase_iloader_init(&handle);

    if ( rc != ALTIBASE_ILO_SUCCESS )
    {
        printf("altibase_iloader_init() failed: %d\n", rc);
        return 1;
    }

    /* Initialize an option structure */
    altibase_iloader_options_init(ALTIBASE_ILOADER_V1, &opt);

    strcpy(opt.serverName, "127.0.0.1");
    strcpy(opt.loginID, "sys");
    strcpy(opt.password, "manager");
    strcpy(opt.formFile, "t1.fmt");
    strcpy(opt.dataFile[0], "t1.dat");
    opt.dataFileNum = 1;

    /* Download data */
    rc = altibase_iloader_dataout(&handle, ALTIBASE_ILOADER_V1, &opt,
NULL, &err);

    if ( rc == ALTIBASE_ILO_SUCCESS )
    {
        printf("SUCCESS\n");
    }
    else
    {
        printf("ERR-%05X [%s] %s\n",
                err.errorCode,
                err.errorState,
                err.errorMessage);
    }

    if ( handle != ALTIBASE_ILOADER_NULL_HANDLE )
    {
        altibase_iloader_final(&handle);
    }

    return 0;
}
```

7.4.3 altibase_iloader_final

This function is used to free a handle and all associated resources.

7.4.3.1 Syntax

```
int altibase_iloader_final ( ALTIBASE_ILOADER_HANDLE *  handle );
```

7.4.3.2 Arguments

Argument	In/Out	Description
handle	Input	This is the pointer to the iLoader handle to be freed.

7.4.3.3 Return Values

`ALTIBASE_ILO_SUCCESS or ALTIBASE_ILO_ERROR`

7.4.3.4 Description

`altibase_iloader_final()` frees the resources associated with the specified iLoader handle.

After a handle has been freed, it cannot be used by the application.

7.4.3.5 Related Functions

`altibase_iloader_init`

7.4.3.6 Example

Please refer to altibase_iloader_init.

7.4.4 altibase_iloader_formout

This function is used to create a table format file (i.e. FORM file).

7.4.4.1 Syntax

```
int altibase_iloader_formout ( ALTIBASE_ILOADER_HANDLE * handle,
                               int                       version
                               void *                    options
                               ALTIBASE_ILOADER_ERROR *  error );
```

7.4.4.2 Arguments

Argument	In/Out	Description
handle	Input	This is the pointer to the iLoader handle.
version	Input	This is the version of the iLoader API.
options	Input	This is the pointer to the option structure.

Argument	In/Out	Description
error	Output	This is the pointer to the error structure in which to return information for diagnosing errors. For more information about this structure, please refer to iLoader API Data Structures.

7.4.4.3 Return Values

`ALTIBASE_ILO_SUCCESS or ALTIBASE_ILO_ERROR`

7.4.4.4 Diagnostics

When `altibase_iloader_formout()` returns `ALTIBASE_ILO_ERROR`, the associated error information is returned in *error*.

7.4.4.5 Description

`altibase_iloader_formout()` is used to create a format file (FORM file) that describes a database table.

This function can be called only after `altibase_iloader_init()` and `altibase_iloader_options_init()` have been called.

The value of the *version* argument must be `ALTIBASE_ILOADER_V1`.

7.4.4.6 Related Functions

`altibase_iloader_init`

`altibase_iloader_options_init`

`altibase_iloader_datain`

`altibase_iloader_dataout`

`altibase_iloader_final`

7.4.4.7 Example

The following example shows how to create a format file that describes table *T1*, both when using the iLoader utility directly and when calling the iLoader API from within an application.

- Creating a Format File Using the iLoader Utility

```
iloader formout -s 127.0.0.1 -u sys -p manager -T T1 -f t1.fmt
```

- Creating a Format File Using an Application that Calls the iLoader API.

```
int main()
{
```

```
    ALTIBASE_ILOADER_HANDLE     handle = ALTIBASE_ILOADER_NULL_HANDLE;
    ALTIBASE_ILOADER_OPTIONS_V1 opt;
    ALTIBASE_ILOADER_ERROR      err;
    int rc;

    /* Allocate an iLoader handle */
    rc = altibase_iloader_init(&handle);

    if ( rc != ALTIBASE_ILO_SUCCESS )
    {
        printf("Failed to altibase_iloader_init() failed: %d\n", rc);
        return 1;
    }

    /* Initialize an option structure */
    altibase_iloader_options_init(ALTIBASE_ILOADER_V1, &opt);

    strcpy(opt.serverName, "127.0.0.1");
    strcpy(opt.loginID, "sys");
    strcpy(opt.password, "manager");
    strcpy(opt.tableName, "t1");
    strcpy(opt.formFile, "t1.fmt");

    /* formout */
    rc = altibase_iloader_formout(&handle, ALTIBASE_ILOADER_V1, &opt,
&err);

    if ( rc == ALTIBASE_ILO_SUCCESS )
    {
        printf("SUCCESS\n");
    }
    else
    {
        printf("ERR-%05X [%s] %s\n",
                err.errorCode,
                err.errorState,
                err.errorMessage);
    }

    if ( handle != ALTIBASE_ILOADER_NULL_HANDLE )
    {
        altibase_iloader_final(&handle);
    }

    return 0;
}
```

7.4.5 altibase_iloader_init

This function allocates an iLoader handle.

7.4.5.1 Syntax

```
int altibase_iloader_init ( ALTIBASE_ILOADER_HANDLE *  handle );
```

7.4.5.2 Arguments

Argument	In/Out	Description
handle	Output	This is a pointer to a buffer in which the handle to the newly allocated data structure is returned.

7.4.5.3 Return Values

`ALTIBASE_ILO_SUCCESS or ALTIBASE_ILO_ERROR`

7.4.5.4 Description

`altibase_iloader_init()` allocates a handle that is used when creating a table format file, downloading data, or uploading data.

One handle cannot be shared by multiple threads, even if the threads are part of the same process.

7.4.5.5 Related Functions

`altibase_iloader_datain`

`altibase_iloader_dataout`

`altibase_iloader_formout`

`altibase_iloader_final`

7.4.5.6 Example

```
int main()
{
    ALTIBASE_ILOADER_HANDLE handle = ALTIBASE_ILOADER_NULL_HANDLE;
    int rc;

    rc = altibase_iloader_init( &handle );
    if ( rc != ILOADER_SUCCESS )
    {
        printf( "altibase_iloader_init() failed: %d\n",rc );
    }

    /* ... omit ... */

    if( handle != ALTIBASE_ILOADER_NULL_HANDLE )
    {
        altibase_iloader_final( &handle );
    }

    return 0;
}
```

7.4.6 altibase_iloader_options_init

This function initializes an option structure to the default values.

7.4.6.1 Syntax

```
int altibase_iloader_options_init ( int      version,
                                    void *   options );
```

7.4.6.2 Arguments

Argument	In/Out	Description
version	Input	This is the version of the iLoader API.
options	Input	This is the pointer to the option structure.

7.4.6.3 Return Values

```
ALTIBASE_ILO_SUCCESS or ALTIBASE_ILO_ERROR
```

7.4.6.4 Description

`altibase_iloader_options_init()` initializes an option structure to the default values. For detailed information about the default values, please refer to iLoader API Data Structures and to the *ALTIBASE HDB iLoader User's Manual*.

The option structure must be initialized using this function before setting the option structure.

7.4.6.5 Related Functions

`altibase_iloader_init`

7.4.6.6 Example

Please refer to the examples for the altibase_iloader_datain and altibase_iloader_dataout function.

7.4.7 CallbackFunctionName

This function is a user-defined callback function for handling the log information that is generated during the execution of an application that uses the iLoader API.

7.4.7.1 Syntax

```
int CallbackFuncationName ( ALTIBASE_ILOADER_LOG_TYPE   type,
                            void *                      log );
```

7.4.7.2 Arguments

Argument	In/Out	Description
type	Input	This is the type of the log structure to return. It may be either ILO_LOG or ILO_STATISTIC_LOG.
log	Input	This is the pointer to the log structure. Depending on the value of `type`, this argument will point to an ALTIBASE_ILOADER_LOG structure or to an ALTIBASE_ILOADER_STATISTIC_LOG structure. If `type` is ILO_LOG, `log` is a pointer to an ALTIBASE_ILOADER_LOG structure, whereas if `type` is ILO_STATISTIC_LOG, `log` is a pointer to an ALTIBASE_ILOADER_STATISTIC_LOG structure. For more information about the structures, please refer to Log Structure.

7.4.7.3 Return Values

If ILO_STATISTIC_LOG is specified for *type* when a user-defined callback function is called, and the user-defined callback function returns anything other than 0 (zero), execution of the current upload or download task will stop.

7.4.7.4 Description

The ALTIBASE HDB iLoader API has the capability to execute user-specific code in addition to iLoader API calls. This functionality allows users to control execution of their applications based on the contents of iLoader logs. When `altibase_iloader_datain()` or `altibase_iloader_dataout()` is called, a callback function, which may be a user-defined function, can be registered. When the callback function is called, it receives the following iLoader log information: the time at which the task started, the total number of rows to be uploaded or downloaded, the number of rows that have been successfully uploaded or downloaded, and the number of rows that could not be uploaded or downloaded due to the occurrence of an error.

The application's callback function will be called at the following times:

- When an error occurs during the course of an upload or download operation. At this time, the structure that is passed to the callback function is ILO_LOG (ALTIBASE_ILOADER_LOG). The ALTIBASE_ILOADER_LOG structure contains an Error Structure, which contains an *errorCode* member. If the value of that member is not 0, this means that an error has occurred.
- After an iLoader task has completely executed following a call to `altibase_iloader_datain()` or `altibase_iloader_dataout()`. At this time, the structure that is passed to the callback function is ILO_LOG (ALTIBASE_ILOADER_LOG). If the value of the `record` member of the log structure is 0, this means that execution has completed.
- Every time the number of rows specified in `setRowFrequency`, a member of the Option Structure, is uploaded or downloaded. At this time, the structure that is passed to the callback function is ILO_STATISTIC_LOG (ALTIBASE_ILOADER_STATISTIC_LOG).

If a user-defined callback function returns anything other than 0 (zero), execution of the current upload or download task will stop. Note that at that point, the value of `loadCount` may be incorrect.

Note that although the user callback function is supposed to be called whenever the number of rows specified in the `setRowFrequency` member is uploaded or downloaded, this behavior may not be exhibited in the following cases:

— When the value of the `arrayCount` member of the option structure is greater than 1, a user callback function cannot be called during the execution of an upload task.
— When the value of the `parallelCount` member of the option structure is greater than 1, a user callback function may not be called for reasons related to synchronization between multiple threads.

7.4.7.5 Related Functions

`altibase_iloader_datain`

`altibase_iloader_dataout`

7.4.7.6 Example

- Defining User Callback

```
int print_callback ( ALTIBASE_ILOADER_LOG_TYPE type, void *log)
{
    int i;

    ALTIBASE_ILOADER_LOG           *slog;
    ALTIBASE_ILOADER_STATISTIC_LOG *statisticlog;

    if ( type == ILO_LOG )
    {
        slog = (ALTIBASE_ILOADER_LOG *) log;

        if ( slog->record == 0 )
        {
            printf("LOG Total Count : %d\n", slog->totalCount);
            printf("LOG Load Count : %d\n", slog->loadCount);
            printf("LOG Error Count : %d\n", slog->errorCount);
        }
        else
        {
            printf("LOG %d\n", slog->record);
            for (i = 0; i < slog->recordColCount; i++)
            {
                printf("    [%d] : %s\n", i, slog->recordData[i]);
            }
        }

        if ( slog->errorMgr.errorCode != 0 )
        {
            printf("    ERR-%05X [%s] %s\n",
                    slog->errorMgr.errorCode,
                    slog->errorMgr.errorState,
                    slog->errorMgr.errorMessage);
        }
    }
```

```
    else if ( type == ILO_STATISTIC_LOG )
    {
        statisticlog = (ALTIBASE_ILOADER_STATISTIC_LOG *) log;

        printf("STATISTIC LOG Start Time  : %s\n", ctime(&statisticlog-
>startTime));
        printf("STATISTIC LOG Table Name  : %s\n", statisticlog->table-
Name );
        printf("STATISTIC LOG Total Count : %d\n", statisticlog->totalC-
ount );
        printf("STATISTIC LOG Load Count  : %d\n", statisticlog->load-
Count);
        printf("STATISTIC LOG Error Count : %d\n", statisticlog->error-
Count);
    }

    return 0;
}
```

- Registering User Callback

```
...
/* upload data */
altibase_iloader_datain(&handle,
                        ALTIBASE_ILOADER_V1,
                        &opt,
                        print_callback,
                        &err);
....
```

8 The CheckServer API

8.1 Overview of the CheckServer API

The CheckServer API of ALTIBASE HDB is an application programming interface for creating applications that use function calls to monitor whether the ALTIBASE HDB server has terminated abnormally. The CheckServer API provides the same functionality as the CheckServer utility. For more information about the CheckServer utility, please refer to the *ALTIBASE HDB Utilities Manual*.

The following table summarizes the CheckServer API functions.

Function Name	Purpose
altibase_check_server_init	This function allocates a CheckServer handle.
altibase_check_server_final	This function frees a handle and all associated resources.
altibase_check_server	This function monitors whether an ALTIBASE HDB server has terminated abnormally.
altibase_check_server_cancel	This function is used to terminate the execution of CheckServer.

8.1.1 Restrictions

- The CheckServer API does not support multi-threaded programs.
- An application that uses the CheckServer API can only be used to monitor an Altibase database server on the local host, i.e. on the same machine as the application.
- Running two or more applications that use the CheckServer API at the same time will cause application errors.

8.2 Using the CheckServer API

8.2.1 Header File

```
$ALTIBASE_HOME/include/chksvr.h
```

8.2.2 The CheckServer Libraries

The library files that are required in order for applications to use the CheckServer API reside in the `$ALTIBASE_HOME/lib` directory. Ensure that applications are always linked with the following libraries:

- UNIX

  ```
  libchksvr.a, libaltiutil.a
  ```

- Windows

  ```
  chksvr.lib, libaltiutil.lib
  ```

8.2.3 Samples

Sample applications that use the CheckServer API can be found in the `$ALTIBASE_HOME/sample/CHECKSERVER` directory.

8.3 CheckServer API Data Structure

This section describes the C type that is made available to applications that use the CheckServer API.

8.3.1 The CheckServer Handle

The CheckServer handle is an opaque data structure that is defined in the CheckServer API library. It is used to store information pertaining to the behavior of applications that use the CheckServer API.

- ALTIBASE_CHECK_SERVER_HANDLE

  This is the CheckServer handle. The CheckServer handle is primarily used when monitoring an ALTIBASE HDB server. The CheckServer handle is allocated with `altibase_check_server_init()` and freed with `altibase_check_server_final()`.

8.4 The CheckServer API

This section describes each of the functions in the CheckServer API.

The following information is provided for each function.

- The name and purpose of the function
- The function syntax
- A list of arguments for the function
- The function's return values
- Diagnostics for the function
- Notes related to use of the function
- A list of related functions
- An example of use of the function in code

8.4.1 altibase_check_server

This function checks whether the ALTIBASE HDB process is running.

8.4.1.1 Syntax

```
int altibase_check_server ( ALTIBASE_CHECK_SERVER_HANDLE handle );
```

8.4.1.2 Arguments

Argument	In/Out	Description
handle	Input	The CheckServer handle

8.4.1.3 Return Values

`ALTIBASE_CS_SERVER_STOPPED, ALTIBASE_CS_ERROR, or ALTIBASE_CS_INVALID_HANDLE`

If the ALTIBASE HDB server terminates abnormally, this function returns ALTIBASE_CS_SERVER_STOPPED.

8.4.1.4 Description

When CheckServer is started or this function is called, a file named `checkserver.pid` is created in the `$ALTIBASE_HOME/trc` directory. The presence of the `checkserver.pid` file prevents

another instance of the CheckServer API application from being started while the current instance is running. Calling `altibase_check_server_final()` removes this file.

If this function is called while the ALTIBASE HDB server is running, the application that called this function will be nonresponsive until an error occurs, until it is detected that the ALTIBASE HDB server is shutting down, or until `altibase_check_server_cancel()` is called.

8.4.1.5 Related Functions

`altibase_check_server_init`
`altibase_check_server_final`
`altibase_check_server_cancel`

8.4.1.6 Example

```
int main()
{
    ALTIBASE_CHECK_SERVER_HANDLE handle = ALTIBASE_CHECK_SERVER_NULL_HANDLE;
    char *homeDir = NULL;
    int rc;

    rc = altibase_check_server_init( &handle, homeDir );

    if ( rc != ALTIBASE_CS_SUCCESS )
    {
        printf( "altibase_check_server_init() failed: %d\n", rc );
    }

    rc = altibase_check_server(handle);

    if ( rc == ALTIBASE_CS_SERVER_STOPED )
    {
        printf( "Server stopped.\n" );
    }
    else
    {
        printf( "An error has occured: %d\n", rc );
    }

    if ( handle != ALTIBASE_CHECK_SERVER_NULL_HANDLE )
    {
        altibase_check_server_final(&handle);
    }

    return 0;
}
```

8.4.2 altibase_check_server_final

This function frees a handle and all associated resources.

8.4.2.1 Syntax

```
int altibase_check_server_final (
        ALTIBASE_CHECK_SERVER_HANDLE *  handle );
```

8.4.2.2 Arguments

Argument	In/Out	Description
handle	Input	This is the pointer to the CheckServer handle to be freed.

8.4.2.3 Return Values

`ALTIBASE_CS_SUCCESS, ALTIBASE_CS_ERROR, or ALTIBASE_CS_INVALID_HANDLE`

8.4.2.4 Description

`altibase_check_server_final()` frees all resources associated with the specified CheckServer handle.

Additionally, this function removes the `checkserver.pid` file, which was created when `altibase_check_server()` was called. If the application using the CheckServer API is terminated abnormally using a command such as `kill`, the `checkserver.pid` file may be left in the file system. In this case, it will be necessary to manually delete the file before it will be possible to run the CheckServer utility or for the application that uses the CheckServer API to call `altibase_check_server()`.

8.4.2.5 Related Functions

`altibase_check_server_init`

8.4.2.6 Example

See altibase_check_server.

8.4.3 altibase_check_server_init

This function allocates a CheckServer handle.

8.4.3.1 Syntax

```
int altibase_check_server_init (
        ALTIBASE_CHECK_SERVER_HANDLE *  handle,
        char *                          home_dir );
```

8.4.3.2 Arguments

Argument	In/Out	Description
handle	Output	This is a pointer to a buffer in which the handle to the newly allocated data structure is returned.
home_dir	Input	This must be set to the `$ALTIBASE_HOME` directory.

8.4.3.3 Return Values

`ALTIBASE_CS_SUCCESS or ALTIBASE_CS_ERROR`

8.4.3.4 Description

This function allocates memory for information related to CheckServer, and passes a pointer to this memory back in `*handle`. Only one ALTIBASE HDB server can be monitored by one application that uses the CheckServer API. The location of this server is specified using the `home_dir` argument. If `home_dir` is set to NULL, the value of the `ALTIBASE_HOME` environment variable is used.

Only one CheckServer handle can be used within one application. Additionally, a CheckServer handle cannot be shared by more than one thread at the same time.

8.4.3.5 Related Functions

`altibase_check_server`

8.4.3.6 Example

See altibase_check_server.

8.4.4 altibase_check_server_cancel

This function cancels the processing of the `altibase_check_server()` function associated with the specified handle.

8.4.4.1 Syntax

```
int altibase_check_server_cancel (
        ALTIBASE_CHECK_SERVER_HANDLE handle);
```

8.4.4.2 Arguments

Argument	In/Out	Description
handle	Input	This is the CheckServer handle to be canceled.

8.4.4.3 Return Values

```
ALTIBASE_CS_SUCCESS or ALTIBASE_CS_ERROR or
ALTIBASE_CS_INVALID_HANDLE
```

8.4.4.4 Description

In a multithreaded application, one thread can call `altibase_check_server_cancel()` to cancel the `altibase_check_server()` function that is running on another thread. If the call to `altibase_check_server_cancel()` is successful, `altibase_check_server()` is stopped, and the value ALTIBASE_CS_ABORTED_BY_USER is returned.

It might take some time from the time that `altibase_check_server_cancel()` is called until the execution of `altibase_check_server()` is terminated.

Once altibase_check_server_cancel() has been called, if it is called again before `altibase_check_server()` returns a result, there is no guarantee that either call to `altibase_check_server_cancel()` will execute correctly.

8.4.4.5 Related Functions

`altibase_check_server`

8.4.4.6 Example

See altibase_check_server.

Index

.NET Data Provider 52
    Array Binding 56
    Compiling Applications 54
    Data Type 61
    Schema 59

A

altibase_check_server() 149
altibase_check_server_cancel() 152
altibase_check_server_final() 150
altibase_check_server_init() 151
altibase_iloader_datain() 131
altibase_iloader_dataout() 133
altibase_iloader_final() 135
altibase_iloader_formout() 136
altibase_iloader_init() 138
altibase_iloader_options_init() 140

B

Blob 34

C

CallableStatement 31
CheckServer API
    Data Structure 148
    Libraries 147
    overview 146
    restrictions 146
    Samples 147
Clob 35
Connection 16
Connection String 65, 85
ConnectionPoolDataSource 35

D

DatabaseMetaData 17
DataSource 35
Driver 15

E

Executing JDBC/XA 101
Executing ODBC/XA 99
Executing SES/XA 100

I

iLoader API
    CallbackFunction 140
    Data Structures 126
    Libraries 125
    overview 124
    Samples 125
Installing Altibase PERL DBD 48

J

JAVA Application 2
JDBC 3.0 API 15
JDBC Connection Fail-over 38
JDBC Distributed Transactions 109
JDBC driver version 2
Jeus 2

N

National Character Sets 9

P

PERL DBD 46
PERL DBI 46
PERL Package Installation 47
PHP
    Installing ODBC Manager 41
    Sample Test 43
    Unix ODBC 41
    Windows ODBC 42
PHP Functions for ODBC Connectivity 43
PHP Module 40
PooledConnection 36
PreparedStatement 30
Processing in-doubt Transactions 120

R

ResultSet 23
ResultSetMetaData 28

S

SavePoint 35
Setting up Connection Pool 10
Settings in JSP 8
Statement 28

T

Tomcat 2
TPM Application 104

U

Using .NET Data Provider 54
Using JDBC to Connect to Altibase 3

Using XA 99

W

WAS 10
WebLogic 2

X

XA Interface 94
    Limitations 106
    overview 90
XA Library 93
XA Tracking Information 120
XAConnection 36
XAConnection Interface 110
XADataSource 36
XAResource 37
xa_switch_t Structure 92
Xid 37
Xid interface 112