Altibase® Application Development

# Altibase Monitoring API Developer's Guide

**Release 7.1 (July 5, 2017)**

ALTIBASE

Altibase® Altibase Monitoring API Developer's Guide
Release 7.1


Altibase Corp.
10F, Daerung PostTower II,
306, Digital-ro, Guro-gu, Seoul 08378, Korea
Telephone: +82-2-2082-1000 Fax: 82-2-2082-1099
Homepage: http://www.altibase.com

# Contents

# Preface

# About This Manual

This manual explains how to use Altibase Monitoring API.

## Audience

This manual has been prepared for the following Altibase users:

- Database administrators
- Performance managers
- Database users
- Application developers
- Technical support staff

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
- Experience in server management, operating systems or network management

## Software Environment

This manual has been prepared assuming that Altibase 7.1 will be used as the database server.

## Organization

This manual has been organized as follows:

- Chapter1: Introduction
  This chapter discusses Altibase Monitoring API and its features.
- Chapter2: Data Types
  This chapter discusses data types that can be used with Altibase Monitoring API.

- Chapter3: Functions

  This chapter discusses Altibase Monitoring API functions.

- Chapter4: Sample Programs

  This chapter provides sample programs in C that were written using Altibase Monitoring API.

## 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 other manuals in the series.

| Convention | Meaning |
|---|---|
| Italic Font | Variables and special values specified by the user |
| Fixed-width Font | Commands within paragraghs or sample codes |

## Related Documents

For more detailed information, please refer to the following documents:

- Administrator's Manual
- Error Message Reference
- Getting Started Guide
- Installation Guide
- iSQL User's Manual
- ODBC Reference
- Precompiler User's Manual
- Replication Manual
- SQL Reference
- Utilities Manual

## Online Manuals

Online versions of our manuals (PDF or HTML) are available from Altibase's Customer Support

site (http://altibase.com/support-center/).

## Altibase Welcomes Your Opinions

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 you are using
- Your comments and suggestions regarding the manual
- Your full name, address, and phone number

In addition to suggestions, this address may also be used to report any errors or omissions discovered in the manual, which we will address promptly.

If you need immediate assistance with technical issues, please contact Altibase's Customer Support site (http://altibase.com/support-center/).

We always appreciate your comments and suggestions.

# 1 Introduction

This chapter discusses Altibase Monitoring API and its features.

# 1.1 What is Altibase Monitoring API?

Altibase Monitoring API is an application programming interface that lets you monitor Altibase from an application.

## 1.1.1 Usage

Altibase Monitoring API is an interface provided for remote monitoring tool developers and allows developers to easily create monitoring tools. You can also collect monitoring data by directly selecting a performance view on Altibase.

By default, it is not provided for users using other interfaces for database access such as ODBC and JDBC.

## 1.1.2 Features

You can view the following data in an application with Altibase Monitoring API:

- Various statistics while Altibase is running
- The number of currently connected sessions
- The maximum number of clients that can connect to an Altibase server
- Session lock information
- Wait event information

## 1.1.3 Supported Altibase Versions

Altibase Monitoring API is supported for Altibase 5.5.1 or above.

## 1.1.4 Considerations

Please consider the following while writng and executing an application with Altibase Monitoring API:

- An Alitbase Monitoring API application connects to an Altibase server using a Unix domain socket. Therefore, the application and Altibase need to run on the same server.

- Memory that is internally allocated by an Altibase Monitoring API function or library is shared by Altibase Monitoring API functions and is not thread-safe. Accordingly, multiple threads should be prevented from concurrently accessing shared memory by synchronization using mutexes. For further information, please refer to sample_7.c.

# 1.2 Building an Application

This section discusses the necessary header file and library files for building an Altibase Monitoring API application and how to compile them.

## 1.2.1 Header File

You need to include and reference the header file altibaseMonitor.h when writing an Alitbase Monitoring API application. This file is located in the $ALTIBASE_HDB_HOME/include directory.

To compile, use the following command-line option.

-I$ALTIBASE_HDB_HOME/include

## 1.2.2 Library Files

To build an Altibase Monitoring API application, you need to link the compiled object file to the Altibase Monitoring API library, ODBC library and several other system libraries.

- Altibase Monitoring API library: libaltibaseMonitor.a, libaltibaseMonitor_sl.so
- ODBC library: libodbccli.a
- System libraries: libpthread.a, libdl.a

The Altibase Monitoring API library and ODBC library are located in the $ALTIBASE_HDB_HOME/lib directory.

## 1.2.3 Compiling

The following is a sample Makefile that compiles the sample.c source file with the $(ALTIBASE_HOME)/install/altibase_env.mk file that is created when the Altibase package is installed.

```
include $(ALTIBASE_HOME)/install/altibase_env.mk
sample: sample.o
        $(LD) $(LDOUT)sample sample.o $(LFLAGS) -laltibaseMonitor -lodbccli $(LIBS)
```

The following example compiles the sample.c source file using the gcc and g++ compilers on the console window.

```
% gcc -c -I$ALTIBASE_HOME/include -o sample.o sample.c
% g++ -o sample sample.o -L$ALTIBASE_HOME/lib -laltibaseMonitor -lodbccli -ldl -lpthread -lcrypt -lrtdl
```

# 2 Data Types

This chapter discusses data types that can be used with Altibase Monitoring API.

# 2.1 Data Structures

This section discusses data structures used as arguments for calling Altibase Monitoring API functions.

## 2.1.1 ABIVSession

This data structure stores the results of SELECT operations on the V$SESSION performance view.

This data structure has the following members. For further information about each column, please refer to the V$SESSION performance view in the *General Reference*.

| Member | Type | Corresponding Column in V$SESSION |
|---|---|---|
| mID | int | ID |
| mTransID | long | TRANS_ID |
| mTaskState[11+1] | char | TASK_STATE |
| mCommName[64+1] | char | COMM_NAME |
| mXASessionFlag | int | XA_SESSION_FLAG |
| mXAAssociateFlag | int | XA_ASSOCIATE_FLAG |
| mQueryTimeLimit | int | QUERY_TIME_LIMIT |
| mDdlTimeLimit | int | DDL_TIME_LIMIT |
| mFetchTimeLimit | int | FETCH_TIME_LIMIT |
| mUTransTimeLimit | int | UTRANS_TIME_LIMIT |
| mIdleTimeLimit | int | IDLE_TIME_LIMIT |
| mIdleStartTime | int | IDLE_START_TIME |
| mActiveFlag | int | ACTIVE_FLAG |
| mOpenedStmtCount | int | OPENED_STMT_COUNT |
| mClientPackageVersion[40+1] | char | CLIENT_PACKAGE_VERSION |

| Member | Type | Corresponding Column in V$SESSION |
|---|---|---|
| mClientProtocolVersion[40+1] | char | CLIENT_PROTOCOL_VERSION |
| mClientPID | long | CLIENT_PID |
| mClientType[40+1] | char | CLIENT_TYPE |
| mClientAppInfo[128+1] | char | CLIENT_APP_INFO |
| mClientNls[40+1] | char | CLIENT_NLS |
| mDBUserName[40+1] | char | DB_USERNAME |
| mDBUserID | int | DB_USERID |
| mDefaultTbsID | long | DEFAULT_TBSID |
| mDefaultTempTbsID | long | DEFAULT_TEMP_TBSID |
| mSysDbaFlag | int | SYSDBA_FLAG |
| mAutoCommitFlag | int | AUTOCOMMIT_FLAG |
| mSessionState[13+1] | char | SESSION_STATE |
| mIsolationLevel | int | ISOLATION_LEVEL |
| mReplicationMode | int | REPLICATION_MODE |
| mTransactionMode | int | TRANSACTION_MODE |
| mCommitWriteWaitMode | int | COMMIT_WRITE_WAIT_MODE |
| mOptimizerMode | int | OPTIMIZER_MODE |
| mHeaderDisplayMode | int | HEADER_DISPLAY_MODE |
| mCurrentStmtID | int | CURRENT_STMT_ID |
| mStackSize | int | STACK_SIZE |
| mDefaultDateFormat[64+1] | char | DEFAULT_DATE_FORMAT |
| mTrxUpdateMaxLogSize | long | TRX_UPDATE_MAX_LOGSIZE |

| Member | Type | Corresponding Column in V$SESSION |
|---|---|---|
| mParallelDmlMode | int | PARALLEL_DML_MODE |
| mLoginTime | int | LOGIN_TIME |
| mFailOverSource[64+1] | char | FAILOVER_SOURCE |

## 2.1.2 ABIVSysstat

This data structure stores the results of SELECT operations on the V$SYSSTAT performance view. This performance view displays statistics about the entire database system.

This data structure has the following members. For further information about each column, please refer to the V$SYSSTAT performance view in the *General Reference*.

| Member | Type | Corresponding Column in V$SYSSTAT |
|---|---|---|
| mValue | long | VALUE |

## 2.1.3 ABIVSesstat

This data structure stores the results of SELECT operations on the V$SESSTAT performance view. This performance view displays statistics about each session.

This data structure has the following members. For further information about each column, please refer to the V$SESSTAT performance view in the *General Reference.*

| Member | Type | Corresponding Column in V$SESSTAT |
|---|---|---|
| mSID | int | SID |
| mValue | long | VALUE |

## 2.1.4 ABIStatName

This data structure stores the results of SELECT operations on the fixed columns of the V$SYSSTAT or V$SESSTAT performance view.

This data structure has the following members. For further information about each column,

please refer to the V$SYSSTAT and V$SESSTAT performance views in the *General Reference*.

| Member | Type | Corresponding Column in V$SYSSTAT or V$SESSTAT |
|---|---|---|
| mSeqNum | int | SEQNUM |
| mName[128+1] | char | NAME |

## 2.1.5 ABIVSystemEvent

This data structure stores the results of SELECT operations on the V$SYSTEM_EVENT performance view.

This data structure has the following members. For further information about each column, please refer to the V$SYSTEM_EVENT performance view in the *General Reference.*

| Member | Type | Corresponding Column in V$SYSTEM_EVENT |
|---|---|---|
| mTotalWaits | long | TOTAL_WAITS |
| mTotalTimeOuts | long | TOTAL_TIMEOUTS |
| mTimeWaited | long | TIME_WAITED |
| mAverageWait | long | AVERAGE_WAIT |
| mTimeWaitedMicro | long | TIME_WAITED_MICRO |

## 2.1.6 ABIVSessionEvent

This data structure stores the results of SELECT operations on the V$SESSION_EVENT performance view.

This data structure has the following members. For further information about each column, please refer to the V$SESSION _EVENT performance view in the *General Reference*..

| Member | Type | Corresponding Column in V$SESSION_EVENT |
|---|---|---|
| mSID | int | SID |
| mTotalWaits | long | TOTAL_WAITS |

| Member | Type | Corresponding Column in V$SESSION_EVENT |
|---|---|---|
| mTotalTimeOuts | long | TOTAL_TIMEOUTS |
| mTimeWaited | long | TIME_WAITED |
| mAverageWait | long | AVERAGE_WAIT |
| mMaxWait | long | MAX_WAIT |
| mTimeWaitedMicro | long | TIME_WAITED_MICRO |

## 2.1.7 ABIEventName

| Member | Type | Corresponding Column in V$SYSTEM_EVENT or V$SESSION_EVENT |
|---|---|---|
| mEventID | int | EVENT_ID |
| mEvent[128+1] | char | EVENT |
| mWaitClassID | int | WAIT_CLASS_ID |
| mWaitClass[128+1] | char | WAIT_CLASS |

This data structure stores the results of SELECT operations on the fixed columns of the V$SYSTEM_EVENT or V$SESSION_EVENT performance view.

This data structure has the following members. For further information about each column, please refer to the V$SYSTEM_EVENT and V$SESSION_EVENT performance views in the General Reference.

## 2.1.8 ABIVSessionWait

This data structure stores the results of SELECT operations on the V$SESSION_WAIT performance view.

This data structure has the following members. For further information about each column, please refer to the V$SESSION_WAIT performance view in the *General Reference.*

| Member | Type | Corresponding Column in V$SESSION_WAIT |
|---|---|---|
| mSID | int | SID |
| mSeqNum | int | SEQNUM |
| mP1 | long | P1 |
| mP2 | long | P2 |
| mP3 | long | P3 |
| mWaitClassID | int | WAIT_CLASS_ID |
| mWaitTime | long | WAIT_TIME |
| mSecondInTime | long | SECOND_IN_TIME |

## 2.1.9 ABISqlText

This is a data structure used for viewing SQL statement text, the start time of a query and checking whether or not to execute a query.

This data structure has the following members shown in the table below.

| Member | Type | Description |
|---|---|---|
| mSessID | int | Session ID |
| mStmtID | int | Statement ID |
| mSqlText | char * | Text of the SQL statement |
| mTextLength | int | Length of the string stored in mSqlText |
| mQueryStartTime | int | The start time of a query |
| mExecuteFlag | int | Whether or not to execute a query<br>0: Executable<br>1: Non-executable |

## 2.1.10 ABILockPair

This data structure retrieves a session holding on to a lock and the session waiting to acquire

that lock.

This data structure has the following members.

| Member | Type | Description |
|---|---|---|
| mHolderSID | int | ID of the session holding on to the lock |
| mWaiterSID | int | ID of the session that is waiting for another session (mHolderSID) to let go of the lock |
| mLockDesc[32+1] | char | Mode of the lock that the session (mWaiterSID) is waiting to obtain |

## 2.1.11 ABIDBInfo

This data structure retrieves database names and database version numbers.

This data structure has the following members.

| Member | Type | Description |
|---|---|---|
| mDBName[128+1] | char | Database name |
| mDBVersion[128+1] | char | Database version number |

## 2.1.12 ABIReadCount

This data structure retrieves the number of data pages that were read from the Altibase server.

This data structure has the following members.

| Member | Type | Description |
|---|---|---|
| mLogicalReadCount | int | The number of data pages that were read in the memory buffer |
| mPhysicalReadCount | int | The number of data pages that were read on disk |

# 2.2 Enumeration Types

The following enumeration types can be used with Altibase Monitoring API applications.

## 2.2.1 enum ABIPropType

This enumeration type is used with the ABISetProperty function to specify the user name and user password for connecting to an Altibase server.

This enumeration type has the following elements.

| Element | Description |
|---|---|
| ABI_USER | Used to specify the user name |
| ABI_PASSWD | Used to specify the user password |
| ABI_LOGFILE | Used to specify the file that stores the error messages that occur in Altibase Monitoring API |

## 2.3 Considerations

Almost all Altibase Monitoring API functions take the above data structures as arguments. This section discusses what you should consider when taking these data structures as arguments.

In an application, you need to declare a pointer variable to a data structure and pass this pointer's address value (a double pointer) to an Altibase Monitoring API function. The function allocates memory on heap to the pointer and sets it to the record fetched from the database, and then returns the result set to the application.

Since Altibase Monitoring API functions manage memory for data structures used with Altibase Monitoring API, the application should not directly allocate memory to a pointer in the data structure or deallocate memory returned as the result of a function.

As shown in the following sample code, a pointer in an ABIVSession data structure should be only declared and memory should not be allocated to sVSession. Moreover, if a result value is referenced from sVSession after a function has been executed, only as many array elements as the number of rows in the result set can be accessed.

```
ABIVSession *sVSession;
int sRowCount;

sRowCount = ABIGetVSession(    &sVSession, 0 );

/* reference the results selected from sVSession */
for (int i=0; i<sRowCount; i++)
{
      /* sVSession[i].mID; */
      /* sVSession[i].mTransID; */
}
```

# 3 Functions

This chapter discusses Altibase Monitoring API functions. The following information is provided:

- Function Name
- Syntax: Function Prototype in C
- Arguments: Data Type, Input/Output, Description
- Return Values
- Description: Usage and Considerations
- Example

# 3.1 ABIInitialize

## 3.1.1 Syntax

int ABIInitialize ( void );

## 3.1.2 Return Values

If successful, returns 0; otherwise, returns an error code.

## 3.1.3 Description

This function needs to be initially invoked to use Altibase Monitoring API. It performs initialization operations such as setting the connection to the Altibase server.

## 3.1.4 Example

```
if( ABIInitialize( ) != 0 )
{
      /* ... error handling ... */
}
```

# 3.2 ABIFinalize

## 3.2.1 Syntax

int ABIFinalize ( void );

## 3.2.2 Return Values

If successful, returns 0; otherwise, returns an error code.

## 3.2.3 Description

This function needs to be invoked to close Altibase Monitoring API. It performs operations such as freeing memory that has been allocated while using Altibase Monitoring API and disconnecting from the Altibase server.

## 3.2.4 Example

```
if( ABIFinalize( ) != 0 )
{
      /* ... error handling ... */
}
```

# 3.3 ABISetProperty

## 3.3.1 Syntax

```
int ABISetProperty (
      ABIPropType          aPropType,
      const char           *aPropValue );
```

## 3.3.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIPropType | *aPropType* | Input | Specifies the name of the property to be set. One of the following can be used: ABI_USER, ABI_PASSWD, ABI_LOGFILE |
| const char * | *aPropValue* | Input | The value of the property to be set |

## 3.3.3 Return Values

If successful, returns 0; otherwise, returns an error code.

## 3.3.4 Description

This function specifies the user name and user password to connect to the Altibase server, and the log file path. On omission, the default values are SYS, MANAGER, and altibaseMonitor.log, respectively.

Error messages that occur in Altibase Monitoring API are written to log files. If the path is omitted and only the file name is specified, a log file is created in the path wherein the application runs

## 3.3.5 Example

```
if( ABISetProperty( ABI_USER, "SYS" ) != 0 )
{
      /* ... error handling ... */
}
```

# 3.4 ABICheckConnection

## 3.4.1 Syntax

int ABICheckConnection ( );

## 3.4.2 Return Values

If the connection status is normal, returns 0; otherwise, returns -1.

## 3.4.3 Description

This function checks the connection between the Altibase server and Altibase Monitoring API.

## 3.4.4 Example

```
if( ABICheckConnection( ) != -1 )
        /* Select performance view */
}
else
{
        /* ... error handling ... */
}
```

# 3.5 ABIGetVSession

## 3.5.1 Syntax

```
int ABIGetVSession (
    ABIVSession          **aHandle,
    unsigned int         aExecutingOnly );
```

## 3.5.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSession** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| unsigned int | *aExecutingOnly* | Input | 0: Selects all sessions<br>1: Selects only active sessions |

## 3.5.3 Return Values

IIf successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.5.4 Description

This function selects the V$SESSION performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSession type that points to an array that stores the result set) is returned.

## 3.5.5 Example

Please refer to sample_1.c for an application related to this function.

```
ABIVSession *sVSession;
ABIVSession *sVSessionActiveOnly;
int sRowCount;
int sRowCountActiveOnly;

/* Select all sessions */
sRowCount = ABIGetVSession(    &sVSession, 0 );
```

```
/* Select only active sessions */
sRowCountActiveOnly = ABIGetVSession( &sVSessionActiveOnly, 1 );
```

# 3.6 ABIGetVSessionBySID

## 3.6.1 Syntax

```
int ABIGetVSessionBySID (
      ABIVSession              **aHandle,
      int                        aSessionID );
```

## 3.6.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSession** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| int | *aSessionID* | Input | The session ID to be selected |

## 3.6.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.6.4 Description

This function selects the V$SESSION performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSession type that points to an array that stores the result set) is returned.

## 3.6.5 Example

Please refer to sample_1.c for an application related to this function.

```
ABIVSession *sVSession;
int sRowCount;

sRowCount = ABIGetVSessionBySID(    &sVSession, 1 );
```

# 3.7 ABIGetVSysstat

## 3.7.1 Syntax

```
int ABIGetVSysstat (
      ABIVSysstat                **aHandle );
```

## 3.7.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSysstat ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.7.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.7.4 Description

This function selects the V$SYSSTAT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSysstat type that points to an array that stores the result set) is returned.

## 3.7.5 Example

Please refer to sample_3.c for an application related to this function.

```
ABIVSysstat *sVSysstat;
int sRowCount;

sRowCount = ABIGetVSysstat( &sVSysstat );
```

# 3.8 ABIGetVSesstat

## 3.8.1 Syntax

```
int ABIGetVSesstat (
      ABIVSesstat          **aHandle,
      unsigned int         aExecutingOnly );
```

## 3.8.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSesstat ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| unsigned int | *aExecutingOnly* | Input | 0: Selects all sessions<br>1: Selects only active sessions |

## 3.8.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.8.4 Description

This function selects the V$SESSTAT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSesstat type that points to an array that stores the result set) is returned.

## 3.8.5 Example

Please refer to sample_3.c for an application related to this function.

```
ABIVSesstat *sVSesstat;
int sRowCount;

sRowCount = ABIGetVSesstat( &sVSesstat );
```

# 3.9 ABIGetVSesstatBySID

## 3.9.1 Syntax

```
int ABIGetVSesstatBySID (
    ABIVSesstat          **aHandle,
    int                    aSessionID );
```

## 3.9.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSesstat ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| int | *aSessionID* | Input | The session ID to be selected |

## 3.9.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.9.4 Description

This function selects statistics about a certain session from the V$SESSTAT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSesstat type that points to an array that stores the result set) is returned.

## 3.9.5 Example

Please refer to sample_3.c for an application related to this function.

```
ABIVSesstat *sVSesstat;
int sRowCount;

/* Select session whose ID is 1 */
sRowCount = ABIGetVSesstatBySID ( &sVSesstat, 1 );
```

# 3.10 ABIGetStatName

## 3.10.1 Syntax

```
int ABIGetStatName (
      ABIStatName              **aHandle );
```

## 3.10.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIStatName ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.10.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.10.4 Description

This function selects the values of the fixed columns, SEQNUM and NAME, from the V$SESSTAT or V$SYSSTAT performance view.

## 3.10.5 Example

Please refer to sample_3.c for an application related to this function.

```
ABIStatName *sStatName;
int sRowCount;

sRowCount = ABIGetStatName( &sStatName );
```

# 3.11 ABIGetVSystemEvent

## 3.11.1 Syntax

```
int ABIGetVSystemEvent (
      ABIVSystemEvent              **aHandle );
```

## 3.11.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSystemEvent ** | aHandle | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.11.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.11.4 Description

This function selects the V$SYSTEM_EVENT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSystemEvent type that points to an array that stores the result set) is returned.

## 3.11.5 Example

Please refer to sample_4.c for an application related to this function.

```
AABIVSystemEvent *sVSystemEvent;
int sRowCount;

sRowCount = ABIGetVSystemEvent( &sVSystemEvent);
```

# 3.12 ABIGetVSessionEvent

## 3.12.1 Syntax

```
int ABIGetVSessionEvent (
      ABIVSessionEvent              **aHandle );
```

## 3.12.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSessionEvent ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.12.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.12.4 Description

This function selects the V$SESSION_EVENT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSessionEvent type that points to an array that stores the result set) is returned.

## 3.12.5 Example

Please refer to sample_4.c for an application related to this function.

```
ABIVSessionEvent *sVSessionEvent;
int sRowCount;

sRowCount = ABIGetVSessionEvent( &sVSessionEvent);
```

# 3.13 ABIGetVSessionEventBySID

## 3.13.1 Syntax

```
int ABIGetVSessionEventBySID (
        ABIVSessionEvent        **aHandle,
        int                     aSessionID );
```

## 3.13.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSessionEvent ** | aHandle | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| int | aSessionID | Input | The session ID to be selected |

## 3.13.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.13.4 Description

This function selects statistics about wait events for certain sessions from the V$SESSION_EVENT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSessionEvent type that points to an array that stores the result set) is returned.

## 3.13.5 Example

Please refer to sample_4.c for an application related to this function.

```
ABIVSessionEvent *sVSessionEvent;
int sRowCount;

sRowCount = ABIGetVSessionEventBySID( &sVSessionEvent, 1);
```

# 3.14 ABIGetEventName

## 3.14.1 Syntax

```
int ABIGetEventName (
      ABIEventName              **aHandle );
```

## 3.14.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIEventName ** | aHandle | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.14.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.14.4 Description

This function selects the values of the fixed columns, EVENT_ID, EVENT, WAIT_CLASS_ID, and WAIT_CLASS, from the V$SYSTEM_EVENT or V$SESSION_EVENT performance view.

## 3.14.5 Example

Please refer to sample_4.c for an application related to this function.

```
ABIEventName *sEventName;
int sRowCount;

sRowCount = ABIGetEventName( &sEventName);
```

# 3.15 ABIGetVSessionWait

## 3.15.1 Syntax

```
int ABIGetVSessionWait (
      ABIVSessionWait              **aHandle );
```

## 3.15.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSessionWait ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.15.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.15.4 Description

This function selects the V$SESSION_WAIT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSessionWait type that points to an array that stores the result set) is returned.

## 3.15.5 Example

Please refer to sample_8.c for an application related to this function.

```
ABIVSessionWait *sVSessionWait;
int sRowCount;

sRowCount = ABIGetVSessionWait( &sVSessionWait);
```

# 3.16 ABIGetVSessionWaitBySID

## 3.16.1 Syntax

```
int ABIGetVSessionWaitBySID (
    ABIVSessionWait          **aHandle,
    int                      aSessionID );
```

## 3.16.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIVSessionWait ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| int | *aSessionID* | Input | The session ID to be selected |

## 3.16.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.16.4 Description

This function selects information about wait events for certain sessions from the V$SESSION_WAIT performance view. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIVSessionWait type that points to an array that stores the result set) is returned.

## 3.16.5 Example

Please refer to sample_8.c for an application related to this function.

```
ABIVSessionWait *sVSessionWait;
int sRowCount;

sRowCount = ABIGetVSessionWaitBySID( &sVSessionWait, 1);
```

# 3.17 ABIGetSqlText

## 3.17.1 Syntax

```
int ABIGetSqlText (
      ABISqlText          **aHandle,
      int                 astmtID );
```

## 3.17.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABISqlText ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |
| int | *astmtID* | Input | The session ID to be selected |

## 3.17.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.17.4 Description

This is a function used for viewing SQL statement, the start time of a query, and checking whether or not to execute a query that a statement is executing through the statement identifier.

## 3.17.5 Example

Please refer to sample_5.c for an application related to this function.

```
ABISqlText *sSqlText;
int sRet;

/* Selects the SQL statement of the statement whose ID is 2 */
sRet = ABIGetSqlText( &sSqlText, 2 );
```

# 3.18 ABIGetLockPairBetweenSessions

## 3.18.1 Syntax

```
int ABIGetLockPairBetweenSessions (
      ABILockPair                **aHandle );
```

## 3.18.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABILockPair ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.18.3 Return Values

If successful, returns the number of rows in the result set that *aHandle* points to; otherwise, returns an error code.

## 3.18.4 Description

This function selects the session that is holding a lock and the session that is waiting to acquire that lock. If this function executes successfully, the *aHandle* pointer (a pointer of the ABILockPair type that points to an array that stores the result set) is returned.

## 3.18.5 Example

Please refer to sample_5.c for an application related to this function.

```
ABILockPair *sLockPair;
int sRowCount;

sRowCount = ABIGetLockPairBetweenSessions( &sLockPair );
```

# 3.19 ABIGetDBInfo

## 3.19.1 Syntax

```
int ABIGetDBInfo (
      ABIDBInfo              **aHandle );
```

## 3.19.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIDBInfo ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.19.3 Return Values

If successful, returns 0; otherwise, returns an error code.

## 3.19.4 Description

This function selects the database name and its version number. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIDBInfo type that points to an array that stores the result set) is returned.

## 3.19.5 Example

Please refer to sample_6.c for an application related to this function.

```
ABIDBInfo *sDBInfo;
int sRet;

sRet = ABIGetDBInfo( &sDBInfo );
```

# 3.20 ABIGetReadCount

## 3.20.1 Syntax

```
int ABIGetReadCount (
      ABIReadCount              **aHandle );
```

## 3.20.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| ABIReadCount ** | *aHandle* | Output | The pointer which retrieves the memory address of the structure array that stores the result set |

## 3.20.3 Return Values

If successful, returns 0; otherwise, returns an error code.

## 3.20.4 Description

This function selects the number of times data pages were read on the Altibase server. If this function executes successfully, the *aHandle* pointer (a pointer of the ABIReadCount type that points to an array that stores the result set) is returned.

## 3.20.5 Example

Please refer to sample_6.c for an application related to this function.

```
ABIReadCount *sReadCount;
int sRet;

sRet = ABIGetReadCount( &sReadCount);
```

# 3.21 ABIGetSessionCount

## 3.21.1 Syntax

```
int ABIGetSessionCount (
      unsigned int                **aExecutingOnly );
```

## 3.21.2 Argument

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| unsigned int | *aExecutingOnly* | Input | 0: Selects all sessions<br>1: Selects only active sessions |

## 3.21.3 Return Values

If successful, returns the total number of sessions in the Altibase server; otherwise, returns an error code.

## 3.21.4 Description

This function selects the total number of sessions currently existing in the Altibase server or the number of active sessions.

## 3.21.5 Example

Please refer to sample_2.c for an application related to this function.

```
int sSessionCount;
int sActiveSessionCount;

/* Selects the total number of sessions */
sSessionCount = ABIGetSessionCount( 0 );
/* Selects the number of active sessions */
sActiveSessionCount    = ABIGetSessionCount( 1 );
```

# 3.22 ABIGetMaxClientCount

## 3.22.1 Syntax

```
int ABIGetMaxClientCount ( );
```

## 3.22.2 Return Values

If successful, returns the maximum number of clients allowed to connect to the Altibase server; otherwise, returns an error code.

## 3.22.3 Description

This function selects the maximum number of clients allowed to connect to the Altibase server. The selected value corresponds to the value set for the MAX_CLIENT property in the altibase.properties file for the Altibase server.

## 3.22.4 Example

Please refer to sample_2.c for an application related to this function.

```
int sMaxClientCount;

sMaxClientCount = ABIGetMaxClientCount( );
```

# 3.23 ABIGetLockWaitSessionCount

## 3.23.1 Syntax

```
int ABIGetLockWaitSessionCount ( );
```

## 3.23.2 Return Values

If successful, returns the number of sessions waiting to acquire locks; otherwise, returns an error code.

## 3.23.3 Description

This function selects the number of sessions waiting to acquire locks on the Altibase server.

## 3.23.4 Example

Please refer to sample_5.c for an application related to this function.

```
int sLockWaitSessionCount;

sLockWaitSessionCount = ABIGetLockWaitSessionCount( );
```

# 3.24 ABIGetErrorMessage

## 3.24.1 Syntax

```
void ABIGetErrorMessage (
        int                     aErrCode,
        const char              *aErrMsg );
```

## 3.24.2 Arguments

| Data Type | Argument | Input/Output | Description |
|---|---|---|---|
| Int | *aErrCode* | Input | Error code |
| const char * | *aErrMsg* | Output | The buffer pointer that retrieves the error message |

## 3.24.3 Description

This function selects an error message by its error code.

## 3.24.4 Example

Please refer to sample_9.c for an application related to this function.

```
ABIVSession *sVSession;
int             sErrCode;
const char    *sErrMsg;

sErrCode = ABIGetVSession( &sVSession, 1 );
if( sErrCode < 0 )
{
        ABIGetErrorMessage( sErrCode, &sErrMsg );
}
```

# 4 Sample Programs

This chapter provides sample programs in C that were written using Altibase Monitoring API.

# 4.1 Makefile

This is an example Makefile to compile the sample programs provided in this chapter. It uses the altibase_env.mk file included in the Altibase package.

```
include $(ALTIBASE_HOME)/install/altibase_env.mk
SRCS = $(wildcard *.c)
OBJS = $(SRCS:.c=.$(OBJEXT))
BINS = $(SRCS:.c=$(BINEXT))

all : $(BINS)

sample_1 : sample_1.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_2 : sample_2.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_3 : sample_3.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_4 : sample_4.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_5 : sample_5.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_6 : sample_6.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_7 : sample_7.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_8 : sample_8.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
sample_9 : sample_9.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT)
$(LIBOPT)odbccli$(LIBAFT) $(LIBS)
clean :
        $(RM) $(OBJS) $(BINS) *.log
```

# 4.2 sample_1.c

This sample program uses the ABIGetVSession and ABIGetVSessionBySID functions to select the V$SESSION performance view.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printVSession( ABIVSession *aVSession, int aRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABIVSession *sVSession = NULL, *sVSessionBySID = NULL;
    int         sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        /*
            IF the second argument sets to
            0 - Return all session information.
            1 - Return executing session information only.
         */
        // Test ABIGetVSession
        sRc = ABIGetVSession( &sVSession, 0 );
        if( sRc >= 0 )
        {
            printf( "**********************************\n" );
            printf( "*          V$Session             *\n" );
            printf( "**********************************\n" );
            printVSession( sVSession, sRc );

            sVSession = NULL;
            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetVSession [ Active Only ]
        sRc = ABIGetVSession( &sVSession, 1 );
        if( sRc >= 0 )
        {
            printf( "************************************************\n" );
            printf( "*          V$Session [ Active Only ]           *\n" );
            printf( "************************************************\n" );
            printVSession( sVSession, sRc );

            sRc = 0;
        }
```

```
            else
            {
                // Error handling
                errorHandling( sRc );
            }

            // Test ABIGetVSessionBySID
            sRc = ABIGetVSessionBySID( &sVSessionBySID, sVSession[0].mID );
            if( sRc >= 0 )
            {
                printf( "*************************************************\n" );
                printf( "*              V$Session [ specified SID ]              *\n" );
                printf( "*************************************************\n" );
                printVSession( sVSessionBySID, sRc );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void printVSession( ABIVSession *aVSession, int aRowCount )
{
        int sI;

        for( sI = 0; sI < aRowCount; sI++ )
        {
            printf( "ID : %d\n", aVSession[sI].mID );
            printf( "TRANS_ID : %lld\n", aVSession[sI].mTransID );
            printf( "TASK_STATE : %s\n", aVSession[sI].mTaskState );
            printf( "COMM_NAME : %s\n", aVSession[sI].mCommName );
            printf( "XA_SESSION_FLAG : %d\n", aVSession[sI].mXASessionFlag );
            printf( "XA_ASSOCIATE_FLAG : %d\n", aVSession[sI].mXAAssociateFlag );
            printf( "QUERY_TIME_LIMIT : %d\n", aVSession[sI].mQueryTimeLimit );
            printf( "DDL_TIME_LIMIT : %d\n", aVSession[sI].mDdlTimeLimit );
            printf( "FETCH_TIME_LIMIT : %d\n", aVSession[sI].mFetchTimeLimit );
            printf( "UTRANS_TIME_LIMIT : %d\n", aVSession[sI].mUTransTimeLimit );
            printf( "IDLE_TIME_LIMIT : %d\n", aVSession[sI].mIdleTimeLimit );
            printf( "IDLE_START_TIME : %d\n", aVSession[sI].mIdleStartTime );
            printf( "ACTIVE_FLAG : %d\n", aVSession[sI].mActiveFlag );
            printf( "OPENED_STMT_COUNT : %d\n", aVSession[sI].mOpenedStmtCount );
            printf( "CLIENT_PACKAGE_VERSION : %s\n", aVSession[sI].mClientPackageVersion );
            printf( "CLIENT_PROTOCOL_VERSION : %s\n", aVSession[sI].mClientProtocolVersion );
            printf( "CLIENT_PID : %lld\n", aVSession[sI].mClientPID );
            printf( "CLIENT_TYPE : %s\n", aVSession[sI].mClientType );
            printf( "CLIENT_APP_INFO : %s\n", aVSession[sI].mClientAppInfo );
            printf( "CLIENT_NLS : %s\n", aVSession[sI].mClientNls );
            printf( "DB_USERNAME : %s\n", aVSession[sI].mDBUserName );
            printf( "DB_USERID : %d\n", aVSession[sI].mDBUserID );
```

```
            printf( "DEFAULT_TBSID : %lld\n", aVSession[sI].mDefaultTbsID );
            printf( "DEFAULT_TEMP_TBSID : %lld\n", aVSession[sI].mDefaultTempTbsID );
            printf( "SYSDBA_FLAG : %d\n", aVSession[sI].mSysDbaFlag );
            printf( "AUTOCOMMIT_FLAG : %d\n", aVSession[sI]. mAutoCommitFlag );
            printf( "SESSION_STATE : %s\n", aVSession[sI].mSessionState );
            printf( "ISOLATION_LEVEL : %d\n", aVSession[sI].mIsolationLevel );
            printf( "REPLICATION_MODE : %d\n", aVSession[sI].mReplicationMode );
            printf( "TRANSACTION_MODE : %d\n", aVSession[sI].mTransactionMode );
            printf( "COMMIT_WRITE_WAIT_MODE : %d\n", aVSession[sI].mCommitWriteWaitMode );
            printf( "OPTIMIZER_MODE : %d\n", aVSession[sI].mOptimizerMode );
            printf( "HEADER_DISPLAY_MODE : %d\n", aVSession[sI].mHeaderDisplayMode );
            printf( "CURRENT_STMT_ID : %d\n", aVSession[sI].mCurrentStmtID );
            printf( "STACK_SIZE : %d\n", aVSession[sI].mStackSize );
            printf( "DEFAULT_DATE_FORMAT : %s\n", aVSession[sI].mDefaultDateFormat );
            printf( "TRX_UPDATE_MAX_LOGSIZE : %lld\n", aVSession[sI].mTrxUpdateMaxLogSize );
            printf( "PARALLEL_DML_MODE : %d\n", aVSession[sI].mParallelDmlMode );
            printf( "LOGIN_TIME : %d\n", aVSession[sI].mLoginTime );
            printf( "FAILOVER_SOURCE : %s\n\n", aVSession[sI].mFailoverSource );
        }
        printf( "\n" );
}

void errorHandling( int aErrCode )
{
        const char *sErrMsg = NULL;

        ABIGetErrorMessage( aErrCode, &sErrMsg );
        printf( "%s\n\n\n", sErrMsg );
        exit(1);
}
```

# 4.3 sample_2.c

This sample program uses the ABIGetSessionCount function to select the total number of existing sessions and active sessions on the Altibase server.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void errorHandling( int aErrCode );

int main()
{
    int             sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        /*
          IF the argument sets to
          0 - Return all session count.
          1 - Return executing session count only.
         */
        // Test ABIGetSessionCount
        sRc = ABIGetSessionCount( 0 );
        if( sRc >= 0 )
        {
            printf( "**************************************\n" );
            printf( "*                Session Count               *\n" );
            printf( "**************************************\n" );
            printf( "Session Count : %d\n\n\n", sRc );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetSessionCount [ Active Only ]
        sRc = ABIGetSessionCount( 1 );
        if( sRc >= 0 )
        {
            printf( "*****************************************************\n" );
            printf( "*                Session Count [ Active Only ]              *\n" );
            printf( "*****************************************************\n" );
            printf( "Session Count : %d\n\n\n", sRc );

            sRc = 0;
        }
        else
        {
            // Error handling
```

```
                errorHandling( sRc );
            }

            // Test ABIGetMaxClientCount
            sRc = ABIGetMaxClientCount();
            if( sRc >= 0 )
            {
                printf( "****************************************\n" );
                printf( "*                Max Client Count                *\n" );
                printf( "****************************************\n" );
                printf( "Max Client Count : %d\n\n\n", sRc );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void errorHandling( int aErrCode )
{
        const char *sErrMsg = NULL;

        ABIGetErrorMessage( aErrCode, &sErrMsg );
        printf( "%s\n\n\n", sErrMsg );
        exit(1);
}
```

# 4.4 sample_3.c

This sample program uses the ABIGetStatName, ABIGetVSysstat, ABIGetVSesstat, and ABIGetVSesstatBySID functions to select statistics on the Altibase system and its sessions.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printVSysstat( ABIVSysstat *aVSysstat, int aRowCount, ABIStatName *aStatName );
void printVSesstat( ABIVSesstat *aVSesstat, int aRowCount, ABIStatName *aStatName, int
aStatNameRowCount );
void printStatName( ABIStatName *aStatName, int aStatNameRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABIVSysstat *sVSysstat = NULL;
    ABIVSesstat *sVSesstat = NULL;
    ABIStatName *sStatName = NULL;
    int          sStatNameRowCount = 0;
    int          sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        // Test ABIGetStatName
        sStatNameRowCount = ABIGetStatName( &sStatName );
        if( sStatNameRowCount >= 0 )
        {
            printf( "*********************************\n" );
            printf( "*              StatName             *\n" );
            printf( "*********************************\n" );
            printStatName( sStatName, sStatNameRowCount );
        }
        else
        {
            // Error handling
            errorHandling( sStatNameRowCount );
        }

        // Test ABIGetVSysstat
        sRc = ABIGetVSysstat( &sVSysstat );
        if( sRc >= 0 )
        {
            printf( "*********************************\n" );
            printf( "*              V$Sysstat            *\n" );
            printf( "*********************************\n" );
            printVSysstat( sVSysstat, sRc, sStatName );

            sRc = 0;
        }
        else
        {
```

```
                // Error handling
                errorHandling( sRc );
            }

            // Test ABIGetVSesstat
            sRc = ABIGetVSesstat( &sVSesstat, 0 );
            if( sRc >= 0 )
            {
                printf( "*********************************\n" );
                printf( "*                V$Sesstat                *\n" );
                printf( "*********************************\n" );
                printVSesstat( sVSesstat, sRc, sStatName, sStatNameRowCount );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }

            // Test ABIGetVSesstatBySID
            sRc = ABIGetVSesstatBySID( &sVSesstat, sVSesstat[0].mSID );
            if( sRc >= 0 )
            {
                printf( "*****************************************************\n" );
                printf( "*                V$Sesstat [ specified SID ]                *\n" );
                printf( "*****************************************************\n" );
                printVSesstat( sVSesstat, sRc, sStatName, sStatNameRowCount );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void printVSysstat( ABIVSysstat *aVSysstat, int aRowCount, ABIStatName *aStatName )
{
        int sI;

        for( sI = 0; sI < aRowCount; sI++ )
        {
            printf( "SEQNUM : %d\n", aStatName[sI].mSeqNum );
            printf( "NAME : %s\n", aStatName[sI].mName );
            printf( "VALUE : %lld\n\n", aVSysstat[sI].mValue );
        }
        printf( "\n" );
}
```

```
void printVSesstat( ABIVSesstat *aVSesstat, int aRowCount, ABIStatName *aStatName, int
aStatNameRowCount )
{
    int sI, sJ;

    for( sI = 0; sI < aRowCount; sI++ )
    {
        sJ = sI % aStatNameRowCount;

        printf( "SID : %d\n", aVSesstat[sI].mSID );
        printf( "SEQNUM : %d\n", aStatName[sJ].mSeqNum );
        printf( "NAME : %s\n", aStatName[sJ].mName );
        printf( "VALUE : %lld\n\n", aVSesstat[sI].mValue );
    }
    printf( "\n" );
}

void printStatName( ABIStatName *aStatName, int aStatNameRowCount )
{
    int sI;

    for( sI = 0; sI < aStatNameRowCount; sI++ )
    {
        printf( "SEQNUM : %d\n", aStatName[sI].mSeqNum );
        printf( "NAME : %s\n\n", aStatName[sI].mName );
    }
    printf( "\n" );
}

void errorHandling( int aErrCode )
{
    const char *sErrMsg = NULL;

    ABIGetErrorMessage( aErrCode, &sErrMsg );
    printf( "%s\n\n\n", sErrMsg );
    exit(1);
}
```

# 4.5 sample_4.c

This sample program uses the ABIGetEventName, ABIGetVSystemEvent, ABIGetVSessionEvent, and ABIGetVSessionEventBySID functions to select statistics on waits events of the Altibase system and its sessions.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printVSystemEvent( ABIVSystemEvent *aVSystemEvent, int aRowCount, ABIEventName *aEventName );
void printVSessionEvent( ABIVSessionEvent *aVSessionEvent, int aRowCount, ABIEventName *aEventName,
int aEventNameRowCount );
void printEventName( ABIEventName *aEventName, int aEventNameRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABIVSystemEvent    *sVSystemEvent = NULL;
    ABIVSessionEvent *sVSessionEvent = NULL;
    ABIEventName         *sEventName = NULL;
    int                  sEventNameRowCount = 0;
    int                  sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        // Test ABIGetEventName
        sEventNameRowCount = ABIGetEventName( &sEventName );
        if( sEventNameRowCount >= 0 )
        {
            printf( "**********************************\n" );
            printf( "*                Event Name                *\n" );
            printf( "**********************************\n" );
            printEventName( sEventName, sEventNameRowCount );
        }
        else
        {
            // Error handling
            errorHandling( sEventNameRowCount );
        }

        // Test ABIGetVSystemEvent
        sRc = ABIGetVSystemEvent( &sVSystemEvent );
        if( sRc >= 0 )
        {
            printf( "****************************************\n" );
            printf( "*                V$System_Event                *\n" );
            printf( "****************************************\n" );
            printVSystemEvent( sVSystemEvent, sRc, sEventName );

            sRc = 0;
        }
```

```
            else
            {
                // Error handling
                errorHandling( sRc );
            }

            // Test ABIGetVSessionEvent
            sRc = ABIGetVSessionEvent( &sVSessionEvent );
            if( sRc >= 0 )
            {
                printf( "****************************************\n" );
                printf( "*                V$Session_Event                *\n" );
                printf( "****************************************\n" );
                printVSessionEvent( sVSessionEvent, sRc, sEventName, sEventNameRowCount );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }

            // Test ABIGetVSessionEventBySID
            sRc = ABIGetVSessionEventBySID( &sVSessionEvent, sVSessionEvent[0].mSID );
            if( sRc >= 0 )
            {
                printf( "************************************************************\n" );
                printf( "*                V$Session_Event [ specified SID ]                *\n" );
                printf( "************************************************************\n" );
                printVSessionEvent( sVSessionEvent, sRc, sEventName, sEventNameRowCount );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void printVSystemEvent( ABIVSystemEvent *aVSystemEvent, int aRowCount, ABIEventName *aEventName )
{
        int sI;

        for( sI = 0; sI < aRowCount; sI++ )
        {
            printf( "EVENT : %s\n", aEventName[sI].mEvent );
            printf( "TOTAL_WAITS : %lld\n", aVSystemEvent[sI].mTotalWaits );
            printf( "TOTAL_TIMEOUTS : %lld\n", aVSystemEvent[sI].mTotalTimeOuts );
            printf( "TIME_WAITED : %lld\n", aVSystemEvent[sI].mTimeWaited );
            printf( "AVERAGE_WAIT : %lld\n", aVSystemEvent[sI].mAverageWait );
```

```
            printf( "TIME_WAITED_MICRO : %lld\n", aVSystemEvent[sI].mTimeWaitedMicro );
            printf( "EVENT_ID : %d\n", aEventName[sI].mEventID );
            printf( "WAIT_CLASS_ID : %d\n", aEventName[sI].mWaitClassID );
            printf( "WAIT_CLASS : %s\n\n", aEventName[sI].mWaitClass );
        }
        printf( "\n" );
}

void printVSessionEvent( ABIVSessionEvent *aVSessionEvent, int aRowCount, ABIEventName *aEventName,
int aEventNameRowCount )
{
        int sI, sJ;

        for( sI = 0; sI < aRowCount; sI++ )
        {
            sJ = sI % aEventNameRowCount;

            printf( "SID : %d\n", aVSessionEvent[sI].mSID );
            printf( "EVENT : %s\n", aEventName[sJ].mEvent );
            printf( "TOTAL_WAITS : %lld\n", aVSessionEvent[sI].mTotalWaits );
            printf( "TOTAL_TIMEOUTS : %lld\n", aVSessionEvent[sI].mTotalTimeOuts );
            printf( "TIME_WAITED : %lld\n", aVSessionEvent[sI].mTimeWaited );
            printf( "AVERAGE_WAIT : %lld\n", aVSessionEvent[sI].mAverageWait );
            printf( "MAX_WAIT : %lld\n", aVSessionEvent[sI].mMaxWait );
            printf( "TIME_WAITED_MICRO : %lld\n", aVSessionEvent[sI].mTimeWaitedMicro );
            printf( "EVENT_ID : %d\n", aEventName[sJ].mEventID );
            printf( "WAIT_CLASS_ID : %d\n", aEventName[sJ].mWaitClassID );
            printf( "WAIT_CLASS : %s\n\n", aEventName[sJ].mWaitClass );
        }
        printf( "\n" );
}

void printEventName( ABIEventName *aEventName, int aEventNameRowCount )
{
        int sI;

        for( sI = 0; sI < aEventNameRowCount; sI++ )
        {
            printf( "EVENT_ID : %d\n", aEventName[sI].mEventID );
            printf( "EVENT : %s\n", aEventName[sI].mEvent );
            printf( "WAIT_CLASS_ID : %d\n", aEventName[sI].mWaitClassID );
            printf( "WAIT_CLASS : %s\n\n", aEventName[sI].mWaitClass );
        }
        printf( "\n" );
}

void errorHandling( int aErrCode )
{
        const char *sErrMsg = NULL;

        ABIGetErrorMessage( aErrCode, &sErrMsg );
        printf( "%s\n\n\n", sErrMsg );
        exit(1);
}
```

# 4.6 sample_5.c

This sample program uses the ABIGetSqlText, ABIGetLockPairBetweenSessions, and ABIGetLockWaitSessionCount functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printSqlText( ABISqlText *aSqlText );
void printLockPairBetweenSessions( ABILockPair *aLockPair, int aRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABISqlText    *sSqlText = NULL;
    ABILockPair *sLockPair = NULL;
    int                 sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        // Test ABIGetSqlText
        sRc = ABIGetSqlText( &sSqlText, 2 );
        if( sRc >= 0 )
        {
            printf( "*********************************\n" );
            printf( "*                SQL Text                *\n" );
            printf( "*********************************\n" );
            printSqlText( sSqlText );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetLockPairBetweenSessions
        sRc = ABIGetLockPairBetweenSessions( &sLockPair );
        if( sRc >= 0 )
        {
            printf( "*************************************************\n" );
            printf( "*                Lock Pair Between Sessions                *\n" );
            printf( "*************************************************\n" );
            printLockPairBetweenSessions( sLockPair, sRc );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
```

```
            }

            // Test ABIGetLockWaitSessionCount
            sRc = ABIGetLockWaitSessionCount();
            if( sRc >= 0 )
            {
                printf( "*********************************************\n" );
                printf( "*                Lock Wait Session Count                *\n" );
                printf( "*********************************************\n" );
                printf( "Lock Wait Session Count : %d\n\n\n", sRc );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void printSqlText( ABISqlText *aSqlText )
{
        printf( "SQL TEXT : %s\n", aSqlText->mSqlText );
        printf( "TEXT LENGTH : %d\n\n\n", aSqlText->mTextLength );
        printf( "QUERY START TIME : %d\n\n\n", aSqlText->mQueryStartTime );
        printf( "EXECUTE FLAG : %d\n\n\n", aSqlText->mExecuteFlag );
}

void printLockPairBetweenSessions( ABILockPair *aLockPair, int aRowCount )
{
        int sI;

        for( sI = 0; sI < aRowCount; sI++ )
        {
            printf( "HOLDER : %d,\t\tWAITER : %d\n", aLockPair[sI].mHolderSID, aLockPair[sI].mWaiterSID );
        }
        printf( "\n\n" );
}

void errorHandling( int aErrCode )
{
        const char *sErrMsg = NULL;

        ABIGetErrorMessage( aErrCode, &sErrMsg );
        printf( "%s\n\n\n", sErrMsg );
        exit(1);
}
```

# 4.7 sample_6.c

This sample program uses the ABIGetDBInfo and ABIGetReadCount functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printSqlText( ABISqlText *aSqlText );
void printLockPairBetweenSessions( ABILockPair *aLockPair, int aRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABISqlText    *sSqlText = NULL;
    ABILockPair *sLockPair = NULL;
    int                 sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        // Test ABIGetSqlText
        sRc = ABIGetSqlText( &sSqlText, 2 );
        if( sRc >= 0 )
        {
            printf( "********************************\n" );
            printf( "*                SQL Text                *\n" );
            printf( "********************************\n" );
            printSqlText( sSqlText );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetLockPairBetweenSessions
        sRc = ABIGetLockPairBetweenSessions( &sLockPair );
        if( sRc >= 0 )
        {
            printf( "************************************************\n" );
            printf( "*                Lock Pair Between Sessions                *\n" );
            printf( "************************************************\n" );
            printLockPairBetweenSessions( sLockPair, sRc );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }
```

```
            // Test ABIGetLockWaitSessionCount
            sRc = ABIGetLockWaitSessionCount();
            if( sRc >= 0 )
            {
                printf( "*********************************************\n" );
                printf( "*                Lock Wait Session Count                *\n" );
                printf( "*********************************************\n" );
                printf( "Lock Wait Session Count : %d\n\n\n", sRc );

                sRc = 0;
            }
            else
            {
                // Error handling
                errorHandling( sRc );
            }
        }
        else
        {
            // Exception handling
        }

        // Test ABIFinalize
        sRc = ABIFinalize();
        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        return 0;
}

void printSqlText( ABISqlText *aSqlText )
{
        printf( "SQL TEXT : %s\n", aSqlText->mSqlText );
        printf( "TEXT LENGTH : %d\n\n\n", aSqlText->mTextLength );
}

void printLockPairBetweenSessions( ABILockPair *aLockPair, int aRowCount )
{
        int sI;

        for( sI = 0; sI < aRowCount; sI++ )
        {
                printf( "HOLDER : %d,\t\tWAITER : %d\n", aLockPair[sI].mHolderSID, aLockPair[sI].mWaiterSID );
        }
        printf( "\n\n" );
}

void errorHandling( int aErrCode )
{
        const char *sErrMsg = NULL;

        ABIGetErrorMessage( aErrCode, &sErrMsg );
        printf( "%s\n\n\n", sErrMsg );
        exit(1);
}
```

# 4.8 sample_7.c

This sample program uses a global variable of the pthread_mutex_t type to synchronize the function calls of ABIGetVSession and ABIGetSessionCount.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <altibaseMonitor.h>

#define LOOP_COUNT 1000

pthread_mutex_t gMutex;

void *call_ABIGetVSession( void *aArgs );
void *call_ABIGetSessionCount( void *aArgs );
void errorHandling( int aErrCode );

int main()
{
    pthread_t sThread[2];
    int       sRc = 0;

    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    sRc = pthread_mutex_init( &gMutex, NULL );
    if( sRc != 0 )
    {
        printf( "Mutex init" );
        exit(1);
    }

    sRc = pthread_create( &( sThread[0] ), NULL, call_ABIGetVSession, NULL );
    if( sRc != 0 )
    {
        printf( "Create thread_1 [ Calling ABIGetVSession ]" );
        exit(1);
    }

    sRc = pthread_create( &( sThread[1] ), NULL, call_ABIGetSessionCount, NULL );
    if( sRc != 0 )
    {
        printf( "Create thread_2 [ Calling ABIGetSessionCount ]" );
        exit(1);
    }

    pthread_join( sThread[0], NULL );
    pthread_join( sThread[1], NULL );

    pthread_mutex_destroy( &gMutex );

    sRc = ABIFinalize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
```

```
    }

    return 0;
}

void *call_ABIGetVSession( void *aArgs )
{
    ABIVSession *sVSession = NULL;
    int         sRc = 0;
    int         sI;

    for( sI = 0; sI < LOOP_COUNT; sI++ )
    {
        pthread_mutex_lock( &gMutex );

        sRc = ABIGetVSession( &sVSession, 1 );

        pthread_mutex_unlock( &gMutex );

        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }

        sleep( 0.05 );
    }

    return NULL;
}

void *call_ABIGetSessionCount( void *aArgs )
{
    int sRc = 0;
    int sI;

    for( sI = 0; sI < LOOP_COUNT; sI++ )
    {
        pthread_mutex_lock( &gMutex );

        sRc = ABIGetSessionCount( 1 );

        pthread_mutex_unlock( &gMutex );

        if( sRc < 0 )
        {
            // Error handling
            errorHandling( sRc );
        }
    }

    return NULL;
}

void errorHandling( int aErrCode )
{
    const char *sErrMsg = NULL;

    ABIGetErrorMessage( aErrCode, &sErrMsg );
    printf( "%s\n\n\n", sErrMsg );
    exit(1);
}
```

# 4.9 sample_8.c

This sample program uses the ABIGetVSessionWait and ABIGetVSessionWaitBySID functions to select wait event information for sessions connected to the Altibase server.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printVSessionWait( ABIVSessionWait *aVSessionWait, int aRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABIVSessionWait *sVSessionWait = NULL, *sVSessionWaitBySID = NULL;
    int                     sRc = 0;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        /*
            IF the second argument sets to
            0 - Return all session information.
            1 - Return executing session information only.
         */
        // Test ABIGetVSessionWait
        sRc = ABIGetVSessionWait( &sVSessionWait );
        if( sRc >= 0 )
        {
            printf( "****************************************\n" );
            printf( "*                V$Session_Wait                *\n" );
            printf( "****************************************\n" );
            printVSessionWait( sVSessionWait, sRc );

            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetVSessionWaitBySID
        sRc = ABIGetVSessionWaitBySID( &sVSessionWaitBySID, sVSessionWait[0].mSID );
        if( sRc >= 0 )
        {
            printf( "************************************************************\n" );
            printf( "*                V$Session_Wait [ specified SID ]                *\n" );
            printf( "************************************************************\n" );
            printVSessionWait( sVSessionWaitBySID, sRc );

            sRc = 0;
        }
        else
```

```
        {
            // Error handling
            errorHandling( sRc );
        }
    }
    else
    {
        // Exception handling
    }

    // Test ABIFinalize
    sRc = ABIFinalize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    return 0;
}

void printVSessionWait( ABIVSessionWait *aVSessionWait, int aRowCount )
{
    int sI;

    for( sI = 0; sI < aRowCount; sI++ )
    {
        printf( "SID : %d\n", aVSessionWait[sI].mSID );
        printf( "SEQNUM : %d\n", aVSessionWait[sI].mSeqNum );
        printf( "P1 : %lld\n", aVSessionWait[sI].mP1 );
        printf( "P2 : %lld\n", aVSessionWait[sI].mP2 );
        printf( "P3 : %lld\n", aVSessionWait[sI].mP3 );
        printf( "WAIT_CLASS_ID : %d\n", aVSessionWait[sI].mWaitClassID );
        printf( "WAIT_TIME : %lld\n", aVSessionWait[sI].mWaitTime );
        printf( "SECOND_IN_TIME : %lld\n\n", aVSessionWait[sI].mSecondInTime );
    }
    printf( "\n" );
}

void errorHandling( int aErrCode )
{
    const char *sErrMsg = NULL;

    ABIGetErrorMessage( aErrCode, &sErrMsg );
    printf( "%s\n\n\n", sErrMsg );
    exit(1);
}
```

# 4.10 sample_9.c

This sample program uses the ABIGetErrorMessage function to select an error message by its error code.

```
#include <stdio.h>
#include <stdlib.h>
#include <altibaseMonitor.h>

void printErrorMessage( int aErrCode, const char *aErrMsg );
void errorHandling( int aErrCode );

int main()
{
    int             sI;
    int             sRc = 0;
    const char *sErrMsg = NULL;

    // Test ABIInitialize
    sRc = ABIInitialize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        printf( "**************************************\n" );
        printf( "*                Error Message               *\n" );
        printf( "**************************************\n" );

        for( sI = -21; sI < 0; sI++ )
        {
            // Test ABIGetErrorMessage
            ABIGetErrorMessage( sI, &sErrMsg );
            printErrorMessage( sI, sErrMsg );
        }
        printf( "\n" );
    }
    else
    {
        // Exception handling
    }

    // Test ABIFinalize
    sRc = ABIFinalize();
    if( sRc < 0 )
    {
        // Error handling
        errorHandling( sRc );
    }

    return 0;
}

void printErrorMessage( int aErrCode, const char *aErrMsg )
{
    printf( "Error Code : %d\n", aErrCode );
    printf( "%s\n\n", aErrMsg );
}
```

```
void errorHandling( int aErrCode )
{
	const char *sErrMsg = NULL;

	ABIGetErrorMessage( aErrCode, &sErrMsg );
	printf( "%s\n\n\n", sErrMsg );
	exit(1);
}
```

# Index

## A



## D



## E



## L