- Makefile
- sample_1.c
- sample_2.c
- sample_3.c
- sample_4.c
- sample_5.c
- sample_6.c
- sample_7.c
- sample_8.c
- sample_9.c
- sample_10.c

Altibase® Application Development

Monitoring API Developer's Guide

Altibase Application Development Monitoring API Developer's Guide

Release 7.1

㈜알티베이스

08378 서울시 구로구 디지털로 306 대륭포스트타워II 10층

전화: 02-2082-1114 팩스: 02-2082-1099

고객서비스포털: http://support.altibase.com

homepage: http://www.altibase.com

서문

이 매뉴얼에 대하여

이 매뉴얼은 Monitoring API 의 사용법에 대해 설명한다.

대상 사용자

이 매뉴얼은 다음과 같은 Altibase 사용자를 대상으로 작성되었다.

- 데이터베이스 관리자
- 성능 관리자
- 데이터베이스 사용자
- 응용 프로그램 개발자
- 기술지원부

다음과 같은 배경 지식을 가지고 이 매뉴얼을 읽는 것이 좋다.

- 컴퓨터, 운영 체제 및 운영 체제 유틸리티 운용에 필요한 기본 지식
- 관계형 데이터베이스 사용 경험 또는 데이터베이스 개념에 대한 이해
- 컴퓨터 프로그래밍 경험
- 데이터베이스 서버 관리, 운영 체제 관리 또는 네트워크 관리 경험

소프트웨어 환경

이 매뉴얼은 Altibase 버전 7.1을 사용한다는 가정 하에 작성되었다.

이 매뉴얼의 구성

이 매뉴얼은 다음과 같이 구성되어 있다.

- 제 1장 소개
  이 장은 Monitoring API가 무엇인지 소개하고 특징을 설명한다.
- 제 2장 데이터 타입
  이 장은 Monitoring API와 함께 사용되는 데이터 타입에 대해서 설명한다.
- 제 3장 함수
  이 장은 Monitoring API의 함수 명세를 기술한다.
- 제 4장 예제 프로그램
  이 장은 Monitoring API를 사용해서 작성된 C 프로그램 예제를 제공한다.

문서화 규칙

이 절에서는 이 매뉴얼에서 사용하는 규칙에 대해 설명한다. 이 규칙을 이해하면 이 매뉴얼과 설명서 세트의 다른 매뉴얼에서 정보를 쉽게 찾을 수 있다. 여기서 설명하는 규칙은 다음과 같다.

규칙	의미
기울임꼴	구문 요소에서 사용자가 지정해야 하는 변수, 특수한 값을 제공해야만 하는 위치 지정자, 강조, 또는 책 제목
고정폭 글꼴	단락 또는 예제 코드 내에 있는 명령어

관련 자료

자세한 정보를 위하여 다음 문서 목록을 참조하기 바란다.

- Installation Guide
- Administrator’s Manual
- Replication Manual
- CLI User's Manual
- iSQL User’s Manual
- Utilities Manual
- Error Message Reference

Altibase는 여러분의 의견을 환영합니다.

이 매뉴얼에 대한 여러분의 의견을 보내주시기 바랍니다. 사용자의 의견은 다음 버전의 매뉴얼을 작성하는데 많은 도움이 됩니다. 보내실 때에는 아래 내용과 함께 고객서비스포털(http://support.altibase.com/)로 보내주시기 바랍니다.

- 사용 중인 매뉴얼의 이름과 버전
- 매뉴얼에 대한 의견
- 사용자의 성함, 주소, 전화번호

이 외에도 Altibase 기술지원 설명서의 오류와 누락된 부분 및 기타 기술적인 문제들에 대해서 이 주소로 보내주시면 정성껏 처리하겠습니다. 기술적인 부분과 관련하여 즉각적인 도움이 필요한 경우에는 기술지원센터로 연락하시기 바랍니다.

여러분의 의견에 항상 감사드립니다.

1.소개

이 장은 Monitoring API가 무엇인지 소개하고 특징을 설명한다.

Monitoring API란?

Monitoring API는 애플리케이션에서 Altibase를 모니터링할 수 있도록 제공되는 애플리케이션 프로그래밍 인터페이스이다.

용도

Monitoring API는 외부 모니터링 툴 개발자를 위해 제공되는 인터페이스이다. 애플리케이션에서 Altibase의 성능 뷰를 직접 조회하여 모니터링 정보를 수집할 수도 있으나, Monitoring API를 사용하면 개발자들이 모니터링 툴을 좀 더 쉽게 개발할 수 있다.

일반적인 데이터베이스 접근 인터페이스(ODBC, JDBC 등)를 사용하는 사용자에게 기본으로 제공되지 않는다.

특징

Monitoring API를 사용하면 애플리케이션에서 다음의 데이터를 조회할 수 있다.

- Altibase 운영 중의 여러 통계 정보
- 현재 접속중인 세션의 수
- Altibase 서버에 접속할 수 있는 최대 클라이언트의 수
- 세션들의 락 정보
- 대기중인 이벤트 정보

지원되는 Altibase 버전

Monitoring API는 Altibase 5.5.1 이상 버전에서 지원된다.

주의사항

Monitoring API로 애플리케이션을 작성하고 실행할 때의 주의사항이다.

- Monitoring API 애플리케이션은 Unix Domain Socket을 통해서 Altibase 서버에 접속하므로 애플리케이션과 Altibase가 같은 장비에서 실행되어야 한다.
- Monitoring API 함수 내부(라이브러리)에서 할당하는 메모리는 Monitoring API 함수들이 공유하기 때문에 thread-safe하지 않다. 따라서 멀티 쓰레드 프로그램을 작성할 때에는 공유 자원에 여러 개의 쓰레드가 동시에 접근하지 못하도록 뮤텍스를 사용한 동기화가 필요하다. 4장에서 예제 프로그램 sample_7.c를 참고하라.

애플리케이션 빌드하기

이 절은 Monitoring API 애플리케이션을 빌드하는데 필요한 헤더 파일과 라이브러리 파일을 알려주고 컴파일 방법에 대해 기술한다.

헤더 파일

Monitoring API 애플리케이션을 작성할 때 포함되어야 하며 컴파일시에 참조되는 헤더 파일은 altibaseMonitor.h이다. 이 파일은 $ALTIBASE_HDB_HOME/include 디렉토리에 존재한다.

컴파일시에는 컴파일 명령어에 다음의 옵션을 사용하라.

```
-I$ALTIBASE_HDB_HOME/include
```

라이브러리 파일

Monitoring API 애플리케이션을 빌드하려면, 컴파일된 오브젝트 파일을 Altibase가 제공하는 Monitoring API 라이브러리와 ODBC 라이브러리, 그리고 몇몇 시스템 라이브러리와 함께 링크해야 한다.

- Monitoring API 라이브러리: libaltibaseMonitor.a, libaltibaseMonitor_sl.so
- ODBC 라이브러리: libodbccli.a
- 시스템 라이브러리: libpthread.a, libdl.a

Monitoring API 라이브러리와 ODBC 라이브러리는 $ALTIBASE_HDB_HOME/lib 디렉토리에 존재한다.

컴파일

아래는 Altibase 패키지를 설치할 때 생기는 $(ALTIBASE_HOME)/install/altibase_env.mk 파일을 이용하여 sample.c 소스 파일을 컴파일하는 Makefile 예제이다.

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

아래는 콘솔창에서 gcc 컴파일러를 사용해서 sample.c 소스 파일을 컴파일하는 예제이다.

```
% 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 -lrt
```

2.데이터 타입

이 장은 Monitoring API와 함께 사용되는 데이터 타입에 대해서 설명한다.

데이터 구조체

이 절은 Monitoring API의 함수를 호출할 때 인자로 사용하는 데이터 구조체를 설명한다.

ABIVSession

V$SESSION 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SESSION 성능 뷰를 참고한다.

멤버	타입	V$SESSION의 대응하는 칼럼
mID	int	ID
mTransID	long 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
mClientProtocolVersion[40+1]	char	CLIENT_PROTOCOL_VERSION

멤버	타입	V$SESSION의 대응하는 칼럼
mClientPID	long 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 long	DEFAULT_TBSID
mDefaultTempTbsID	long 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 long	TRX_UPDATE_MAX_LOGSIZE
mParallelDmlMode	int	PARALLEL_DML_MODE
mLoginTime	int	LOGIN_TIME
mFailOverSource[64+1]	char	FAILOVER_SOURCE

ABIVSysstat

데이터베이스 시스템 전체의 통계 정보를 보여주는 V$SYSSTAT 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SYSSTAT 성능 뷰를 참고한다.

멤버	타입	V$SYSSTAT의 대응하는 칼럼
mValue	long long	VALUE

ABIVSesstat

세션 별 통계 정보를 보여주는 V$SESSTAT 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SESSTAT 성능 뷰를 참고한다.

멤버	타입	V$SESSTAT의 대응하는 칼럼
mSID	int	SID
mValue	long long	VALUE

ABIStatName

V$SYSSTAT 또는 V$SESSTAT 성능 뷰의 고정 칼럼들의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SYSSTAT와 V$SESSTAT 성능 뷰를 참고한다.

멤버	타입	V$SYSSTAT 또는 V$SESSTAT의 대응하는 칼럼
mSeqNum	int	SEQNUM
mName[128+1]	char	NAME

ABIVSystemEvent

V$SYSTEM_EVENT 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SYSTEM_EVENT 성능 뷰를 참고한다.

멤버	타입	V$SYSTEM_EVENT의 대응하는 칼럼
mTotalWaits	long long	TOTAL_WAITS
mTotalTimeOuts	long long	TOTAL_TIMEOUTS
mTimeWaited	long long	TIME_WAITED
mAverageWait	long long	AVERAGE_WAIT
mTimeWaitedMicro	long long	TIME_WAITED_MICRO

ABIVSessionEvent

V$SESSION_EVENT 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SESSION _EVENT 성능 뷰를 참고한다.

멤버	타입	V$SESSION_EVENT의 대응하는 칼럼
mSID	int	SID
mTotalWaits	long long	TOTAL_WAITS
mTotalTimeOuts	long long	TOTAL_TIMEOUTS
mTimeWaited	long long	TIME_WAITED
mAverageWait	long long	AVERAGE_WAIT
mMaxWait	long long	MAX_WAIT
mTimeWaitedMicro	long long	TIME_WAITED_MICRO

ABIEventName

V$SYSTEM_EVENT 또는 V$SESSION_EVENT 성능 뷰의 고정 칼럼들의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SYSTEM_EVENT와 V$SESSION_EVENT 성능 뷰를 참고한다.

멤버	타입	V$SYSTEM_EVENT 또는 V$SESSION_EVENT의 대응하는 칼럼
mEventID	int	EVENT_ID
mEvent[128+1]	char	EVENT
mWaitClassID	int	WAIT_CLASS_ID
mWaitClass[128+1]	char	WAIT_CLASS

ABIVSessionWait

V$SESSION_WAIT 성능 뷰의 조회 결과를 저장하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다. 각 칼럼의 의미는 *General Reference*에서 V$SESSION_WAIT 성능 뷰를 참고한다.

멤버	타입	V$SESSION_WAIT의 대응하는 칼럼
mSID	int	SID
mSeqNum	int	SEQNUM
mP1	long long	P1
mP2	long long	P2
mP3	long long	P3
mWaitClassID	int	WAIT_CLASS_ID
mWaitTime	long long	WAIT_TIME
mSecondInTime	long long	SECOND_IN_TIME

ABISqlText

Statement ID로 SQL문 텍스트, 쿼리 시작 시간, 쿼리의 수행 여부를 조회하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명
mSessID	int	세션 ID

멤버	타입	설명
mStmtID	int	Statement ID
mSqlText	char *	SQL문 텍스트
mTextLength	int	mSqlText에 저장된 문자열의 길이
mQueryStartTime	int	쿼리 시작 시간
mExecuteFlag	int	쿼리의 수행 여부 0 : 수행됨 1: 수행안됨
mParseTime	long	파싱 소요 시간
mSoftPrepareTime	long	Prepare 과정중 SQL Plan Cache에서 plan 탐색 시간
mLastQueryStartTime	int	가장 최근의 쿼리 시작 시간
mExecuteTime	long	실행 소요 시간
mFetchTime	long	Fetch 소요 시간
mFetchStartTime	int	현재 Fetch 시작 시간
mTotalTime	long	총 경과 시간
mValidateTime	long	정당성 검사 소요 시간
mOptimizeTime	long	최적화 소요 시간

ABILockPair

락(Lock)을 잡고 있는 세션과 이 락을 획득하기 위해 대기하는 세션의 쌍을 조회하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명
mHolderSID	int	락을 잡고 있는 세션의 ID
mWaiterSID	int	락을 잡고 있는 세션(mHolderSID)으로 인해 대기중인 세션의 ID
mLockDesc[32+1]	char	대기중인 세션(mWaiterSID)이 획득하고자 하는 락 모드

ABIDBInfo

데이터베이스의 이름, 버전 번호를 조회하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명
mDBName[128+1]	char	데이터베이스 이름
mDBVersion[128+1]	char	데이터베이스 버전 번호

ABIReadCount

Altibase 서버에서 발생한 데이터 페이지 읽기 횟수를 조회하는 데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명
mLogicalReadCount	int	메모리 버퍼에서 데이터 페이지를 읽은 횟수
mPhysicalReadCount	int	디스크에서 데이터 페이지를 읽은 횟수

ABIRepGap

Altibase 서버에서 발생하는 이중화 송신자의 작업 로그 레코드와 가장 최근에 생성된 로그 레코드간의 차이를 조회하는데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명
mRepName[40+1]	char	이중화 객체의 이름
mRepGap	long long	마지막으로 전송된 로그 레코드의 번호(REP_LAST_SN)와 현재 전송중인 로그 레코드(REP_SN)의 차이

ABIRepSentLogCount

Altibase 서버에서 이중화 송신자가 전송한 로그의 개수를 조회하는데 사용되는 데이터 구조체이다.

이 구조체는 아래 표와 같은 멤버를 가진다.

멤버	타입	설명

멤버	타입	설명
mRepName[40+1]	char	이중화 객체의 이름
mTableName[128+1]	char	테이블 객체의 이름
mInsertLogCount	int	INSERT 로그의 개수
mDeleteLogCount	int	DELETE 로그의 개수
mUpdateLogCount	int	UPDATE 로그의 개수

열거형

Monitoring API 애플리케이션 작성을 위해 제공되는 열거형은 다음과 같다.

enum ABIPropType

Altibase 서버에 접속하기 위한 사용자와 사용자 암호를 지정하기 위해 ABISetProperty 함수와 함께 사용되는 열거형이다.

이 열거형의 원소는 다음과 같다

원소	설명
ABI_USER	사용자 이름을 지정할 때 사용된다.
ABI_PASSWD	사용자 암호를 지정할 때 사용된다.
ABI_LOGFILE	Monitoring API에서 발생하는 에러 메시지가 기록되는 파일을 지정할 때 사용된다.

주의사항

Monitoring API 함수 대부분은 위의 절에서 설명한 구조체를 인자로 가진다. 이 절은 이들 구조체를 인자로 사용할 때의 주의점을 설명한다.

애플리케이션에서는 구조체형 포인터 변수를 선언하고, 이 포인터의 주소값(이중 포인터)을 Monitoring API 함수에 전달해야 한다. Monitoring API 함수는 전달받은 포인터에 힙 메모리를 할당하고 데이터베이스에서 조회한 결과를 셋팅함으로써 애플리케이션에 결과 셋을 반환한다.

즉, Monitoring API용의 데이터 구조체를 위한 메모리 할당과 해제는 Monitoring API 함수 내부에서 이루어지므로, 애플리케이션에서는 직접 해당 구조체형 포인터에

메모리를 할당하거나 함수 수행 결과로 반환받은 메모리를 해제하지 않아야 한다.

아래 예제 코드에서 보듯이 애플리케이션에서는 ABIVSession 구조체형 포인터 변수를 선언할 뿐, sVSession에 메모리를 할당해서는 안 된다. 또한 함수 수행 후 sVSession에서 결과 값을 참조할 때 결과 셋의 로우 개수 내에서 배열 요소들을 액세스해야 한다.

```
ABIVSession *sVSession;
int sRowCount;

sRowCount = ABIGetVSession(  &sVSession, 0 );

/* sVSession에서 조회된 결과 참조하기 */
for (int i=0; i<sRowCount; i++)
{
    /* sVSession[i].mID; */
    /* sVSession[i].mTransID; */
}
```

3.함수

이 장은 Monitoring API 함수들의 명세를 기술한다. 각 함수 별로 다음의 정보가 제공된다.

- 함수명
- 구문: 함수의 C 언어 프로토타입
- 인자: 함수의 각 인자별 자료 유형, 입/출력, 부연 설명
- 반환 값: 반환 가능한 이 함수의 리턴 값
- 설명: 함수 사용 방법 및 주의 사항
- 예제: 이 함수가 사용된 소스 코드의 일부

ABIInitialize

구문

```
int ABIInitialize ( void );
```

반환값

함수 수행이 성공하면 0, 그렇지 않으면 음의 정수값으로 에러 코드가 반환된다.

설명

Monitoring API를 사용하기 위해 첫 번째로 반드시 호출해야 하는 함수이다. Altibase 서버와의 연결 설정 같은 초기화 작업을 수행한다.

예제

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

ABIFinalize

구문

```
int ABIFinalize ( void );
```

반환값

함수 수행이 성공하면 0, 그렇지 않으면 음의 정수값으로 에러 코드가 반환된다.

설명

Monitoring API 사용의 마지막에 반드시 호출해야 하는 함수이다. Monitoring API를 사용하면서 할당된 메모리를 해제하고 Altibase 서버와의 연결을 종료하는 등의 작업을 수행한다.

예제

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

ABISetProperty

구문

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

인자

자료유형	인자	입/출력	설명
ABIPropType	aPropType	입력	설정할 프로퍼티의 이름을 지정한다. 아래의 값 중에서 하나를 사용할 수 있다. ABI_USER, ABI_PASSWD, ABI_LOGFILE
const char *	aPropValue	입력	설정할 프로퍼티의 값

반환값

함수 수행이 성공하면 0, 그렇지 않으면 음의 정수값으로 에러 코드가 반환된다.

설명

이 함수는 Altibase 서버에 접속할 사용자 이름, 사용자 암호를 설정하거나, 로그 파일을 설정한다. 설정하지 않을 경우 기본값은 각각 "SYS", "MANAGER", " altibaseMonitor.log"이다.

로그 파일은 Monitoring API 에서 발생하는 에러 메시지를 기록하는 파일이다. 경로 없이 파일 이름만 설정하는 경우 애플리케이션이 수행되는 경로에 로그 파일이 생성된다.

예제

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

ABICheckConnection

구문

```
int ABICheckConnection ( );
```

반환값

접속 상태가 정상이면 0, 그렇지 않으면 -1이 반환된다.

설명

이 함수는 Altibase 서버와의 연결 상태를 검사한다.

예제

```
if( ABICheckConnection( ) != -1 )
{
    /* 퍼포먼스 뷰 조회 */
}
else
{
    /* ... error handling ... */
}
```

ABIGetVSession

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSession**	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
unsigned int	aExecutingOnly	입력	0: 전체 세션 조회 1: Active 세션만 조회

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSION 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에
결과 셋이 저장된 ABIVSession 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_1.c를 참고하도록 한다.

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

/* 전체 세션 조회 */
sRowCount = ABIGetVSession(  &sVSession, 0 );

/* Active 세션만 조회 */
sRowCountActiveOnly = ABIGetVSession( &sVSessionActiveOnly, 1 );
```

ABIGetVSessionBySID

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSession**	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
int	aSessionID	입력	조회할 세션의 ID

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSION 성능 뷰에서 특정 세션에 대한 정보만 조회하는 함수이다. 함수가 성공적으로 수행되면 결과 셋이 저장된 *aHandle*에 ABIVSession 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_1.c를 참고하도록 한다.

```
ABIVSession *sVSession;
int sRowCount;

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

ABIGetVSysstat

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSysstat **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SYSSTAT 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에
결과 셋이 저장된 ABIVSysstat 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_3.c를 참고하도록 한다.

```
ABIVSysstat *sVSysstat;
int sRowCount;

sRowCount = ABIGetVSysstat( &sVSysstat );
```

ABIGetVSesstat

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSesstat **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
unsigned int	aExecutingOnly	입력	0: 전체 세션 조회 1: Active 세션만 조회

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSTAT 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에
결과 셋이 저장된 ABIVSesstat 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_3.c를 참고하도록 한다.

```
ABIVSesstat *sVSesstat;
int sRowCount;

sRowCount = ABIGetVSesstat( &sVSesstat );
```

ABIGetVSesstatBySID

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSesstat **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
int	aSessionID	입력	조회할 세션의 ID

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSTAT 성능 뷰에서 특정 세션에 대한 통계자료를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSesstat 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_3.c를 참고하도록 한다.

```
ABIVSesstat *sVSesstat;
int sRowCount;

/* ID가 1인 세션 조회 */
sRowCount = ABIGetVSesstatBySID ( &sVSesstat, 1 );
```

ABIGetStatName

구문

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

인자

자료유형	인자	입/출력	설명

자료유형	인자	입/출력	설명
ABIStatName **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SYSSTAT 또는 V$SESSTAT 성능 뷰의 고정 칼럼들인 SEQNUM과 NAME의 값을 조회하는 함수이다.

예제

이 함수와 관련된 프로그램은 4장의 sample_3.c를 참고하도록 한다.

```
ABIStatName *sStatName;
int sRowCount;

sRowCount = ABIGetStatName( &sStatName );
```

ABIGetVSystemEvent

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSystemEvent **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SYSTEM_EVENT 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSystemEvent 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_4.c를 참고하도록 한다.

```
ABIVSystemEvent *sVSystemEvent;
int sRowCount;

sRowCount = ABIGetVSystemEvent( &sVSystemEvent);
```

ABIGetVSessionEvent

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSessionEvent **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSION_EVENT 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSessionEvent 타입의 배열을 가리키는 포인터가

반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_4.c를 참고하도록 한다.

```
ABIVSessionEvent *sVSessionEvent;
int sRowCount;

sRowCount = ABIGetVSessionEvent( &sVSessionEvent);
```

ABIGetVSessionEventBySID

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSessionEvent **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
int	aSessionID	입력	조회할 세션의 ID

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSION_EVENT 성능 뷰에서 특정 세션에 대한 대기 이벤트들의 통계정보를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSessionEvent 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_4.c를 참고하도록 한다.

```
ABIVSessionEvent *sVSessionEvent;
int sRowCount;

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

ABIGetEventName

구문

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

인자

자료유형	인자	입/출력	설명
ABIEventName **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SYSTEM_EVENT 또는 V$SESSION_EVENT 성능 뷰의 고정 칼럼들인 EVENT_ID, EVENT, WAIT_CLASS_ID, WAIT_CLASS의 값을 조회하는 함수이다.

예제

이 함수와 관련된 프로그램은 4장의 sample_4.c를 참고하도록 한다.

```
ABIEventName *sEventName;
int sRowCount;

sRowCount = ABIGetEventName( &sEventName);
```

ABIGetVSessionWait

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSessionWait **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$SESSION_WAIT 성능 뷰를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSessionWait 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_8.c를 참고하도록 한다.

```
ABIVSessionWait *sVSessionWait;
int sRowCount;

sRowCount = ABIGetVSessionWait( &sVSessionWait);
```

ABIGetVSessionWaitBySID

구문

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

인자

자료유형	인자	입/출력	설명
ABIVSessionWait **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터
int	aSessionID	입력	조회할 세션의 ID

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로의 에러 코드가 반환된다.

설명

V$SESSION_WAIT 성능 뷰에서 특정 세션에 대한 대기 이벤트 정보를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABIVSessionWait 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_8.c를 참고하도록 한다.

```
ABIVSessionWait *sVSessionWait;
int sRowCount;

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

ABIGetSqlText

구문

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

인자

자료유형	인자	입/출력	설명

자료유형	인자	입/출력	설명
ABISqlText **	aHandle	출력	SQL문이 저장된 구조체의 메모리 주소를 받아올 포인터
int	astmtID	입력	조회할 statement의 ID aStmtID가 0일경우 현재 Active상태인 statement의 정보를 모두 반환한다.

반환값

함수 수행이 성공하면 0을 반환한다. 실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

Statement 식별자를 이용하여 해당 statement가 수행하고 있는 SQL문, 쿼리 시작 시간, 쿼리 수행 여부를 조회하는 함수이다.

예제

이 함수와 관련된 프로그램은 4장의 sample_5.c를 참고하도록 한다.

```
ABISqlText *sSqlText;
int sRet;

/* ID가 2인 statement의 SQL문 조회 */
sRet = ABIGetSqlText( &sSqlText, 2 );
```

ABIGetLockPairBetweenSessions

구문

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

인자

자료유형	인자	입/출력	설명

자료유형	인자	입/출력	설명
ABILockPair **	aHandle	출력	결과 셋이 저장된 구조체 배열의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 *aHandle*에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

락(Lock)을 잡고 있는 세션과 이 락을 획득하고자 대기하고 있는 세션의 쌍을 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과 셋이 저장된 ABILockPair 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_5.c를 참고하도록 한다.

```
ABILockPair *sLockPair;
int sRowCount;

sRowCount = ABIGetLockPairBetweenSessions( &sLockPair );
```

ABIGetDBInfo

구문

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

인자

자료유형	인자	입/출력	설명
ABIDBInfo **	aHandle	출력	결과값이 저장된 구조체의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 0을 반환한다. 실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

데이터베이스의 이름, 버전 번호를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과값이 저장된 ABIDBInfo 타입 구조체의 메모리 주소가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_6.c를 참고하도록 한다.

```
ABIDBInfo *sDBInfo;
int sRet;

sRet = ABIGetDBInfo( &sDBInfo );
```

ABIGetReadCount

구문

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

인자

자료유형	인자	입/출력	설명
ABIReadCount **	aHandle	출력	결과값이 저장된 구조체의 메모리 주소를 받아올 포인터

반환값

함수 수행이 성공하면 0을 반환한다. 실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

Altibase 서버에서 발생한 데이터 페이지 읽기 횟수를 조회하는 함수이다. 함수가 성공적으로 수행되면 *aHandle*에 결과값이 저장된 ABIReadCount 타입 구조체의 메모리 주소가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_6.c를 참고하도록 한다.

```
ABIReadCount *sReadCount;
int sRet;

sRet = ABIGetReadCount( &sReadCount);
```

ABIGetSessionCount

구문

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

인자

자료유형	인자	입/출력	설명
unsigned int	aExecutingOnly	입력	0: 전체 세션 수 조회 1: Active 세션 수만 조회

반환값

함수 수행이 성공하면 Altibase 서버에 존재하는 세션의 총 개수가 반환된다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

현 시점에서 Altibase 서버에 존재하는 세션의 총 개수 또는 active 세션의 개수를
조회하는 함수이다.

예제

이 함수와 관련된 프로그램은 4장의 sample_2.c를 참고하도록 한다.

```
int sSessionCount;
int sActiveSessionCount;

/* 전체 세션 개수 조회 */
sSessionCount = ABIGetSessionCount( 0 );
/* Active 세션 개수 조회 */
sActiveSessionCount  = ABIGetSessionCount( 1 );
```

ABIGetMaxClientCount

구문

```
int ABIGetMaxClientCount ( );
```

반환값

함수 수행이 성공하면 Altibase 서버에 접속할 수 있는 클라이언트의 최대 개수가 반환된다. 실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

Altibase 서버에 접속할 수 있는 클라이언트의 최대 개수를 조회하는 함수이다.
조회되는 값은 Altibase 서버의 altibase.properties 파일에 MAX_CLIENT 프로퍼티로 설정한 값과 동일하다.

예제

이 함수와 관련된 프로그램은 4장의 sample_2.c를 참고하도록 한다.

```
int sMaxClientCount;

sMaxClientCount = ABIGetMaxClientCount( );
```

ABIGetLockWaitSessionCount

구문

```
int ABIGetLockWaitSessionCount ( );
```

반환값

함수 수행이 성공하면 락(lock)을 획득하기 위해 대기중인 세션의 개수가 반환된다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

Altibase 서버에서 락을 획득하기 위해 대기중인 세션의 개수를 조회하는 함수이다.

예제

이 함수와 관련된 프로그램은 4장의 sample_5.c를 참고하도록 한다.

```
int sLockWaitSessionCount;

sLockWaitSessionCount = ABIGetLockWaitSessionCount( );
```

ABIGetRepGap

구문

```
int ABIGetRepGap(
   ABIRepGap **aHandle );
```

반환값

함수 수행이 성공하면 aHandle에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$REPGAP 성능 뷰에서 이중화 송신자가 마지막으로 전송한 로그 레코드와 가장 최근에 생성된 로그 레코드의 차이를 조회하는 함수이다.

함수가 성공적으로 수행되면 aHandle에 결과 셋이 저장된 ABIRepGap 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_10.c를 참고하도록 한다.

```
ABIRepGap *sRepGap;
int sRowCount;

sRowCount = ABIGetRepGap( &sRepGap );
```

ABIGetRepSentLogCount

구문

```
int ABIGetRepSentLogCount(
   ABIRepSentLogCount **aHandle );
```

반환값

함수 수행이 성공하면 aHandle에 가져온 결과 셋 내의 로우 개수를 반환한다.
실패하면 음의 정수 값으로 에러 코드가 반환된다.

설명

V$REPSENDER_SENT_LOG_COUNT 성능 뷰에서 이중화 송신자가 전송한 로그를 DML 타입 별로 분류하여 로우의 개수를 조회하는 함수이다.

함수가 성공적으로 수행되면 aHandle에 결과 셋이 저장된 ABIRepSentLogCount 타입의 배열을 가리키는 포인터가 반환된다.

예제

이 함수와 관련된 프로그램은 4장의 sample_10.c를 참고하도록 한다.

```
ABIRepSentLogCount *sRepSentLogCount;
int sRowCount;

sRowCount = ABIRepSentLogCount( &sRepSentLogCount );
```

ABIGetErrorMessage

구문

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

인자

자료유형	인자	입/출력	설명
int	aErrCode	입력	에러 코드
const char *	aErrMsg	출력	에러 메시지를 받아올 버퍼 포인터

설명

에러 코드를 이용하여 에러 메시지를 조회하는 함수이다. Monitoring API의 함수가 에러 코드를 반환할 때 에러 코드에 해당하는 에러 메시지를 조회할 수 있다.

예제

이 함수와 관련된 프로그램은 4장의 sample_9.c를 참고하도록 한다.

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

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

4.예제 프로그램

이 장은 Monitoring API를 사용해서 작성된 C 프로그램 예제를 제공한다.

Makefile

이 장에서 제공하는 예제 프로그램을 컴파일하기 위한 Makefile이다. Altibase 패키지에 포함되어 있는 altibase_env.mk 파일이 이용된다.

```
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) $(LIBS)
sample_10 : sample_10.$(OBJEXT)
        $(LD) $(LDOUT) $@ $^ $(LFLAGS) $(LIBOPT)altibaseMonitor$(LIBAFT) $(LIBOPT)odbccli$(LIBAFT) $(LIBS)
clean :
        $(RM) $(OBJS) $(BINS) *.log
```

sample_1.c

ABIGetVSession과 ABIGetVSessionBySID 함수를 사용해서 V$SESSION 성능 뷰를 조회하는 예제 프로그램이다.

```
#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);
}
```

sample_2.c

이 예제 프로그램은 ABIGetSessionCount 함수를 사용해서 Altibase 서버에 존재하는 세션의 총 개수와 active 세션의 개수를 조회한다. 그리고 ABIGetMaxClientCount 함수를 사용해서 Altibase 서버에 접속할 수 있는 클라이언트의 최대 개수를 조회한다.

```
#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);
}
```

sample_3.c

ABIGetStatName, ABIGetVSysstat, ABIGetVSesstat, ABIGetVSesstatBySID함수를 사용해서 Altibase 시스템과 세션들의 통계정보를 조회하는 예제 프로그램이다.

```
#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);
}
```

sample_4.c

ABIGetEventName, ABIGetVSystemEvent, ABIGetVSessionEvent, ABIGetVSessionEventBySID 함수를 사용해서 Altibase 시스템과 세션들의 대기 이벤트들에 대한 통계정보를 조회하는 예제 프로그램이다.

```
#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 aEventNam
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 aEventNam
{
    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);
}
```

sample_5.c

ABIGetSqlText, ABIGetLockPairBetweenSessions, ABIGetLockWaitSessionCount 함수를 사용하는 예제 프로그램이다.

```
#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 );
    printf( "PARSE TIME : %lld\n\n\n", aSqlText->mParseTime);
    printf( "SOFT PREPARE TIME : %lld\n\n\n", aSqlText->mSoftPrepareTime);
    printf( "LAST QUERY START TIME : %d\n\n\n", aSqlText->mLastQueryStartTime);
    printf( "EXECUTE TIME : %lld\n\n\n", aSqlText->mExecuteTime);
```

```
    printf( "FETCH TIME : %lld\n\n\n", aSqlText->mFetchTime);
    printf( "FETCH START TIME : %d\n\n\n", aSqlText->mFetchStartTime);
    printf( "TOTAL TIME : %lld\n\n\n", aSqlText->mTotalTime);
    printf( "VALIDATE TIME : %lld\n\n\n", aSqlText->mValidateTime);
    printf( "OPTIMIZE TIME : %lld\n\n\n", aSqlText->mOptimizeTime);
}

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);
}
```

sample_6.c

ABIGetDBInfo, ABIGetReadCount 함수를 사용하는 예제 프로그램이다.

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

void printDBInfo( ABIDBInfo *aDBInfo );
void printReadCount( ABIReadCount *aReadCount );
void errorHandling( int aErrCode );

int main()
{
    ABIDBInfo    *sDBInfo = NULL;
    ABIReadCount *sReadCount = NULL;
    int           sRc = 0;

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

    // Testing ABICheckConnection()
    if( ABICheckConnection() != -1 )
    {
        // Testing ABIGetDBInfo
        sRc = ABIGetDBInfo( &sDBInfo );
        if( sRc >= 0 )
        {
            printf( "*****************************\n" );
            printf( "*          DB Info          *\n" );
            printf( "*****************************\n" );
            printDBInfo( sDBInfo );

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

        // Testing ABIGetReadCount
        sRc = ABIGetReadCount( &sReadCount );
        if( sRc >= 0 )
        {
            printf( "********************************\n" );
            printf( "*          Read Count          *\n" );
            printf( "********************************\n" );
            printReadCount( sReadCount );
```

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

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

    return 0;
}

void printDBInfo( ABIDBInfo *aDBInfo )
{
    printf( "DB NAME : %s\n", aDBInfo->mDBName );
    printf( "VERSION : %s\n\n\n", aDBInfo->mDBVersion );
}

void printReadCount( ABIReadCount *aReadCount )
{
    printf( "Logical Read Count : %d\n", aReadCount->mLogicalReadCount );
    printf( "Physical Read Count : %d\n\n\n", aReadCount->mPhysicalReadCount );
}

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

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

sample_7.c

pthread_mutex_t 타입의 전역 변수를 사용하여 두 쓰레드에서 ABIGetVSession와 ABIGetSessionCount 함수 호출을 동기화하는 예제 프로그램이다.

```
#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);
}
```

sample_8.c

ABIGetVSessionWait, ABIGetVSessionWaitBySID 함수를 사용해서 Altibase 서버에 접속중인 세션들의 대기 이벤트 정보를 조회하는 예제 프로그램이다.

```
#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);
}
```

sample_9.c

ABIGetErrorMessage 함수를 사용해서 에러 메시지를 검사하는 예제 프로그램이다.

```
#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);
}
```

sample_10.c

ABIGetRepGap, ABIGetRepSentLogCount 함수를 사용하는 예제 프로그램이다.

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

void printRepGap( ABIRepGap *aRepGap, int aRowCount );
void printRepSentLogCount( ABIRepSentLogCount *aRepSentLogCount, int aRowCount );
void errorHandling( int aErrCode );

int main()
{
    ABIRepGap          *sRepGap = NULL;
    ABIRepSentLogCount *sRepSentLogCount = NULL;
    int                 sRc = 0;

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

    // Test ABICheckConnection
    if( ABICheckConnection() != -1 )
    {
        // Test ABIRepGap
        sRc = ABIGetRepGap( &sRepGap );
        if( sRc >= 0 )
        {
            printf( "*******************************\n" );
            printf( "*           RepGap            *\n" );
            printf( "*******************************\n" );
            printRepGap( sRepGap, sRc );
            sRepGap = NULL;
            sRc = 0;
        }
        else
        {
            // Error handling
            errorHandling( sRc );
        }

        // Test ABIGetRepSentLogCount
        sRc = ABIGetRepSentLogCount( &sRepSentLogCount );
        if( sRc >= 0 )
        {
            printf( "*********************************************\n" );
            printf( "*               RepSentLogCount             *\n" );
            printf( "*********************************************\n" );
            printRepSentLogCount( sRepSentLogCount, sRc );
            sRepSentLogCount = NULL;
```

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

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

    return 0;
}

void printRepGap( ABIRepGap *aRepGap, int aRowCount )
{
    int sI;

    for( sI = 0; sI < aRowCount; sI++ )
    {
        printf( "REP_NAME : %s\n", aRepGap[sI].mRepName );
        printf( "REP_GAP : %lld\n", aRepGap[sI].mRepGap );
    }
    printf( "\n" );
}

void printRepSentLogCount( ABIRepSentLogCount *aRepSentLogCount, int aRowCount )
{
    int sI;

    for( sI = 0; sI < aRowCount; sI++ )
    {
        printf( "REP_NAME : %s\n", aRepSentLogCount[sI].mRepName );
        printf( "TABLE_NAME : %s\n", aRepSentLogCount[sI].mTableName );
        printf( "INSERT_LOG_COUNT : %d\n", aRepSentLogCount[sI].mInsertLogCount );
        printf( "DELETE_LOG_COUNT : %d\n", aRepSentLogCount[sI].mDeleteLogCount );
        printf( "UPDATE_LOG_COUNT : %d\n", aRepSentLogCount[sI].mUpdateLogCount );
    }
    printf( "\n" );
}
```

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

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