Contents

Overview

SWBLogicalLocks is an in-process server (ActiveX DLL) that allows Visual Basic programmers to implement logical locks in their application. Logical locks can be used alone, or in conjunction with database access, to allow locking that is not otherwise provided by the database engine. For example, a Microsoft Access database does not allow for single record locks. It uses a method called *page locking*, which locks several records (one 2K page) at a time. Since this is often cumbersome in multi-user applications where pessimistic locking is desired, logical locking offers an alternative that enables an application to "protect" a one record at a time. Since logical locks do not truly lock any records in your application's database, they can also be used to lock logical "processes" as well as actual data records.

Why Use Logical Locks?
Using Logical Locks

Properties

DatabaseDir
UserName

Methods

InitializeLocks
AcquireLock
ReleaseLock
EnumerateLocks
GetLockInfo
GetLockCount
ClearLocks
ShutdownLocks

Why Use Logical Locks?

Developers have often implemented schemes to emulate single-record locking in VB (with Access) by using a combination of optimistic locking and adding a field to each record that indicates if the record is locked or being edited. The problem with persistent locking indicators is that they can become inactive an not get released in the case of abnormal program termination.

For example, if a station acquires a lock on a record and updates the record so its "lock field" shows that it is locked, but then that application terminates abnormally, or the system it is running on locks and needs to be rebooted while a record is being edited, the lock is never cleared.

SWBLogicalLocks works around this problem by determining whether each requested lock: 1) already exists in the lock table, and 2) is held by a station that is truly still active. If the lock exists and the owner's station is still active, then the lock request is denied. If the lock exists, but the owner's station is no longer active, then the lock request is granted, and the requesting station "takes over" the inactive lock, making it active again. If a station should become inactive, or a process terminated abnormally before active locks can be released, SWBLogicalLocks will indicate to other requesting processes that those locks, when requested, are no longer active, and therefore can be reacquired. An inactive lock can normally be recovered in less than a minute from the time the owner's process aborts, or the owner's station is shut down. Depending on the type of process termination, this timeout period can be even shorter.

Using Logical Locks

You can use logical locking to emulate "single record locks" while using a Microsoft Access database as follows:

1. Set up your application to open the database in "optimistic locking" mode. In this mode, Access does not try to acuire a lock on the data page containing the edited record when the .Edit method is called.
2. Whenever your application is about to read a record from the database, try to acquire a logical lock on the unique "key fields" to that record. The logical lock uses a "name" field that can be up to 64 characters long. By concatenating the primary key field values together for the record you are about to edit, you can create a unique logical lock name that can be consistently generated by other users requesting the same record. For example, if you were about to request a record containing the information for an employee named "John Smith", the lock name might be "SMITHJOHN", a concatenation of the last name and first name (assuming those are fields that uniquely identify a single record in the Employee table).
3. If your application is unable to acquire the lock, then another user of your application has the requested record open for editing. If you *are* able to acquire the lock, read the record and allow editing.
4. When the user is done editing the record and the application has updated the database with the changes, release the logical lock, and the record will again be available for editing by other users using this application on the network.

Note that logical locking does not truly lock any portion of the actual data records in the database. Therefore, it will only prevent access to the data for applications that are designed to recognize and respect the logical locks in effect. In the case of a multi-user application, this works well because the same application is in use at all stations. It does not stop someone, however, from opening Access and editing records while your application is running.

DatabaseDir Property

Type: String

Access: Read/Write

Description:

Sets or returns the directory of the Microsoft Access database that will contain the logical lock table. This can be your application's database, or a separate database to be used only for the lock table.

NOTE: Set this property prior to calling the InitializeLocks method.

UserName Property

Type: String

Access: Read-Only

Description:

Returns the user name for this station. This name is set up in the Network Neighborhood as the Computer Name. It is highly recommended that all stations have a unique name when using logical locks.

InitializeLocks Method

```
InitializeLocks (DBName as String, Optional Password as String)
```

Description:

Initializes the logical lock server on this station for the calling application. The DatabaseDir property must be set prior to calling this method.

DBName Name of the Access database that will contain the logical locking table. This can be your application's database, or a separate one dedicated solely to the locking table. The locking table will automatically be created if it does not exist in the specified database.

Password Optional. If the database specified in DBName has a password associated with it, specify the passwod in this argument.

AcquireLock Method

```
AcquireLock (LockName as String, LockType as Integer) As Long
```

Description:

Attempts to acquire a lock on the specified lock name. A lock can be either exclusive or shareable. If the lock requested is exclusive, it can not already be held by another station as either an exclusive lock or a shareable lock. However, if the lock requested is shareable, it can be held by other stations as shareable. That is, sharable locks for the same name can co-exist with other shareable locks of the same name, however, an exclusive lock on a name cannot co-exist with any other type of lock on the same name.

Note: A single station cannot hold more than one copy of the same lock name, either shareable or exclusive.

LockName A string that identifies the lock to be acquired.

LockType The type of lock requested:

Exclusive	0	A lock that can be held by only one process.
Shareable	1	A lock that can be held by more than one process.

Returns

The method returns a long integer that is the "handle" for the lock if it is successfully acquired by the requesting process. This handle is zero (0) if unsuccessful, and greater than zero if successful. A lock is released by calling the ReleaseLock method and passing the handle of the lock to be released.

ReleaseLock Method

```
ReleaseLock (LockHandle As Long) As Boolean
```

Description:

Releases the lock specified by LockHandle.

LockHandle A unique lock handle obtained as the result of successfully calling AcquireLock.

Returns

The method returns a boolean True if the lock is successfully released. If the lock could not be released for any reason, False is returned.

EnumerateLocks Method

```
EnumerateLocks (LockHandle As Long, Optional UserName as String) As
Boolean
```

Description:

Enumerates the locks currently held by all stations, or by a particular user (if UserName parameter is specified).

To enumerate all locks currently held by any station, start by passing LockHandle as a long integer variable set to zero. When the function returns, LockHandle will be set to the handle of the first lock in the lock table, and the method will return True as its value. If there were no locks in the lock table, the method returns False. To retrieve the next lock in the table, pass the same variable as LockHandle, making sure its value is the last value returned by the function. EnumerateLocks uses the last LockHandle returned as its index to retrieve the next handle in the table. Continue to call EnumerateLocks until the return value for the method is False.

LockHandle A variable that will contain the last lock handle returned.

UserName Optional. If specified, the method returns lock handles only for locks held by the specified user. If not specified, all lock handles are returned regardless of owner.

Returns

The method returns a boolean True if a lock handle was found and returned in the variable passed as LockHandle. If no more locks were found in the table, the method returns False.

GetLockInfo Method

```
GetLockInfo (LockHandle As Long, LockName as String, UserName as String,
LockType as Integer) As Boolean
```

Description:

Returns the name, owner, and type of the lock identified by LockHandle.

LockHandle (Input) The handle of the lock to return information for.

LockName (Output) Returns the name of the lock.

UserName (Output) Returns the owner of the lock.

LockType (Output) Returns the lock type (Exclusive or Shareable).

Returns

The method returns a boolean True if the specified lock handle was found in the lock table and information was returned. If the handle does not identify a valid lock, the method returns False.

GetLockCount Method

```
GetLockCount (LockName as String) As Long
```

Description:

Returns the number of processes holding the lock specified by LockName. For exclusively held locks, the return value will always be one (1). For shared locks, any number of processes can hold a shared lock.

LockName The name of the lock.

Returns

The method returns a long integer count of the number of processes holding the specified lock name.

ClearLocks Method

`ClearLocks`

Description:

Releases all locks for the calling process. Locks held by other processes are not affected.

ShutdownLocks Method

`ShotdownLocks`

Description:

Releases all locks held by the calling process, and releases the references to the lock table and database held by the calling application. This method is normally called when terminating an application. Once ShutdownLocks is called, the application must call InitializeLocks before further logical lock processing can occur.