The C++ AbstractMutex Class

`AbstractMutex` defines a mutex base interface used by the Freeze background save evictor. The interface allows the evictor to synchronize with servants that are stored in a Freeze database. The class has the following definition:

C++

```
class AbstractMutex {
public:
    typedef LockT<AbstractMutex> Lock;
    typedef TryLockT<AbstractMutex> TryLock;

    virtual ~AbstractMutex();

    virtual void lock() const = 0;
    virtual void unlock() const = 0;
    virtual bool tryLock() const = 0;
};
```

This class definition is provided in `IceUtil/AbstractMutex.h`. The same header file also defines a few template implementation classes that specialize `AbstractMutex`, as described below.

AbstractMutexI

This template class implements `AbstractMutex` by forwarding all member functions to its template argument:

C++

```
template <typename T>
class AbstractMutexI : public AbstractMutex, public T {
public:
    typedef LockT<AbstractMutexI> Lock;
    typedef TryLockT<AbstractMutexI> TryLock;

    virtual void lock() const {
        T::lock();
    }

    virtual void unlock() const {
        T::unlock();
    }

    virtual bool tryLock() const {
        return T::tryLock();
    }

    virtual ~AbstractMutexI() {}
};
```

AbstractMutexReadI

This template class implements a read lock by forwarding the `lock` and `tryLock` functions to the `readLock` and `tryReadLock` functions of its template argument:

C++

```
template <typename T>
class AbstractMutexReadI : public AbstractMutex, public T {
public:
    typedef LockT<AbstractMutexReadI> Lock;
    typedef TryLockT<AbstractMutexReadI> TryLock;

    virtual void lock() const {
        T::readLock();
    }

    virtual void unlock() const {
        T::unlock();
    }

    virtual bool tryLock() const {
        return T::tryReadLock();
    }

    virtual ~AbstractMutexReadI() {}
};
```

`AbstractMutexWriteI`

This template class implements a write lock by forwarding the `lock` and `tryLock` functions to the `writeLock` and `tryWriteLock` functions of its template argument:

C++

```
template <typename T>
class AbstractMutexWriteI : public AbstractMutex, public T {
public:
    typedef LockT<AbstractMutexWriteI> Lock;
    typedef TryLockT<AbstractMutexWriteI> TryLock;

    virtual void lock() const {
        T::writeLock();
    }

    virtual void unlock() const {
        T::unlock();
    }

    virtual bool tryLock() const {
        return T::tryWriteLock();
    }

    virtual ~AbstractMutexWriteI() {}
};
```

Apart from use with Freeze servants, these templates are also useful if, for example, you want to implement your own evictor.

See Also

- Background Save Evictor