



CExampleTimer implementation shown in CExampleTimer.cpp illustrates simple timer implementation using active objects and RTimer. To construct the time just use the static NewL function and supply the active object priority along side with the callback interface to be notified when the timer expires.

The priority value should be one of the priorities defined inside CActive (i.e EPriorityIdle, EPriorityLow, EPriorityStandard, EPriorityUserInput or EPriorityHigh)

And for activating the timer there are three public function provided, which are:

1. At(aTime), timer is set to expire at the time specified.
2. After(aInterval), timer is set to expire after the specified interval.
3. Inactivity(aSeconds), timer is set to expire after specified interval of user inactivity.

## CExampleTimer.cpp

```

CExampleTimer::CExampleTimer(const TInt aPriority, MExampleTimerNotify& aNotify)
:CActive(aPriority), iNotify(aNotify)
{
}

CExampleTimer::~CExampleTimer()
{
    ()Cancel
    CTimer;
}

CExampleTimer* CExampleTimer::NewL(const TInt aPriority, MExampleTimerNotify& aNotify)
{
    CExampleTimer* me = new (ELeave) CExampleTimer(aPriority, aNotify);
    CleanupStack::PushL(me);
    ->ConstructL();
    CleanupStack::Pop();
    return me;
}

void CExampleTimer::ConstructL(void)
{
    CActiveScheduler::Add(this);
    CTimerLocal();
}

void CExampleTimer::After(TTimeIntervalMicroSeconds32 aInterval)
{
    ()Cancel
    Affirm(iStatus, aInterval);
    SetActive
}

void CExampleTimer::At(const TTime& aTime)
{
}

```

## Simple\_Timer\_implementation

```
    () Cancel
    AtTime(aTime);
    SetActive
}

void CExampleTimer::Inactivity(TTimeIntervalSeconds aSeconds)
{
    () Cancel
    Inactivity(iStatus, aSeconds);
    SetActive
}

void CExampleTimer::DoCancel()
{
    Cancel();
}

void CExampleTimer::RunL()
{
    TimerExpired(this, iStatus.Int());
}

```

## CExampleTimer.h

```
#include <E32BASE.H>

class MExampleTimerNotify
{
public:
    virtual void TimerExpired(TAny* aTimer, TInt aError) = 0;
};

class CExampleTimer: public CActive
{
public:
    static CExampleTimer* NewL(const TInt aPriority, MExampleTimerNotify& aNotify);
    ();CExampleTimer

public:
    void At(const TTime& aTime);
    void After(TTimeIntervalMicroSeconds32 aInterval);
    void Inactivity(TTimeIntervalSeconds aSeconds);
protected:
    void RunL();
    void DoCancel();
private:
    (const CExampleTimerPriority, MExampleTimerNotify& aNotify);
    void ConstructL(void);
private:
    RTimer iTimer
    MExampleTimerNotify iNotify
};

```

The basic .h file for the timer controller class, demonstrating how to use the timer

```
class CYourTimerController : public CBase, public MExampleTimerNotify
{
public:
    ....
}

```

## Simple\_Timer\_implementation

```
void TimerExpired(TAny* aTimer, TInt aError);

private :
    CExampleTimer;
};
```

### The basic .cpp file for the controller class

```
void CYourTimerController::ConstructL()
{
    iYourCExampleTimer::NewL(CActive::EPriorityStandard, *this);
    TTimeIntervalMicroSeconds32 someInterval; //you can call After/At/Inactivity depending on
    iYourTimer(someInterVal);
}

/**
 * Callback implementation when the timer activity happens in the CExampleTimer class
 */
void CYourTimerController::TimerExpired(TAny* aTimer, TInt aError)
{
    if(aError == KErrNone)
    {
        // Timer successfully completed, handle it
        CExampleTimer= (CExampleTimer*)aTimer;
        TTimeIntervalSeconds(10) seconds
        ->Inactivity(seconds); //Notify inactivity after 10 seconds
    }
}
```

The NewL/NewLC and other functions have been omitted as the implementation details would vary from

**Note :** When the system time changes, the At-timers will complete immediately with the result KErrAbort. So this must be handled by the applicaiton.