



ID	CS000992	Creation date	May 28, 2008
Platform	S60 3rd Edition, FP1	Tested on devices	Nokia N93
Category	Symbian C++	Subcategory	Files/Data

Keywords (APIs, classes, methods, functions): `RScheduler`, `TSchedulerItemRef`, `TTaskSchedulerCondition`, `TScheduleEntryInfo2`, `TTaskInfo`, `TTsTime`, `TScheduleState2`, `TScheduleType`, `RScheduler::Connect()`, `RScheduler::Register()`, `RScheduler::Close()`, `RScheduler::ScheduleTask()`, `RScheduler::GetScheduleTypeL()`, `RScheduler::GetScheduleL()`, `RScheduler::DeleteTask()`, `RScheduler::DeleteSchedule()`

Overview

This code snippet shows how to delete schedules and tasks using the class `RScheduler`. To be able to delete a schedule, it cannot contain any scheduled tasks, so the tasks must be deleted from it first. Tasks can be deleted from a schedule by using method `DeleteTask()` with a specific task ID. This method returns `KErrNotFound` if the task is not found. The method `DeleteSchedule()` is used to delete schedules by giving it a schedule handle.

In this example new example tasks are created to time-based and condition-based schedules. The created task IDs are then used to delete these tasks. Finally this example shows how to delete all tasks one by one from a given schedule and after that the empty schedule is deleted with the method `DeleteSchedule()`.

This snippet can be self-signed.

MMP file

The following libraries are required:

```
LIBRARY          schsvr.lib
```

Preconditions

Before this code snippet can be executed, `ExampleTaskHandler.exe` must be created and time-based and condition-based schedules with example tasks must be scheduled. See code snippets [CS000986 - Creating and](#)

registering a task handler with RScheduler, CS000987 - Creating persistent and transient schedules with RScheduler, and CS000988 - Creating a condition-based schedule with RScheduler for more information.

Resource files

.RSS

```
RESOURCE MENU_PANE r_schedulerexample_menu
{
    items =
        {
            //...
            MENU_ITEM {command = EDeleteExample;           txt = "DeleteExample";},
            MENU_ITEM {command = EAknSoftkeyExit;         txt = "Exit";}
        };
}
```

.hrh

```
enum TSchedulerExampleIds
{
    //...
    EDeleteExample
};
```

Header file

```
#ifndef __SCHEDULEREXAMPLEAPPUI_H__
#define __SCHEDULEREXAMPLEAPPUI_H__

#include <csch_cli.h> // RScheduler
#include <schinfo.h> // TSchedulerItemRef, TTaskInfo...

class CSchedulerExampleAppUi : public CAknAppUi
{
    //...
public:
    void HandleCommandL(TInt aCommand);
    //...
private:
    TInt ScheduleNewTaskL(TInt aScheduleId, RScheduler& aScheduler, TInt& aNewTaskId);

    TInt DeleteConditionBasedScheduleTaskByIdL(RScheduler& aScheduler,
                                                TInt aScheduleId,
                                                TInt aTaskId,
                                                TBool& aDeleted);

    TInt DeleteTimeBasedScheduleTaskByIdL(RScheduler& aScheduler,
                                           TInt aScheduleId,
                                           TInt aTaskId,
                                           TBool& aDeleted);

    TInt DeleteAllScheduleTasksL(RScheduler& aScheduler, TInt aScheduleId);

    //...
private:
```

Preconditions

```

RScheduler iScheduler;

TSchedulerItemRef iPersistentScheduleHandle;
TSchedulerItemRef iConditionScheduleHandle;
};

#endif // __SCHEDULEREXAMPLEAPPUI_H__

```

Source file

```

void CSchedulerExampleAppUi::ConstructL()
{
    //...
    User::LeaveIfError(iScheduler.Connect());

    _LIT(KExampleTaskHandlerExe, "ExampleTaskHandler.exe");
    TFileName exampleHandler(KExampleTaskHandlerExe);

    User::LeaveIfError(iScheduler.Register(exampleHandler, CActive::EPriorityStandard));
}

CSchedulerExampleAppUi::~CSchedulerExampleAppUi()
{
    //...
    iScheduler.Close();
}

void CSchedulerExampleAppUi::HandleCommandL(TInt aCommand)
{
    TBuf<100> Text1; //first line of dialog text
    TBuf<100> Text2; //second line of dialog text

    switch(aCommand)
    {
        case EEikCmdExit:
        case EAknSoftkeyExit:
            Exit();
            break;
    //...
    case EDeleteExample:
        {
            TInt err(KErrNone);

            TInt timeTaskId=-1;
            TInt conditionTaskId=-1;

            TBool timeTaskDeleted=EFalse;
            TBool conditionTaskDeleted=EFalse;

            //schedule a new example task to the time-based schedule and store the task ID
            err = ScheduleNewTaskL(iPersistentScheduleHandle.iHandle, iScheduler, timeTaskId);

            Text1.Append(_L("TimeTask:"));
            Text1.AppendNum(timeTaskId);
        }
    }
}

```

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

```
if(err == KErrNone )
{
    //delete the example task you just created using the stored ID
    err = DeleteTimeBasedScheduleTaskByIdL(iScheduler,
iPersistentScheduleHandle.iHandle, timeTaskId, timeTaskDeleted);

    if(err == KErrNone && timeTaskDeleted)
    {
        //task deleted succesfully...
        Text1.Append(_L("del OK\n"));
    }
    else
    {
        //delete task failed...
        Text1.Append(_L("del NOK\n"));
    }
}

//schedule a new example task to the condition-based schedule and store the task ID
err = ScheduleNewTaskL(iConditionScheduleHandle.iHandle,
iScheduler, conditionTaskId);

Text1.Append(_L("CondTask:"));
Text1.AppendNum(conditionTaskId);

if(err == KErrNone )
{
    //delete the just created example task using the stored ID
    err = DeleteConditionBasedScheduleTaskByIdL(iScheduler,
iConditionScheduleHandle.iHandle, conditionTaskId, conditionTaskDeleted);

    if(err == KErrNone && conditionTaskDeleted)
    {
        //task deleted succesfully...
        Text1.Append(_L(" del OK\n"));
    }
    else
    {
        //delete task failed...
        Text1.Append(_L(" del NOK\n"));
    }
}

//delete all tasks from the given schedule
err = DeleteAllScheduleTasksL(iScheduler, iPersistentScheduleHandle.iHandle);

if(err == KErrNone)
    Text2.Append(_L("del all OK\n"));
else
    Text2.Append(_L("del all NOK\n"));

//delete the given schedule
err = iScheduler.DeleteSchedule(iPersistentScheduleHandle.iHandle);

if(err == KErrNone)
    Text2.Append(_L("del sch OK\n"));
else
    Text2.Append(_L("del sch NOK\n"));

CEikonEnv::Static()->InfoWinL(Text1, Text2);
}
break;
```

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

```
default:
    //Panic(ESchedulerExampleUi);
    break;
}
}

TInt CSchedulerExampleAppUi::ScheduleNewTaskL(TInt aScheduleId,
                                             RScheduler& aScheduler, TInt& aNewTaskId)
{
    TInt ret(KErrNone);

    TTaskInfo taskInfo;
    taskInfo.iTaskId      = 0;
    taskInfo.iName       = _L("NewTask");
    taskInfo.iPriority    = 2;
    taskInfo.iRepeat     = 0;
    HBufC* data          = _L("NewTaskData").AllocLC();
    ret = aScheduler.ScheduleTask(taskInfo, *data, aScheduleId);

    aNewTaskId = taskInfo.iTaskId;

    CleanupStack::PopAndDestroy(); // data

    return ret;
}

TInt CSchedulerExampleAppUi::DeleteConditionBasedScheduleTaskByIdL(
                                             RScheduler& aScheduler,
                                             TInt aScheduleId,
                                             TInt aTaskId,
                                             TBool& aDeleted )
{
    TInt ret(KErrNone);
    TScheduleType type;
    TBool deleted = EFalse;

    ret = aScheduler.GetScheduleTypeL(aScheduleId, type);

    if(ret != KErrNone)
    {
        aDeleted = EFalse;
        return ret;
    }

    if(type != EConditionSchedule)
    {
        //the given schedule id is not of a condition-based schedule
        aDeleted = EFalse;
        return KErrGeneral;
    }

    CArrayFixFlat<TTaskInfo>* tasks = new (ELeave) CArrayFixFlat<TTaskInfo>(1);
    CleanupStack::PushL(tasks);

    TTsTime time;
    TScheduleState2 state;
    CArrayFixFlat<TTaskSchedulerCondition>* conditions =
    new (ELeave) CArrayFixFlat<TTaskSchedulerCondition>(1);
    CleanupStack::PushL(conditions);
```

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

```
tasks->Reset();
ret = aScheduler.GetScheduleL(aScheduleId,
                             state,
                             *conditions,
                             time,
                             *tasks);
CleanupStack::PopAndDestroy(conditions);

for (TInt index = 0 ; index < tasks->Count(); index++)
{
    if( tasks->At(index).iTaskId == aTaskId )
    {
        ret = iScheduler.DeleteTask(tasks->At(index).iTaskId);
        (ret == KErrNone )?deleted = ETrue:deleted = EFalse ;
        break;
    }
}

if(deleted)
    aDeleted = ETrue;
else
    aDeleted = EFalse;

CleanupStack::PopAndDestroy(tasks);

return ret;
}

TInt CSchedulerExampleAppUi::DeleteTimeBasedScheduleTaskByIdL(RScheduler& aScheduler,
                                                             TInt aScheduleId,
                                                             TInt aTaskId,
                                                             TBool& aDeleted )
{
    TInt ret(KErrNone);
    TScheduleType type;
    TBool deleted = EFalse;

    ret = aScheduler.GetScheduleTypeL(aScheduleId, type);

    if(ret != KErrNone)
    {
        aDeleted = EFalse;
        return ret;
    }

    if(type != ETimeSchedule)
    {
        //the given schedule id is not time based schedule
        aDeleted = EFalse;
        return KErrGeneral;
    }

    CArrayFixFlat<TTaskInfo>* tasks = new (ELeave) CArrayFixFlat<TTaskInfo>(1);
    CleanupStack::PushL(tasks);

    TTsTime time;
    TScheduleState2 state;
    CArrayFixFlat<TScheduleEntryInfo2>* entries =
new (ELeave) CArrayFixFlat<TScheduleEntryInfo2>(1);
    CleanupStack::PushL(entries);
```

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

```
tasks->Reset();
ret = aScheduler.GetScheduleL(aScheduleId,
                             state,
                             *entries,
                             *tasks,
                             time);

CleanupStack::PopAndDestroy(entries);
TInt count = tasks->Count();
for (TInt index = 0 ; index < count; index++)
{
    if( tasks->At(index).iTaskId == aTaskId )
    {
        ret = iScheduler.DeleteTask(tasks->At(index).iTaskId);
        (ret == KErrNone )?deleted = ETrue:deleted = EFalse ;
        break;
    }
}

if(deleted)
    aDeleted = ETrue;
else
    aDeleted = EFalse;

CleanupStack::PopAndDestroy(tasks);

return ret;
}

TInt CSchedulerExampleAppUi::DeleteAllScheduleTasksL(RScheduler& aScheduler,
                                                    TInt aScheduleId)
{
    TInt ret(KErrNone);
    TScheduleType type;

    TTsTime time;
    TScheduleState2 state;
    CArrayFixFlat<TTaskInfo>* tasks = NULL;

    User::LeaveIfError(aScheduler.GetScheduleTypeL(aScheduleId, type));

    //the given schedule is time-based schedule
    if(type == ETimeSchedule)
    {
        tasks = new (ELeave) CArrayFixFlat<TTaskInfo>(1);
        CleanupStack::PushL(tasks);

        CArrayFixFlat<TScheduleEntryInfo2>* entries =
new (ELeave) CArrayFixFlat<TScheduleEntryInfo2>(1);
        CleanupStack::PushL(entries);

        tasks->Reset();
        User::LeaveIfError(aScheduler.GetScheduleL(aScheduleId,
                                                  state,
                                                  *entries,
                                                  *tasks,
                                                  time));

        CleanupStack::PopAndDestroy(entries);
        TInt count = tasks->Count();
```

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

```
for (TInt index = 0 ; index < count; index++)
{
    User::LeaveIfError(iScheduler.DeleteTask(tasks->At(index).iTaskId));
}

CleanupStack::PopAndDestroy(tasks);
}
//the given schedule is a condition-based schedule
else if (type == EConditionSchedule)
{
    tasks = new (ELeave) CArrayFixFlat<TTaskInfo>(1);
    CleanupStack::PushL(tasks);

    CArrayFixFlat<TTaskSchedulerCondition>* conditions =
new (ELeave) CArrayFixFlat<TTaskSchedulerCondition>(1);
    CleanupStack::PushL(conditions);

    tasks->Reset();
    User::LeaveIfError(aScheduler.GetScheduleL(aScheduleId,
        state,
        *conditions,
        time,
        *tasks));
    CleanupStack::PopAndDestroy(conditions);

    for (TInt index = 0 ; index < tasks->Count(); index++)
    {
        User::LeaveIfError(iScheduler.DeleteTask(tasks->At(index).iTaskId));
    }

    CleanupStack::PopAndDestroy(tasks);

}
//Cannot determine schedule type
else
{
    User::LeaveIfError(KErrGeneral);
}

return ret;
}
```

Postconditions

The new example tasks are created to time-based and condition-based schedules and then deleted. After this, all time-based schedule tasks are deleted one by one and finally the schedule itself is deleted.

See also

- [CS000986 - Creating and registering a task handler with RScheduler](#)
- [CS000987 - Creating persistent and transient schedules with RScheduler](#)
- [CS000988 - Creating a condition-based schedule with RScheduler](#)
- [CS000989 - Getting schedule and task info using RScheduler](#)

CS000992_-_Deleting_schedules_and_tasks_using_RScheduler

- [CS000990 - Getting schedule and task count using RScheduler](#)
- [CS000991 - Editing a schedule using RScheduler](#)