TaskSchedulerEngine

Travis Pettijohn

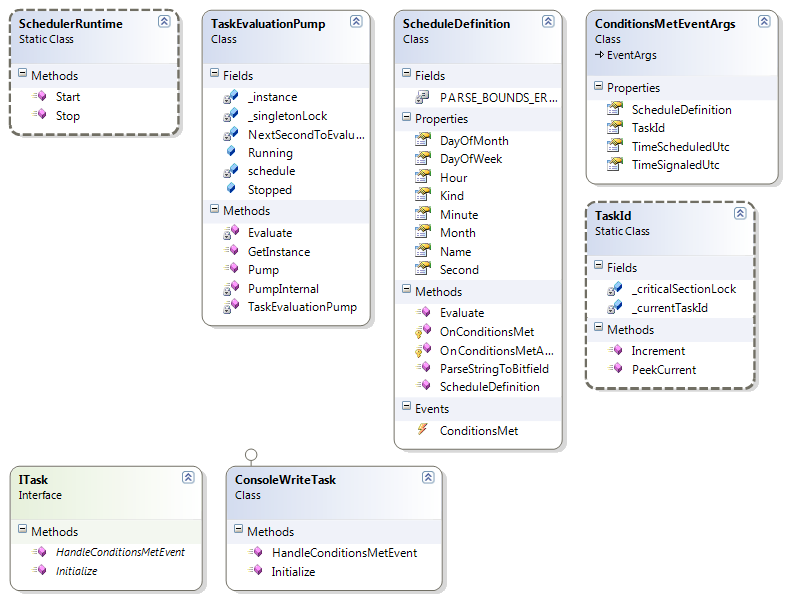
# Overview

A task scheduling engine for .NET. Executes in as finely as 1-second intervals. Configured much like **cron**, the schedule executes ITask events with a config-driven setup.

# Quick Start

* Create a class that implements ITask. This is the class that will respond to a scheduled event.
  + Initialize will be called when the schedule is created; it will only be called once for a given schedule.
* Set up a schedule, using either the fluent API or the XML configuration, instructing it to execute your ITask (you may set up multiple tasks to execute in the same schedule; keep calling Execute)
  + var s = new Schedule()  
     .AtSeconds(0, 10, 20, 30, 40, 50)  
     .WithLocalTime()  
     .Execute<ConsoleWriteTask>();
* Call Start with your schedule(s): SchedulerRuntime.Start(s);

# Runtime Components



SchedulerRuntime is a static class that contains one-line access to start & stop the scheduling engine.

TaskEvaluationPump is a singleton that pumps every second. Every time it pumps, it evaluates its Schedule to see if any of the events should be executed right now. PumpInternal simply loops on its own worker thread; it compares Now to the NextSecondToEvaluate and then sleeps until NextSecondToEvaluate occurs. When it wakes, it processes the schedule evaluations, adds one second to NextSecondToEvaluate, and sleeps again. The constructor constructs the Schedule from config and wires up delegates to tasks.

ScheduleDefinition is optimized for speed. While config uses easy-to-understand text evaluations (like “0,5,10”), ScheduleDefinition uses longs for bitfields (turning on the 0th, 5th, and 10th bits). This allows evaluations to be simple bitwise ANDs—if Now & Schedule != Zero then Now is a valid time to act for the schedule in question. In this case, a ConditionsMet event is raised on its own thread. The EventArgs include a TaskId, which is an auto-incrementing integer that counts the total number of tasks that have been executed by this application.

# Threading

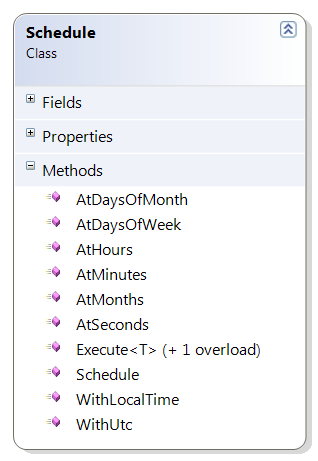
The TaskEvaluationPump runs on its own thread. Start and Stop it via the SchedulerRuntime.

In ScheduleDefinition, a ConditionsMet event is raised on its own thread via the OnConditionsMetAsync method (as called by the Evaluate method). Each of the ITasks that are connected to the ConditionsMet event will be executed in series on the worker thread.

For a given ScheduleDefinition, each associated ITask will have one instance

In configuration-speak: If an “At” schedule has multiple “Tasks” associated with it, a new thread will be created to execute all of the Tasks when the schedule’s conditions are met. This means that each of Tasks will be executed in series, sequentially as they are defined in config. If you wish to execute multiple tasks in parallel in the same schedule, you will have to define duplicate schedules.

# Fluent API



The fluent API allows you to programmatically set up your scheduling, and is considerably simpler to use than the XML configuration given the benefits of strong-typing and Intellisense. Here’s an example of a task that executes every ten seconds:

var s = new Schedule()

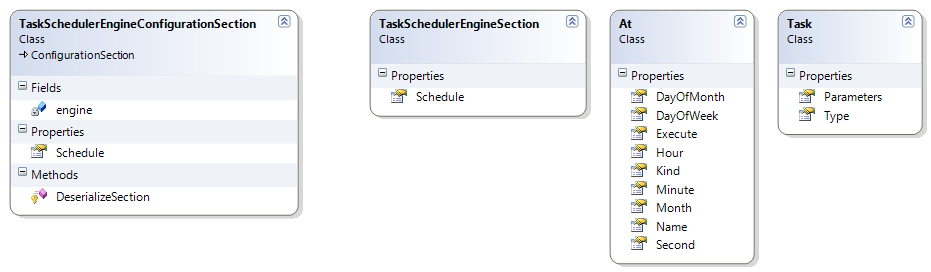
.AtSeconds(0, 10, 20, 30, 40, 50)

.WithLocalTime()

.Execute<ConsoleWriteTask>();

SchedulerRuntime.Start(s);

# Configuration and Usage



TaskSchedulerEngineConfigurationSection is the hook into the App.Config. The remaining classes are simply serialization assemblies so that config is human-readable. Sample configuration follows.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="taskSchedulerEngine"

type="TaskSchedulerEngine.Configuration.TaskSchedulerEngineConfigurationSection, TaskSchedulerEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

</configSections>

<taskSchedulerEngine>

<schedule>

<at name="primaryKeyName"

month="\*"

dayOfMonth="\*"

dayOfWeek="\*"

hour="\*"

minute="\*"

second="0,10,20,30,40,50"

kind="Utc">

<execute>

<task type="TaskSchedulerEngine.ConsoleWriteTask, TaskSchedulerEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"

parameters="" />

</execute>

</at>

</schedule>

</taskSchedulerEngine>

</configuration>

</configuration>

Note that comma-separated values can be specified for all of the “at” attributes (if an attribute is missing or blank, it is considered ‘\*’).

**Valid values** (default is ‘\*’ unless specified otherwise)**:**

* Month: 1-12
* Day of Month: 1-31
* Day of week: 0 (Sunday) to 6 (Saturday)
* Kind: **UTC** or **Local**. Default is UTC.
* Hour: 0 to 23
* Minute and second: 0 to 59.

**Note that there is no runtime validation**—you can create a schedule to run at the 63rd second of the 63rd minute of the 63rd day of the 63rd month—it will be loaded into the runtime but will never execute. (As there are 64-bits in a long, all of these can take any combination of values from 0-63.)

The attribute **Name** must be unique.

Task can be any ITask.

# At Examples

## Execute a task every second

<at name="scheduleName" month="\*" dayOfMonth="\*" dayOfWeek="\*" hour="\*" minute="\*" second="\*" kind="UTC" />

## Execute a task every five seconds

<at name="scheduleName" month="\*" dayOfMonth="\*" dayOfWeek="\*" hour="\*" minute="\*" second="0,5,10,15,20,25,30,35,40,45,50,55" kind="UTC" />

## Execute a task every minute

This more explicitly reads, execute a task on the 0th second of every minute.

<at name="scheduleName" month="\*" dayOfMonth="\*" dayOfWeek="\*" hour="\*" minute="\*" second="0" kind="UTC" />

## Execute a task at 3:00:00am and 3:00:00pm every Sunday

<at name="scheduleName" month="\*" dayOfMonth="\*" dayOfWeek="0" hour="3,15" minute="0" second="0" kind="UTC" />

## Never execute this task

Probably not something you would intend to do; but—it is legal syntax. Execute this task at 3am on the 31st of February.

<at month="2" dayOfMonth="31" dayOfWeek="\*" hour="3" minute="0" second="0" kind="UTC" />