Job Scheduler 3.1 Component Specification

1. Design

The Job Scheduling Component enables the timed execution of specified tasks. This functionality is similar to the unix cron utility and variants that exist on most operating system. Users can schedule both one-time and repeating tasks.

In the second version of this component, the concepts of Job dependency and grouping were introduced to create a more robust capability. Jobs could be created that are dependent on other jobs, and to more easily perform jobs by grouping them. The third version, as proposed by this design, is to separate the scheduling part of the design from the processing part. This new scheduler will be solely responsible for managing the jobs, including their persistence. The processor will be moved to a separate component.

Versions 1.0 and 2.0

A job is a specific O/S Command/java class to be launched on a particular schedule. The Job information is stored in a configuration file. Jobs can be created either through this component or through manual edit of the configuration file.

A job can be of two specific types. The first – External – are for operating system executables and scripts. This component uses the Executable Wrapper to execute these. The second – Internal – are Java classes implementing both the Runnable and Schedulable interfaces, the later being in this component. The component uses reflection to load and execute these classes.

Support for multiple schedules is provided by re-adding a job with a different name and a new schedule.

Both internal and external jobs are executed asynchronously. This permits simultaneous and overlapping jobs.

The Logging Wrapper component is used to log the results (run start, run completion, status) of each launched job. This provides an essential audit trail of all execution attempts and results.

A job can be dependent on another job regarding execution time. For example Job B does not have a configured date time. It is configured to run on the successful completion of Job A. Allow the option to execute Job B even if Job A fails. Also allow configuration of an option time delay before Job B executes after Job A. However, a Job must have either a scheduled date time or a task dependency but not both.

The component allows the set up of an email alert if a job returns a failure or cannot be executed. Configuration is at the job or job group level.

A job group is a grouping of jobs to simplify job scheduler configuration. A group can contain one or more jobs. A job can belong to one or more groups. The job groups are used to ease "Alert Notification Configuration".

Versions 3.0

Version 3.0 will incorporate the following functionality:

1) The processing will be removed from the Scheduler class and moved to a separate component.
2) The scheduling of a job will be enhanced beyond the use of Calendars and simple intervals.
3) Giving the scheduler CRUD functionality.
4) Retaining as much of the current design

1) Refactoring

The first point will be fairly straightforward. The Current design already has a separate processor entity. What the new design will do is refactor all Scheduler tasks that deal with running the jobs strictly to the processor, and strictly deal with the persistence of the Jobs. As such, the current relationship between the Scheduler and the Processor will be reversed: Now, the processor will query the scheduler for Jobs, instead of the Scheduler feeding the Jobs to the Processor as they are added.

2) Scheduling

The second point will be accomplished by expanding the interval fields of the Job to be able to handle more complicated combinations of job scheduling. The current design only allows for specific interval units: Second, Minute, Hour, Day, Week, Month, and Year. Furthermore, these are strictly relative to the start date. The new design will allow more elaborate units, such as specific days of the week or month, or year, as well as combinations of these, and they would be tied to the calendar, not the start date. This will be accomplished by substituting the current design's intervalUnit use of Calendar constants with a hierarchy of DateUnit classes, some of whom are marker extensions only: Second, Minute, Hour, Day, Week, Month, Year, DaysOfWeek, WeekOfMonth, DayOfMonth, WeekMonthOfYear, and DayOfYear. The processor will be responsible for knowing how to interpret this.

Second, Minute, Hour, Day, Week, Month, Year: Marker extension of DateUnit that act in the same manner as in the current design's Calendar constants. Thy simply represent an specific amount of time to wait before the next job run.

DaysOfWeek: Defines specific days of the week when the Job is to be run. It can define one to seven days of the week. There will exist two convenience implementations that encompass weekdays and weekends.

WeekOfMonth: Defines a specific day of the week in a month. For example, one can define the first Saturday of a month. This might be ideal for setting up notifications for a meting that must occur on a specific day of the business week but also at the start of a month.

DayOfMonth: Defines a specific day in a month. For example, one can define the 15th day of a month. This could be used for monthly check generation for disgruntled TopCoder designers.

WeekMonthOfYear: Defines a specific day of the week in a month in a given year. For example, one can define the first Saturday of January. This might be ideal for post-new year’s celebrations pink slip generation for rowdy employees.

DayOfYear: Defines a specific day in a month. For example, one can define the 15th day of a year. This could be similar to the WeekMonthOfYear date unit, except when the date into the year must be consistent regardless of the vagaries of the calendar with its leap years.

Apart from date units, it is important to also incorporate the idea of intervals and recurrences. The first item is already incorporated in the current design, and it allows for the schedule to occur every number of date units. For example, we might want something to occur on the first Saturday of a month, but every second month.

The other concept of recurrence simply means that we might want a Job to run a specified amount of times. This might work in lieu of a end date, or with an end date to indicate something like: “Do this on the Saturday of every second week five times or until March 31st”

3) CRUD functionality

The current scheduler already incorporates CRUD operations, so this will not change fundamentally. What will change is that the CRUD operations will occur via an interface. The scheduler itself will not be affected, but the processor that will use it will not be tied to a specific implementation, as it will be accessed via a interface using Strategy, but that is more in the scope of the processor design.

4) Current design

It is very much desired to keep the changes to the code as few as possible. Fortunately, this is very possible here. The Job class will mostly not change, except in that the protected methods will be made public, so the processor can more easily work with them, and the interval information, as mentioned in section

3 above, will be enhanced. Most of the configuration will not change either. The link to the scheduler will be cut, though, and this will have some consequences.

In fact, only the Scheduler will be modified on a large scale. A new Scheduler interface will be created, and two implementations of it will be added, one that will read from ConfigManager as is done currently (it's writing ability will be left intact), and one that uses the Informix database. These scheduler implementations will have no interaction with the removed processor, and the methods to manage the running of jobs will be moved to the processor, such as start(), stop(), shutdown(), getJobExecutionList(), stopJob(), etc.

One of the aspects of the Job was the addition of the TriggerEventHandler. This, unfortunately, had a reference to the processor required to run it. As such, this event handler will be added by the processor at the time it reads the version 2.0 jobs. Also, Email alerts are fully configured independently of the Scheduler.

One final note about the persistence operations. When a Job or JobGroup is retrieved, it will contain the associated JobGroups and Jobs, respectively. But these will contain just the name of each, and not a recursive relationship. If the user wants to know more about them, they will have to retrieve them explicitly. This equally applies to creating, updating and deleting. When creating and updating a Job, for instance, the JobGroups must already exist, and will not be created. When deleting a Job, the JobGroups will not be deleted as well. This applies to CRU operations for JobGroups equally.

Version 3.1:

1. Two new job types: JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE and JOB_TYPE_SCHEDULED_ENABLE_OBJECT_FACTORY are added, which enable the Job class to create the running object from the Object Factory with configured values loaded via Configuration Manager and the pluggable ScheduledEnableObjectFactory.
2. Attributes management APIs are added to Job class to store the extra information generated during the job execution. And a new ScheduledJobRunner interface is added, which contains getJob / setJob methods to interact with the job instance.
3. The getRunningJob method is added to the Job class to retrieve the running job/object depending on the job type.
4. A ConfigurationObjectScheduler implementation is added to schedule jobs and job groups in ConfigurationObject.

NOTE: Changes are colored in BLUE, and new additions are colored in RED.
NOTE: Backward compatibility is required and kept for this update.

1.1 Design Patterns

Listener Pattern:
The EventHandler uses the **listener** pattern. It is used to send email alerts or trigger the dependent job to start in this version.

Strategy Pattern:
The ScheduledEnableObjectFactory interface and its implementations are strategies in the Job class to create ScheduledEnable object.

The Scheduler implementations will be used as **Strategies** by other components.

Type Safe Pattern:
This component uses the **Type Safe** pattern to define the Job Types, but these days, this does not really apply as anything but a fundamental pattern.

Factory Pattern:
The ScheduledEnableObjectFactory interface and its implementations implement this pattern to create objects implementing ScheduledEnable interface.

1.2 Industry Standards

None.

1.3 Required Algorithms

1.3.1 Existing algorithms

This section will elaborate existing algorithms.

Implementation:
When a job executes successfully, fails or even doesn’t start, the job will raise the corresponding event. The EmailEventHandlers are listening to the job and will send email alerts according to the event.

Here is the pseudo code:

```
if (event.equals(requiredEvent)) {
//1. Generate the message
NodeList msgdata = job.getMessageData();
Node[] nodes;
if (msgdata == null) nodes = new Node[2];
else nodes = new Node[msgdata.getNodes().size() + 2];

nodes[0] = new Field("JobName", job.getName(), null, true);
nodes[1] = new Filed("JobStatus", event, null, true);
copy the nodes in msgdata.getNodes() to the rest of the array.

TemplateFields data = new TemplateFields(nodes, template);
String message = DocumentGenerator.getInstance().applyTemplate(data);

// 2. send the email using EmailEngine.

        TCSEmailMessage email = new TCSEmailMessage();
email.setSubject(emailAlertSubject);
email.setFromAddress(emailFromAddress);
for each recipient in recipients {
        email.addToAddress(recipient, 0);
}
EmailEngine.send(email);

}
```

1.3.2 Logging

Logging is performed in four places in this component: ConfigurationObjectScheduler, DBScheduler, ConfigManagerScheduler, and EmailEventHandler. This section serves as an authoritative list of where and how logging should be performed, in lieu of putting this information in method documentation.

The entry and exit of all methods will be logged at TRACE level. Any exception is logged at ERROR level.

1.4 Component Class Overview

Scheduler

This interface defines the CRUD operations for Jobs and JobGroups. It defines two extensions: One for reading-only data from the ConfigManager, and the second for reading from and writing to a database.

Job

This class is the specific job instance to schedule. A Job must have either a scheduled date time or a task dependency but not both.

Two new job types are supported, and its setRunCommand method is changed to support them. Methods are added to manage the job attributes used to store extra information.

JobType

An enumeration of the two types of Jobs: External and Java Class. This class extends the Enum class form the Type Safe Enumeration component.

Two new job types are added: JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE and JOB_TYPE_SCHEDULED_ENABLE_OBJECT_FACTORY.

JobGroup

A job group is a grouping of jobs to simplify job scheduler configuration. A group can contain one or more jobs. A job can belong to one or more groups. In current version, the job groups are used to ease “Alert Notification Configuration".

Dependence

A job can be dependent on another job regarding execution time. This class represents this relationship. It has a dependent job name, dependent event and delay properties.

EventHandler

There are three event of a job: Not Started, Executed Successful, and Executed Failed. The EventHandler is designed to handle these events of jobs.

EmailEventHandler

It is used to send email alert notifications when a job raises an event matching the requiredEvent initialized in the constructor. The email message is generated by Document Generator component.

ScheduledJobRunner

This interface extends ScheduledEnable interface and it defines methods to get / set job for this runner, so implementation can interact with the job instance.

ScheduledEnable

The ScheduledEnable interface is the only required interface for classes run as a job in Job Scheduler 2.0. It extends Schedulable interface (v1.0) and Runnable interface, and forces the implementation classes to return running status and message data at runtime.

Schedulable

The Schedulabe interface is one of two required interfaces (the other being Runnable) for classes to be run as a job.

DateUnit

A marker interface representing a unit of time to be used in configuring the schedule of the Job. Many of the implementations will refer to simply intervals, such as every second, day, week, or so, based on the start date and time. Some will be more complex, however, like referring to a specific day of the year or month. The Job will use this to indicate a specific date or time to do the job, or a date or time from the start date to perform it. It will work in conjunction with the interval and recurrence to track when and how often the job is done.

Second

A marker extension of DateUnit that represents a base interval unit of a second. Jobs configured with it will be run in intervals of seconds, depending on the associated interval setting.

Minute

A marker extension of DateUnit that represents a base interval unit of a minute. Jobs configured with it will be run in intervals of minutes, depending on the associated interval setting.

Hour

A marker extension of DateUnit that represents a base interval unit of an hour. Jobs configured with it will be run in intervals of hours, depending on the associated interval setting.

Day

A marker extension of DateUnit that represents a base interval unit of a day. Jobs configured with it will be run in intervals of days, depending on the associated interval setting.

Week

A marker extension of DateUnit that represents a base interval unit of a week. Jobs configured with it will be run in intervals of weks, depending on the associated interval setting.

Month

A marker extension of DateUnit that represents a base interval unit of a month. Jobs configured with it will be run in intervals of months, depending on the associated interval setting.

Year

A marker extension of DateUnit that represents a base interval unit of a year. Jobs configured with it will be run in intervals of years, depending on the associated interval setting.

DaysOfWeek

A marker extension of DateUnit that defines specific days of the week when the Job is to be run. It can define one to seven days of the week. There will exist two convenience implementations that encompass weekdays and weekends.

WeekOfMonth

A marker extension of DateUnit that defines a specific day of the week in a month. For example, one can define the first Saturday of a month. This might be ideal for setting up notifications for a meting that must occur on a specific day of the business week but also at the start of a month.

DayOfMonth

A marker extension of DateUnit that defines a specific day in a month. For example, one can define the 15th day of a month. This could be used for monthly check generation for disgruntled TopCoder designers.

WeekMonthOfYear

A marker extension of DateUnit that defines a specific day of the week in a month in a given year. For example, one can define the first Saturday of January. This might be ideal for post-new year's celebrations pink slip generation for rowdy employees.

DayOfYear

A marker extension of DateUnit that defines a specific day in a year. For example, one can define the 15th day of a year. This could be similar to the WeekMonthOfYear date unit, except when the date into the year must be consistent regardless of the vagaries of the calendar with its leap years.

ScheduledEnableObjectFactoryManager

This class contains several static methods to manage the mappings from string name to ScheduledEnableObjectFactory instances.

ScheduledEnableObjectFactory

This interface defines the contract to create the scheduled enable object (instance of ScheduledEnable).

ConfigManagerScheduler

A Scheduler implementation that uses the ConfigManager as the persistent data source. As in version 2.0, it will perform all CRUD operations.

As two new job types are added, so the pareJobType method in the SchedulerHelper used by this class should be changed to accept the new job types.

DBScheduler

A Scheduler implementation that uses the database as the persistent data source. It works with all CRUD methods. It uses connections obtained from the DBConnectionFactory for access and generate ids using the IDGenerator component.

As two new job types are added, so the pareJobType method in the SchedulerHelper used by this class should be changed to accept the new job types. The data type of JOB.JobType db table column is changed from VARCHAR(20) to VARCHAR(50) to hold the new job types.

ConfigurationObjectScheduler

This class implements the Scheduler interface, and it manages the jobs and job groups in a ConfigurationObject object from the Configuration API TCS component.

1.5 Component Exception Definitions

SchedulingException

This exception will be thrown when a Scheduler operation fails. This will pertain strictly to persistence operations on Jobs or JobGroups.

ConfigurationException

This exception will be thrown when there is a configuration problem in the scheduler constructors.

ScheduledEnableObjectCreationException

This exception extends the BaseException from the Base Exception TCS component, and it is thrown from the ScheduledEnableObjectFactory interface and its implementations if any error occurs when creating the running object.

1.6 Other Design Considerations

1.6.1 Configuration File Data Items

The Property Name Space is the job name.

Within each property name space the following name:value pairs are stored:

StartDTG: The start date / time for the job.
EndDTG: The end date/time for the job.
IntervalValue: The numeric interval value for the job.
IntervalUnit: The unit of time (see #4 above) for the interval value.
JobType: Whether the job is an external command or internal java class.
JobCmd: The job execution string (either external command or java class name, depending on the JobType value).

1.6.2 Invalid Jobs

The following conditions would make a job invalid during creation or modification:

a. A start date past the end date.
b. A name which already exists.
c. A name which is null or an empty String.
d. A negative or 0 increment value.
e. An increment unit value not following one of the DataUnit types outlined in the introduction
f. A job type not one specified in the Scheduler Object.
g. A null start date.

1.7 Thread Safety

This component is almost thread-safe. Scheduler, JobGroup, Dependence, and EmailEventHandler class are thread-safe. The only exception is Job class; some properties in it are mutable, so the user can make modifications and persist them. However, it is not anticipated that the same Job will be handled by more than one thread. Note that the newly added attributes variable is initialized as a synchronized map, as it needs to be accessed in multiple threads.

Also, the implementations of the Schedulable, ScheduledEnable, and ScheduledJobRunner interfaces should be thread-safe, that means when the job is running, the isDone, getRunningStatus etc methods can be called in thread-safe way.

The Map variable in ScheduledEnableObjectFactoryManager is initialized as synchronized Map, so this class is thread-safe. ScheduledEnableObjectFactory implementation should be thread-safe as it can be accessed from multiple threads simultaneously.

The ConfigurationObjectScheduler is thread-safe as its configuration variable is initialized as SynchronizedConfigurationObject, and use is required not to change it externally. So this class is thread-safe too.

2. Environment Requirements

2.1 Environment

. Development language: Java1.4
Compile target: Java1.4, Java1.5

2.2 TopCoder Software Components:

Configuration Manager 2.1.5

Used for text-based persistence of a schedule, as well as the configuration of the DB scheduler.

Document Generator 2.1

Used to generate the email message.

Email Engine 3.0

Used to send the email.

TypeSafe Enum 1.1

Provides type safe enum functionality for this pre Java 5 component.

Base Exception 2.0

Provides a uniform base exception class.

DB Connection Factory 1.1

Provides connections to the DBScheduler class.

ID Generator 3.0

Provides ID generation for the DBScheduler class.

Executable Wrapper 1.0

Provides access to command line execution for an external Job.

Logging Wrapper 2.0

Used by DBScheduler, ConfigManagerScheduler, and EmailEventHandler to log.

Object Factory 2.0.1

Used to create ScheduledEnable object using configured key.

Configuration API 1.0

Used in the newly added ConfigurationObjectScheduler implementation.

Note: The Alert Factory and Event Email Processor are not used. Because both of them need to persist the messages and are a little complex. In this component, it is not worth to add this complexity while using these two components.

2.3 Third Party Components:

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.scheduler.scheduling
com.topcoder.util.scheduler.scheduling.persistence

3.2 Configuration Parameters

3.2.1 ConfigManagerScheduler Configuration Parameters

(ConfigurationObjectScheduler uses the same configuration format, except that the sub-property in Configuration Manager is represented as a child ConfigurationObject in Configuration API.)

Parameter	Description	Values
<JobName>	Define a job	Can be any string except “DefinedGroups” and “Logger”

<JobName>.StartDate	The start date of the Job Required if this job is not dependent on another job.	A valid date
<JobName>.StartTime	The start time of the Job Required if this job is not dependent on another job.	A valid time of day, in milliseconds
<JobName>.EndDate	The end date of the job Required if this job is not dependent on another job.	A valid date, after start time
<JobName>.JobType	The type of the Job Required	JOB_TYPE_EXTERNAL, JOB_TYPE_JAVA_CLASS, JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE, or JOB_TYPE_SCHEDULED_ENABLE_OBJECT_FACTORY
<JobName>.JobCommand	The job command name Required.	If external, then a command name, if java class, then the name of the class.
<JobName>.Active	Flag whether the job is active or not Required	True or False
<JobName>.AsyncMode	Flag whether the job is async mode or not Optional The default value is True	True or False
<JobName>.Recurrence	A number stating how many time this job will be run Required	A positive number
<JobName>.Interval.Value	The value of the interval Required	Any positive number
<JobName>.Interval.Unit.Type	The class name of the DateUnit Required	A valid DateUnit class name
<JobName>.Interval.Unit.DateUnitDays	The day or days that makes up the date unit. If days, these will be comma-delimited. Required if the type demands it.	A positive integer valid for the date unit type.
<JobName>.Interval.Unit.DateUnitWeek	The week that makes up the date unit. Required if the type demands it.	A positive integer valid for the date unit type.
<JobName>.Interval.Unit.DateUnitMonth	The month that makes up the date unit. Required if the type demands it.	A positive integer valid for the date unit type.
<JobName>.ModificationDate	The date this Job was last modified. Required	A date
.<JobName>.Dependence	Specify the depended job, Optional	
<JobName>.Dependence.<dependedJobName>	The name of the depended job	Must have been defined in the configuration file
<JobName>.Dependence.<dependedJobName>.Status	Tell when the job should be started, on which status of the completion of the depended job, required	One of SUCCESSFUL, FAILED or BOTH
<JobName>.Dependence.<dependedJobName>.Delay	Specify a time delay before the job is triggered, optional.	Non-negative value, the unit is ms.
<JobName>.Messages	Define the message alerts of this job, Optional	
<JobName>.Messages.<Status>	When to send the email alert,	One of SUCCESSFUL,

	optional	FAILED and NOTSTARTED
<JobName>.Messages.<Status>. TemplateText	The name of the template text to generate message, optional. If it is not specified, the default template in scheduler will be used.	A Template text string
<JobName>.Messages.<Status>. FromAddress	The From Email Address of the message alerts, required	Email address
<JobName>.Messages.<Status>. Subject	The Subject of the message alerts, required	Subject
<JobName>.Messages.<Status>. Recipients	The Recipients of the message alerts, required	Email addresses
DefinedGroups	Define job groups, optional	
DefinedGroups.<GroupName>	The job group name, optional	
DefinedGroups.<GroupName>. Jobs	The jobs this group contained, required, at least one.	The name of defined jobs
DefinedGroups.<GroupName>. Messages	The message alerts configuration based on the group, similar with configuration on a single job. Optional	Similar with the configuration on a single job.
Logger	Name of logger	Optional. Defaults to fully-qualified name of ConfigManagerScheduler class in ConfigManagerScheduler. And it defaults to fully-qualified name of ConfigurationObjectScheduler class in ConfigurationObjectScheduler.

3.2.2 DBScheduler Configuration Parameters

Parameter	Description	Values
ConnectionFactoryClassName	Fully-qualified name of the DBConnection Factory	“com.topcoder.db.connectionfactory. DBConnectionFactoryImpl” Optional. Will default to the value above.
ConnectionFactoryNamespace	Namespace that the DBConnection Factory implementation uses.	“test” Required.
ConnectionName	Name of the connection to the persistence to get from the DBConnection Factory	“PersistenceConnection” Required.
IDGenSeqName	Named sequence for the tables. Used to retrieve an IDGenerator that can service it.	“SchedulerId” Required.
IDGenClassName	Name of the IDGenerator class that services the named sequence	“com.topcoder.util.idgenerator. OracleSequenceGenerator” Optional. Will attempt to find a generator already configured to handle the name sequence.
Logger	Name of logger	Optional. Defaults to fully-qualified name of DBScheduler class.

3.2.3 Job Configuration Parameters When its JobType is JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE

Parameter	Description	Values
ObjectFactoryNamespace	The namespace to create the ObjectFactory instance.	Any non-empty string. Required.
ScheduledEnableObjectKey	The key to create the ScheduledEnable object using ObjectFactory.	Any non-empty string. Required.

3.3 Dependencies Configuration

The developer should refer to the component specification of the used TopCoder components to configure them.

3.3.1 DDL for DBScheduler

```
CREATE TABLE JOB (
JobId INT NOT NULL PRIMARY KEY,
Name VARCHAR(40) NOT NULL,
StartDate DATE,
StartTime INT,
EndDate DATE,
DateUnit VARCHAR(60) NOT NULL,
DateUnitDays VARCHAR(20), // this one will be comma-delimited
DateUnitWeek VARCHAR(2),
DateUnitMonth VARCHAR(2),
Interval INTEGER NOT NULL,
Recurrence INTEGER NOT NULL,
Active CHAR(1) NOT NULL,
JobType VARCHAR(50) NOT NULL,  -- changed to hold the new job types
JobCommand VARCHAR(40) NOT NULL,
DependenceJobName VARCHAR(60),
DependenceJobStatus VARCHAR(20),
DependenceJobDelay VARCHAR(20),
AsyncMode VARCHAR(10)          -- changed to hold the async mode
);

CREATE TABLE Message (
MessageId INT NOT NULL PRIMARY KEY,
OwnerId INT NOT NULL,
Name VARCHAR(40) NOT NULL,
FromAddress VARCHAR(40) NOT NULL,
Subject VARCHAR(40) NOT NULL,
TemplateText VARCHAR(40),
Recipients VARCHAR(255) NOT NULL // this one will be comma-delimited
);

CREATE TABLE Group (
GroupId INTEGER NOT NULL PRIMARY KEY,
Name VARCHAR(40)
);

CREATE TABLE GroupJob (
GroupId INTEGER NOT NULL,
JobId INTEGER NOT NULL
);
```

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow demo.

4.3 Demo

4.3.1 Old Demo

This demo will show the use of the DBScheduler, since it works on all operations. The ConfigScheduler is worked on in the same manner, but it only has retrieve ops.

```
// Instantiate the DBScheduler, passing it the name of the config file
// containing job data.
Scheduler myScheduler = new DBScheduler(NAMESPACE);
```

```
// add new jobs.
// This job will start at 1 am on the 10th of March (GregorianCalendar months
// run from 0 to 11), and will run once a day, at 1 am, everyday until
// the 10th of March 2004 (inclusive).
Job deleteFiles = new Job("Nightly file cleanup", JobType.JOB_TYPE_EXTERNAL,
 "erase *.tmp");
deleteFiles.setStartDate(new GregorianCalendar(2003, 04, 10, 01, 00, 00));
deleteFiles.setStopDate(new GregorianCalendar(2004, 04, 10, 01, 00, 00));
deleteFiles.setIntervalUnit(new Day());
myScheduler.addJob(deleteFiles);

// Add a job dependent on another job regarding the execution time.
// jobA has a specific schedule time. jobB is dependent on jobA and
// jobC is dependent on jobB
Job jobA = new Job("jobA", JobType.JOB_TYPE_JAVA_CLASS,
 "com.topcoder.util.scheduler.scheduling.MyJob");
// Calendar representing the date
jobA.setStartDate(new GregorianCalendar(2003, 04, 10, 01, 00, 00));
// long representing a time of day
jobA.setStartTime(580);
// Calendar representing the date
jobA.setStopDate(new GregorianCalendar(2004, 04, 10, 01, 00, 00));
jobA.setIntervalUnit(new Day());
jobA.setIntervalValue(5);

myScheduler.addJob(jobA);

// another job with name jobB
Job jobB = new Job("jobB", JobType.JOB_TYPE_EXTERNAL, "dir");
// the delay is 10s
jobB.setDependence(new Dependence("jobA", EventHandler.SUCCESSFUL, 10000));
jobB.setIntervalUnit(new Week());
jobB.setIntervalValue(1);

myScheduler.addJob(jobB);

// another job, dependent on jobB, and configured for no delay
Job jobC = new Job("jobC", JobType.JOB_TYPE_EXTERNAL, "java -version");
jobB.setDependence(new Dependence("jobB", EventHandler.SUCCESSFUL, 0));
jobC.setIntervalUnit(new Month());
jobC.setIntervalValue(5);

// add email alert event handler to jobC, if the jobC executed
// unsuccessfully, an email will be sent to name1@topcoder.com
// the typical messageTemplate is like
//
// The Job %JobName% is %JobStatus%...
//
Log logger = LogFactory.getLog();
Template template = new XsltTemplate();
template.setTemplate("The Job %JobName% is %JobStatus%...");
EmailEventHandler handler1 = new EmailEventHandler(template, Arrays.asList(new
 String[] {"name1@topcoder.com",
"service@topcoder.com", "Failure of Job"}), EventHandler.FAILED,
 "admin@topcoder.com", "Notification", logger);
jobC.addHandler(handler1);

myScheduler.addJob(jobC);

// add a job group to scheduler
JobGroup group = new JobGroup("group_1", Arrays.asList(new Job[] {jobA, jobB,
 jobC}));

myScheduler.addGroup(group);

// add an email Event Handler to the group
// the following code means if any one of the jobs in the group executed
// successfully, an email alert will be sent to "name2@topcoder.com" and
// "name3@topcoder.com"
EmailEventHandler handler2 = new EmailEventHandler(template, Arrays.asList(new
 String[] {"name2@topcoder.com",
"name3@topcoder.com", "service@topcoder.com", "Success of Job"}),
EventHandler.SUCCESSFUL, "admin@topcoder.com", "Notification", logger);
group.addHandler(handler2);

// you can remove the handler from job and group, but you then have to
// update each
jobC.removeHandler(handler1);
group.removeHandler(handler2);

myScheduler.updateJob(jobC);
myScheduler.updateGroup(group);

// you also can remove job and group from the scheduler
myScheduler.deleteGroup(group.getName());
myScheduler.deleteJob(jobA);
```

4.3.2 Custom ScheduledJobRunner interacts with jobs of different job types

```
// note: synchronization is not shown for simplicity
// custom ScheduledJobRunner to interact with the job.
public class CustomScheduledJobRunner implements ScheduledJobRunner {

    /**
     * A flag denoting if the task is done.
     */
    private boolean done = false;

    /**
     * the job to interact with.
     */
    private Job job = null;

    /**
     * The job name.
     */
    private String jobName;

    /**
     * Returns <code>true</code> if the job is done, <code>false</code>
     * otherwise.
     * @return if the job is done
     */
    public boolean isDone() {
        return this.done;
    }

    /**
     * Closes the job.
     */
    public void close() {
    }

    /**
     * Gets the status of this job, Successful or Running.
     * @return the running status
     */
    public String getStatus() {
        return this.done ? SUCCESSFUL : RUNNING;
    }

    /**
     * Gets the message data.
     * @return always null.
     */
    public NodeList getMessageData() {
        return null;
    }

    /**
     * Gets the running status, Successful or Running.
     * @return the running status
     */
    public String getRunningStatus() {
        return getStatus();
    }

    /**
     * Set the job to interact with.
     * @param job the job to interact with
     */
    public void setJob(Job job) {
        this.job = job;
    }

    /**
     * Gets the job to interact with.
     * @return the job to interact with
     */
    public Job getJob() {
        return this.job;
    }

    /**
     * Runs the job.
     */
    public void run() {
        // put a attribute into job's attribute.
        job.setAttribute("name", "value");
        this.done = true;
    }

    /**
     * Gets the job name.
     * @return the job name.
     */
    public String getJobName() {
        return jobName;
    }

    /**
     * Sets the job name.
     * @param jobName the job name to set.
     */
    public void setJobName(String jobName) {
        this.jobName = jobName;
    }
  }

// custom ScheduledEnableObjectFactory
public class CustomScheduledEnableObjectFactory implements
        ScheduledEnableObjectFactory {

    /**
     * Represents the flag to indicate if the
     * <code>ScheduledEnableObjectCreationException</code> should be thrown in
     * <code>createScheduledEnableObject()</code> method.
     */
    private boolean throwException;

    /**
     * Creates the <code>SimpleScheduledEnableObjectFactory</code> instance.
     */
    public CustomScheduledEnableObjectFactory() {
    }

    /**
     * Forces the <code>createScheduledEnableObject()</code> method throw
     * <code>ScheduledEnableObjectCreationException</code>.
     */
    public void setThrowException() {
        this.throwException = true;
    }

    /**
     * Returns the <code>SimpleJob</code> instance for testing.
     * @return a <code>ScheduledEnable</code> instance.
     * @throws ScheduledEnableObjectCreationException if
     *             <code>throwException</code> is set <code>true</code>.
     */
    public ScheduledEnable createScheduledEnableObject()
        throws ScheduledEnableObjectCreationException {
        if (throwException) {
            throw new ScheduledEnableObjectCreationException("Test");
        }
        return new CustomScheduledJobRunner();
    }

}

// create a job with JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE job type.
// values are properly configured in “test_config” namespace to create the
// CustomScheduledJobRunner object
// create a job with JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE job
// type. the job will be created by the object factory by the given
// namespace.
Job job = new Job("test",
        JobType.JOB_TYPE_SCHEDULED_ENABLE_OBJECT_NAMESPACE,
        "com.topcoder.util.scheduler.scheduling.job");

// start the job
job.start();
// waiting for complete
Thread.sleep(100);
// stop the job
job.stop();

// get the execution result from job’s attributes
String value = (String) job.getAttribute("name");

// the value should be "value"
System.out.println(value);

// register CustomScheduledEnableObjectFactory into the manager
ScheduledEnableObjectFactoryManager.addScheduledEnableObjectFactory(
        "testFactory", new CustomScheduledEnableObjectFactory());

// create object with JOB_TYPE_SCHEDULED_ENABLE_OBJECT_FACTORY job type
// where the "testFactory" corresponds the factory name in
// ScheduledEnableObjectFactoryManager
job = new Job("test", JobType.JOB_TYPE_SCHEDULED_ENABLE_OBJECT_FACTORY,
        "testFactory");

// start the job
job.start();
Thread.sleep(100);
// stop the job
job.stop();

// get the value, the value should be "user"
value = (String) job.getAttribute("name");
System.out.println(value);
```

4.3.3 Schedule the job in ConfigurationObjectScheduler

```
// create the ConfigurationObjectScheduler
ConfigurationObjectScheduler scheduler = new ConfigurationObjectScheduler(
        new DefaultConfigurationObject("test"));

// create a new job
Job job = new Job("job", JobType.JOB_TYPE_EXTERNAL, "dir");
job.setActive(true);
job.setIntervalUnit(new Week());
job.setIntervalValue(2);
```

```
    job.setRecurrence(10);
    job.setModificationDate(new GregorianCalendar());
    List recipients = new ArrayList();
    recipients.add("recipients@topcoder.com");
    GregorianCalendar startDate = new GregorianCalendar();
    GregorianCalendar stopDate = (GregorianCalendar) startDate.clone();
    stopDate.add(Calendar.DATE, 10);
    job.setStartDate(startDate);
    job.setStartTime(3000000);
    job.setStopDate(stopDate);

    // add job
    scheduler.addJob(job);

    // change the job.
    job.setActive(false);

    // update the job
    scheduler.updateJob(job);

    // get back the job
    Job result = scheduler.getJob("test");
   // the other operations are the similar as those in section 4.3.1, the only
   // difference is the job is persisted in the configuration rather than db.
```

5. Future Enhancements

Add more EventHandlers to provide more functions in Job Scheduler.
Add custom implementations to ScheduledEnableObjectFactory, ScheduledJobRunner.