René Schumacher, Carsten Düllmann

Rene.Schumacher@Microsoft.com, Carsten.Duellmann@Microsoft.com

Abstract

This document describes the sample implementation of several TFS job extensions   
(background jobs) as well as a basic framework for job extension development.

PSfD Team Foundation Job Extensions Sample Documentation

Table of Contents

[1 Background 1](#_Toc415142385)

[2 Framework 1](#_Toc415142386)

[2.1 BackgroundJobBase 1](#_Toc415142387)

[2.1.1 DoWork (abstract, void) 1](#_Toc415142388)

[2.1.2 GetFriendlyJobName (abstract, string) 2](#_Toc415142389)

[2.1.3 GetLogLevelRegistryKey (abstract, string) 2](#_Toc415142390)

[2.1.4 LogWarning (void) 2](#_Toc415142391)

[2.1.5 LogError (void) 2](#_Toc415142392)

[2.2 JobContextAttribute 2](#_Toc415142393)

[2.3 JobDataResourceAttribute 3](#_Toc415142394)

[2.4 JobIdAttribute 3](#_Toc415142395)

[2.5 JobNameAttribute 3](#_Toc415142396)

[2.6 RegistryInfoResourceAttribute 3](#_Toc415142397)

[3 Background Job Samples 3](#_Toc415142398)

[3.1 Workspace Cleanup Job 3](#_Toc415142399)

[3.1.1 EmailTemplateSearchPath 4](#_Toc415142400)

[3.1.2 EmailTemplate 5](#_Toc415142401)

[3.1.3 WarningAgeInDays 5](#_Toc415142402)

[3.1.4 DeleteAgeInDays 6](#_Toc415142403)

[3.1.5 EmailSubject 6](#_Toc415142404)

[3.1.6 EmailPriority 6](#_Toc415142405)

[3.1.7 WarnIfNoEmailAddress 6](#_Toc415142406)

[3.1.8 DeleteIfNoEmailAddress 6](#_Toc415142407)

[3.1.9 LogLevel 7](#_Toc415142408)

[3.2 Shelveset Cleanup Job 7](#_Toc415142409)

[3.2.1 EmailTemplate 7](#_Toc415142410)

[3.2.2 EmailSubject 8](#_Toc415142411)

[4 Deployment 8](#_Toc415142412)

[4.1 Job Assemblies 8](#_Toc415142413)

[4.2 Team Foundation Custom Job Scheduler 9](#_Toc415142414)

Table of Figures

[Figure 1: Start screen of the Team Foundation Custom Job Scheduler 9](#_Toc415142415)

[Figure 2: Creating a schedule for a job 10](#_Toc415142416)

[Figure 3: Registry view for job configuration parameters 11](#_Toc415142417)

[Figure 4: Job schedule information of an existing job 11](#_Toc415142418)

# Background

Team Foundation Server uses background jobs (job extensions) for repetitive administrative tasks like regular consistency checks, building the data warehouse or cleanup of old data. Those jobs are run by the *Team Foundation Background Job Agent* either on a scheduled basis or on demand (mostly triggered by a TFS event). Using the *Team Foundation Client Object Model* it is possible to create custom background jobs.

This document describes a simple framework that encapsulates parts of the object model to further simplify custom background job development. In addition, it explains two sample jobs that automate typical administrative tasks. Lastly, the document shows how to use a custom tool to register and configure custom background jobs in Team Foundation Server.

# Framework

This section describes in detail the framework that was used to implement the sample jobs. The framework implementation can be found in the *Common* project of the *BackgroundJobs* solution. By referencing the *Microsoft.PSfD.TeamFoundation.BackgroundJobs.Common.dll* the framework can be used to implement additional background jobs.

## BackgroundJobBase

The abstract class *BackgroundJobsBase* is the foundation for all custom job extensions and implements the *ITeamFoundationJobExtension* interface provided by the TFS client object model. The base class simplifies job development by providing infrastructure code, which is needed to register and configure a custom job using the *Job Scheduler* (see 4.2). Furthermore, it provides unified access to logging functionality. Every custom job extension must be inherited from this class.

The following methods are used to implement custom background jobs:

### DoWork (abstract, void)

The abstract method *DoWork* contains the actual functionality of a custom background job.

The input parameter *requestContext* provides access to Team Foundation Server resources like the TFS registry or information about the TFS infrastructure. In addition to the *requestContext* the method gets a reference to the job’s definition read from the TFS database (parameter *jobDefinition*) as well as the time the job was queued (parameter *queueTime*).

Via the output parameter *result* the job’s execution result is handed back to the *Team Foundation Background Job Agent*. Possible job results are *Blocked* (the job is blocked due to a data constellation or configuration issue), *Failed*, *PartiallySucceeded* (the job succeeded but emitted warnings), *Stopped* (the job has been stopped by the TFS; this value should not be used), and *Succeeded* (the job has succeeded without warnings or errors). If *result* isn’t set within the *DoWork* method, it is automatically set by the base implementation based on the number of warnings and errors unless the parameter *skipFurtherResultAnalysis* is set to true. In case of an unhandled exception, the job status is automatically set to *Failed*.

To write additional information to the job history us the *resultMessage* parameter.

### GetFriendlyJobName (abstract, string)

The abstract method *GetFriendlyJobName* must return a unique and easy to understand name for the custom job. This name is used as part of the job’s log information.

### GetLogLevelRegistryKey (abstract, string)

This abstract method is used to provide the TFS registry key, which configures the job’s logging behavior. Typically, a registry key in the *Configuration* section is used, e.g. /Configuration/Shelve­setCleanupJob/LogLevel.

The base class parses the value of the registry key as an integer and uses it as the logging level. If the registry key does not exist, the lowest logging level is used. Allowed values are *0* (no logging except for unhandled exceptions as well as number of warnings and errors if any), *1* (logging of errors), *2* (logging of errors and warnings). All other values are ignored and result in the lowest logging level.

### LogWarning (void)

The two *LogWarning* overloads provide a simple mechanism for logging warnings with or without additional exception information. The method automatically evaluates the current logging level and keeps a counter for warnings, which is used in the summary information when logging is disabled.

### LogError (void)

Similar to the *LogWarning* overloads the two *LogError* methods provide a simple mechanism for logging errors with or without additional exception information. This method also keeps a counter for errors, which is used in the summary information, if necessary.

## JobContextAttribute

The optional *JobContext* attribute is part of the infrastructure code used by the *Job Scheduler* (see 4.2). It determines the scope of a custom background job, which can either be the Team Foundation Server instance (*JobContext.Server*) or a specific Team Project Collection *(JobContext.Collection*). If the attribute is not provided, the default value *JobContext.Any* is used, which allows configuring the job in the server as well as the collection scope.

In most cases, custom background jobs run in the collection context and should be marked with the appropriate *JobContext* attribute.

## JobDataResourceAttribute

The optional *JobDataResource* attribute is also part of the infrastructure code needed by the *Job Scheduler* (see 4.2).

In general, TFS job extensions can be configured in two different ways: using the TFS registry to store configuration data or storing XML data (also called *job data*) as part of the job definition or when queueing a job. The *JobDataResource* attribute allows to specify an assembly manifest resource that contains a template for such an XML artifact. The attribute value is the fully qualified name of the assembly manifest resource (e.g. *Full.Namespace.JobData.xml*). The *Job Scheduler* displays the XML data and allows the user to change it when registering a custom job extension (see *Job Data* field in Figure 2).

## JobIdAttribute

The *JobId* attribute is used to define a unique ID (GUID) for a custom background job. The *Job Scheduler* (see 4.2) as well as TFS use this ID to identify and schedule a job, so every custom job extension class must have this attribute.

## JobNameAttribute

The optional *JobName* attribute is again only used by the *Job Scheduler* (see 4.2) to display a user-friendly name for custom job (see Figure 2). If this attribute is not present, *Job Scheduler* displays the name of the custom job extension class.

## RegistryInfoResourceAttribute

The optional *RegistryInfoResource* attribute is similar to the *JobDataResource* (see 2.3) attribute and can be used to provide information about a job’s configuration options stored in the TFS registry. As with the *JobDataResource* attribute, the configuration information must be stored as XML in an assembly manifest resource that is specified using its fully qualified name (e.g. *Full.Namespace.RegistryInfo.xml*). The XML resource must comply with the *RegistryInfo.xsd* schema file located in the *Schemas* solution folder of the *BackgroundJobs* solution.

The *Job Scheduler* (see 4.2) uses the information to display the possible TFS registry keys and their default values in the *Registry* tab (see Figure 3).

# Background Job Samples

The following sections describe in detail the two sample job extensions, which have been implemented based on the aforementioned framework.

## Workspace Cleanup Job

The *Workspace Cleanup Job* demonstrates how an automated process to clean up outdated workspaces can be implemented as a TFS background job.

In large TFS installations a common problem is outdated workspaces that have not been deleted when users return their hardware or temporarily switch to different machines. Those workspaces use up valuable resources in TFS (e.g. database space), which can – over time – negatively impact the performance of a TFS instance. Thus, one of the standard jobs of an administrator of a large TFS installation is to identify and delete workspaces that have not been used for a long time. Unfortunately, there is no standard tool for this task and administrators need to resort to using the command line tool *tf.exe* (*workspaces* and *workspace* commands) or 3rd party tools like the *Sidekicks* provided by Attrice.

The *Workspace Cleanup Job* automates this task by regularly checking the date a workspace has last been accessed for all workspaces in the Team Project Collection. If a workspace hasn’t been used for more than *WarningAgeInDays* (see 3.1.3) days, the job sends out an email to the workspace owner to notify him of the pending deletion of his workspace. If the user still doesn’t use (or delete) the workspace it is automatically deleted without further notice once it becomes older than *DeleteAgeInDays* (see 3.1.4) days.

The job must always be registered in the context of a Team Project Collection and should run once a day. If possible, configure it to run during off-peak times.

Following are the configuration parameters of the *Workspace Cleanup job*. They are stored in the TFS registry under the path ***/Configuration/WorkspaceCleanupJob***. Even though the job is registered for a Team Project Collection, configuration parameters can be set in either the server-level registry or the collection-level registry. Collection-level parameter always override server-level parameters, thus, it is e.g. possible to specify a general value for *WarningAgeInDays*, while setting a different value for a single Team Project Collection.

### EmailTemplateSearchPath

This parameter configures the path in which the job looks for email template used to create the notifications sent to workspace owners (see also 3.1.2). To follow the general configuration pattern of Team Foundation Server the template should be stored in the *Transforms* subfolder of the *Team Foundation Background Job Agent* installation folder.

The default is “Transforms”.

### EmailTemplate

Use this parameter to specify the name of email template file. The template must be a valid XML transformation file (XSLT) that creates XHTML output. For an example look at the *WorkspaceCleanupTemplate.xsl* that is part of the *WorkspaceCleanupJob* project. The XML data used for the transformation looks like this:

<WorkspaceCleanup server="TFS" collection="TPC"

                  ownerDisplayName="Max Mustermann" owner="sample\mmuster"

                  dateGenerated="1.1.2014 10:00:00" warnAge="300" delAge="360">

  <Workspace computer="MaschineA" name="MaschineA" comment="Kommentar"

             type="Local" lastAccessed="1.2.2013 10:45:23" age="310" />

  <Workspace computer="MaschineA" name="MaschineA\_1" comment="Kommentar"

             type="Server" lastAccessed="1.2.2013 10:45:23" age="310" />

  <Workspace computer="MaschineB" name="MaschineB" comment="Kommentar"

             type="Local" lastAccessed="1.2.2013 10:45:23" age="310" />

</WorkspaceCleanup>

The different attributes can be used in the XML transform file and have the following meaning:

| Attribute | Meaning |
| --- | --- |
| server | Name of the TFS |
| collection | Name of the TPC that contains the workspaces |
| ownerDisplayName | Display name of the workspaces owner |
| owner | Username of the workspace owner |
| dateGenerated | Date and time the email was created |
| warnAge | The value of the *WarningAgeInDays* parameter (see 3.1.3) |
| delAge | The value of the *DeleteAgeInDays* parameter (see 3.1.4) |
| computer | Name of the machine the workspace resides on |
| name | Name of the workspace |
| comment | Comment used during workspace creation |
| type | Workspace location (either *Local* or *Server*) |
| lastAccessed | Date and time the workspace was last accessed |
| age | The current age of the workspace |

Table 1: XML attributes used in XML transformation by the Workspace Cleanup Job

The default is „WorkspaceCleanupTemplate.xsl“.

### WarningAgeInDays

This parameter configures the email behavior of the *Workspace Cleanup Job*. If a workspace hasn’t been used for more than the number of days defined by this parameter, the job sends out a warning mail to the workspace owner. The value must be lower than the value of *DeleteAgeInDays* (see 3.1.4), otherwise workspaces are deleted before a warning can be sent out.

The default is 300.

### DeleteAgeInDays

This parameter configures how workspaces are deleted by the job. If a workspace hasn’t been used for more the number of days defined by this parameter, the job deletes it. The value must be higher than the value of *WarningAgeInDays* (see 3.1.3), otherwise workspaces are deleted before a warning can be sent out.

The default is 360.

### EmailSubject

This parameter defines the subject used when sending emails.

The default is „ACTION REQUIRED - Workspace Cleanup Warning”.

### EmailPriority

This parameter specifies the email priority. Allowed values are *High*, *Normal*, and *Low*.

The default is „High“.

### WarnIfNoEmailAddress

This parameter configures the job’s behavior when it encounters workspace owners that do not have an associated email address. Usually, all users in TFS should have an email address that is taken directly from Active Directory. In addition, a user can change his or her preferred email address within the profile in TFS Web Access. If a user does not have an associated email address, the job is unable to send out warning emails to that user. This may result in unexpected deletion of a workspace. Thus, the job will by default generate a warning, whenever it cannot find the email address for a workspace owner. If you set this parameter to *False*, such a warning is not generated. In general, you should rarely need to change this parameter.

The default is „True“.

### DeleteIfNoEmailAddress

In addition to the *WarnIfNoEmailAddress* setting, this parameter also configures the job’s behavior in case there is no email address associated to a workspace owner. If you set this parameter to *True*, the job will delete workspaces, even if the owner does not have an associated email address.

**ATTENTION:** Setting this parameter to *True* may lead to unintentional and unnoticed loss in workspace data!

The default is “False”.

### LogLevel

This parameter configures the logging behavior of the *Workspace Cleanup Job*. There are three valid values:

* 0 – No logging except unhandled exceptions; if there are warnings or errors, a summary message is logged
* 1 – Errors only
* 2 – Errors and warnings

The default is 0.

## Shelveset Cleanup Job

The *Shelveset Cleanup Job* is very similar to the *Workspace Cleanup Job* and demonstrates how an automated cleaning mechanism for outdated shelvesets might look like.

In contrast to the number of workspaces per user, which is usually quite low, some users like to use a lot of shelvesets for backing up data, storing unfinished ideas, or doing code reviews with others. Due to the fact that shelvesets are not deleted when their data is unshelved, a user needs to remember to manually delete a shelveset that isn’t needed anymore. Usually, most users forget that last step, so it is very common to have a huge number of unused and outdated shelvesets. Since shelvesets can contain a large amount of data, TFS administrators need to regularly cleanup those shelvesets.

The *Shelveset Cleanup Job* is more or less identical to the *Workspace Cleanup Job*, but works on shelvesets instead of workspaces. As a result, most of the configuration parameters are identical and, as with the *Workspace Cleanup Job*, the job must also be configured in the collection context. It should be run once a day, if possible during off-peak times.

The configuration settings are kept in TFS registry under the path ***/Configuration/Shelve­setCleanupJob*** in either the server-level registry, the collection-level registry or both (see 3.1). Since most of the configuration parameters are identical to those of the *Workspace Cleanup Job* the following sections only describe the two parameters that have different default values.

### EmailTemplate

For a detailed description see 3.1.2.

The XML data used for transformation by the *Shelveset Cleanup Job* looks like this:

<ShelvesetCleanup server="TFS" collection="TPC"

                  ownerDisplayName="Max Mustermann" owner="sample\mmuster"

                  dateGenerated="1.1.2014 10:00:00" warnAge="300" delAge="360">

  <Shelveset name="Shelveset 1" comment="Kommentar"

             created="1.2.2013 10:45:23" age="310" />

  <Shelveset name="Shelveset 2" comment="Kommentar"

             created="1.2.2013 10:45:23" age="310" />

</ShelvesetCleanup>

The different attributes can be used in the XML transform file and have the following meaning:

| Attribute | Meaning |
| --- | --- |
| server | Name of the TFS |
| collection | Name of the TPC that contains the shelvesets |
| ownerDisplayName | Display name of the shelveset owner |
| owner | Username of the shelveset owner |
| dateGenerated | Date and time the email was created |
| warnAge | The value of the *WarningAgeInDays* parameter (see 3.1.3) |
| delAge | The value of the *DeleteAgeInDays* parameter (see 3.1.4) |
| name | Name of the shelveset |
| comment | Comment used during shelveset creation |
| created | Date and time the shelveset was created |
| age | The current age of the shelveset |

Table 2: XML attributes used in XML transformation by the Shelveset Cleanup Job

The default is „ShelvesetCleanupTemplate.xsl“.

### EmailSubject

For a detailed description see 3.1.5.

The default is „ACTION REQUIRED - Shelveset Cleanup Warning”.

# Deployment

The following sections describe the necessary steps to deploy and register custom TFS job extensions in a TFS infrastructure.

## Job Assemblies

Before a job can be registered in TFS, its assemblies and all its dependencies must be deployed to ***all application tier servers***. Job assemblies and non-GAC references must be put into the *Plugins* folder in the *Team Foundation Background Job Agent* installation folder, e.g. *C:\Program Files\Microsoft Team Foundation Server <version>\Application Tier\TfsJobAgent\Plugins*. To deploy the two sample jobs described in section 3, the following assemblies need to be deployed:

* Microsoft.PSfD.TeamFoundation.BackgroundJobs.Common.dll (framework)
* Microsoft.PSfD.TeamFoundation.ShelvesetCleanupJob.dll
* Microsoft.PSfD.TeamFoundation.WorkspaceCleanupJob.dll

Even though it is not necessary, you can also deploy the related debug symbols (pdb files) got get more detailed information in case of unhandled exceptions. To avoid file locking issues and ensure that new job assemblies are picked up by the *Team Foundation Background Job Agent* the service (*TfsJobAgent*) must be stopped before and started again after the deployment.

Keep in mind that some jobs require additional items to be deployed, e.g. the email templates for the *Workspace* and *Shelveset Cleanup Jobs* (see 3.1.2 and 3.2.1).

## Team Foundation Custom Job Scheduler

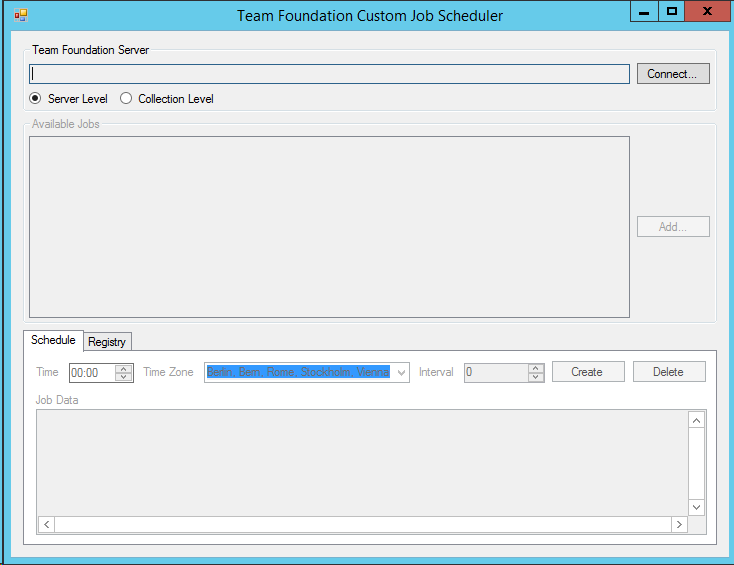
After the deployment of the assemblies, references and all other dependencies is done, every deployed job needs to be registered in Team Foundation Server. Since there is no standard tool to do such job registrations, a custom tool called *Team Foundation Custom Job Scheduler* is provided with the sample code, which helps with registering and configuring custom TFS job extensions. The tool needs to be run directly on a TFS application tier server (if you have multiple app tier servers, it doesn’t matter on which it is run) and starts as shown in Figure 1.

Figure 1: Start screen of the Team Foundation Custom Job Scheduler

To register a custom job, first connect to a Team Project Collection using the *Connect* button and then click *Add* to load all necessary job assemblies. To ensure that the required job assemblies have already been deployed to the *Plugins* folder, the tool always opens the *Plugins* folder, when trying to load job assemblies.

Every added assembly is inspected for custom TFS job extensions that have been developed based on the framework described in section 2, which are then listed in the *Available Jobs* list with their name, job ID and the supported job context. If a job is only supported in a single context, the tool will automatically switch the context as needed. If the job supports both server and collection context, make sure to select the desired context from the *Team Foundation Server* group.

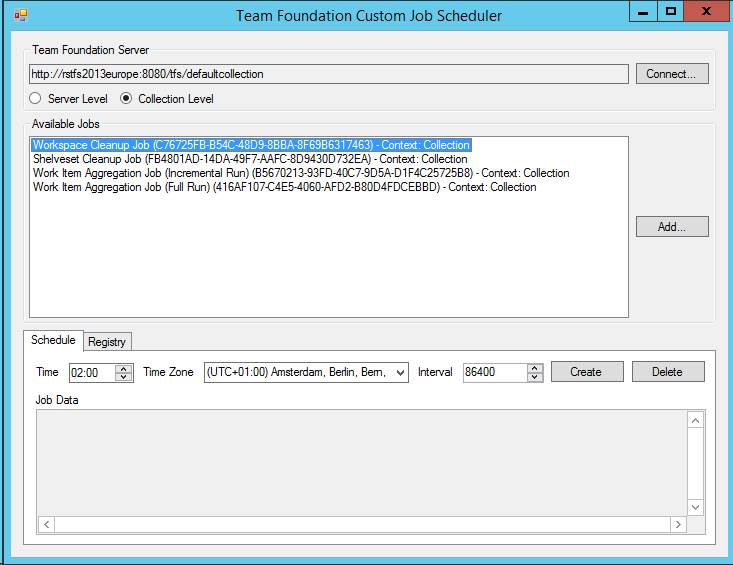
Once you click on a specific job, the *Schedule* and *Registry* tabs are enabled, the job data template (see 2.3), if it exists, is displayed in the *Job Data* box and the existing schedule, if there is one, is read from Team Foundation Server. To create a new schedule, enter the time (24h format) at which the job should initially run as well as the interval in seconds between job runs. Please note that only multiples of 24 hours or 86400 seconds can be used as the interval if a time zone other than UTC is selected. This is a limitation of the TFS job API. If you need to run a job multiple times a day, use UTC as the time zone and pay attention to the resulting time delta to your local time zone.

Figure 2: Creating a schedule for a job

If needed, change the displayed job data to reflect the desired configuration. Additionally, you can switch to the *Registry* tab (see Figure 3) and also change the job’s registry settings. Make sure to select the correct registry context (server or collection registry) first and then change the settings. The colored lines in the registry grid have the following meanings:

* **Gray text on yellow background, no checkmark**

Lines with this color show valid registry settings along with their default values for the selected job, which are not currently set in the TFS registry. This is just a convenience feature, so you don’t need to remember all available settings or default values.

* **Bold text, checkmark set**

Those colors are used for registry settings that have been changed or added by clicking the checkbox but have not yet saved into the TFS registry.

* **Regular text on white background, checkmark set**

Settings that look like this have actually been read from the TFS registry.

* **Crossed out text, no checkmark**

Crossed out lines have been marked for deletion by removing the checkmark. The setting will be deleted on the next save.

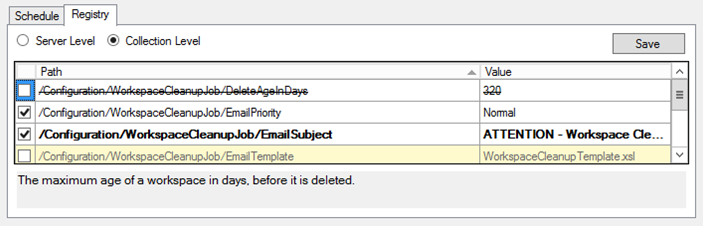


Figure 3: Registry view for job configuration parameters

After configuring all settings the job schedule is created using the *Create* button. If a job schedule already existed when the job was selected from the *Available Jobs* list, its schedule, job data and registry settings are read from TFS and displayed. Note that you cannot currently create multiple schedules for a single job. If you change the schedule, though, you can update it in TFS by clicking the *Update* button (see Figure 4).

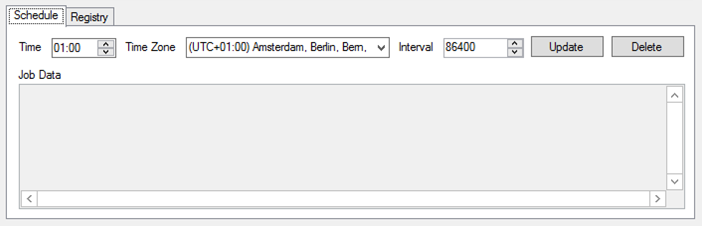


Figure 4: Job schedule information of an existing job

If you no longer need the job, make sure to remove the schedule by clicking the *Delete* button before you remove the job assemblies from your deployment.