Deltek

Deltek Maconomy® 2.5.4

Background Tasks Guide

September 24, 2021

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.

This edition published September 2021.

Contents

Overview

This document contains details on how to set up and use Background Tasks in Maconomy, including setup and configuration, a description of the main concepts, field descriptions, and troubleshooting.

In Maconomy, companies can set up tasks to run in the background (that is, without manual intervention and according to a predetermined schedule). There are two options:

- **Schedule Rules** – At scheduled intervals, Maconomy triggers a rule, and the rule generates one or more concrete tasks. Each task runs only once. The rule is triggered periodically, with each triggered instance generating a new set of the same concrete tasks.
- **Background Tasks** – These refer to standalone tasks that run once.
    - Certain tasks are well-suited to this functionality. For example, you can use schedule rules for periodically ensuring that all employee information is current, calculating To-Dos, and so on. As for background tasks, you can use these for the occasional task that has to take effect at a specific time. For example, if you need to update a specific system parameter's value by a certain date, you can set this up as a background task. You will not have to log in to Maconomy right before the target date to apply the change; Maconomy automatically does it for you.

> **Note:** While Maconomy does not strictly require scheduled background tasks to function, this feature greatly enhances efficiency by minimizing the tasks that users have to perform manually. It is recommended that companies fully utilize this feature to reduce the administrative demands they have to make on their employees.

Background Task Framework Configuration

The following contains information about how to configure the Background Task Framework in Maconomy. This functionality is available in all systems (standard, CPA, and PSO).

server.ini file

The Background Task Framework initial configuration is done by defining a number of properties in the server.ini file, located in the Coupling Service of a Maconomy System.

Change the different properties by opening the server.ini file in a text editor, complete the updates, and save the file. Some of the properties are maintained by MConfig and therefore must be changed using MConfig.

If a property is defined more than once in the server.ini file, the latest instance in the file is the definition used for the current property, though more specific definitions (by system nature and short-name; see below) may potentially override the value.

Comments

Define comments by prefixing the comment with a hashtag. The rest of the line is then a comment. For example:

```
# This is a comment
my.property = MyValue # This is also a comment
```

General Properties

Each parameter in the server.ini file is defined as a property name followed by an equal sign and a value of a specific type. Typically, property names consist of a number of sub-names divided by dots. For example:

```
batch.administrator.loginname = BatchAdministrator
```

This statement assigns the value (`BatchAdministrator`) to the property (`batch.administrator.loginname`).

System nature and short-name specific definitions of properties overwrite general values for a property.

Most of the properties used by the Background Tasks Framework can be defined in a system nature or short-name specific way.

System Nature Specific Properties

A system nature can be seen as a named grouping for one (or more) short-name(s) handled by a Coupling Service. See System Nature Setup for more information about System Natures.

Some properties are defined for a specific system nature.

Properties defined in scope of a specific system nature retain the format of the system nature property,as follows:

`<property-name>.nature.<system-nature-name> = …`

where `<property-name>` is the current property name

(such as, `batch.administrator.loginname`)

and `<system-nature-name>` is the name of the name of system nature (such as, `test`)

The complete definition of a property in a system nature scope could therefore be the following:

```
batch.administrator.loginname.nature.test = …
```

System nature-specific definitions of properties overwrite default values and general defined values for a property in the current system nature context.

Short-name Specific Properties

Properties which can be defined in a system nature specific way can also be defined in a short-name specific way, which enables even more specialization that the system nature specific definition.

For properties which can be defined in scope of a short-name,the format of the short-name property is as follows:

```
<property-name>.<short-name> = …
```

where `<property-name>` is the current property name (such as. `batch.administrator.loginname`)

and `<short-name>` is the name of the current short name (such as, `m21`).

The complete definition of a property in a short name scope could therefore be the following:

```
batch.administrator.loginname.m21 = …
```

Short-name specific definitions of properties overwrite default values, general defined values and system nature defined values for a property.in the current short-name context.

Mandatory Properties

Most properties have sensible default values. This means that if a property is not mentioned in the server.ini file it is still assigned a value from within the system.

A few properties are mandatory; this means that a value **must** be defined for the property for the system to work properly. A property either has a default value or requires mandatory input. For both situations, it is noted in the description whether the property is mandatory or what the default value is.

The following properties are mandatory (or recommend default value):

`mailserver.address`—Mandatory
`batch.administrator.loginname`—Recommend default value
`email.from.default`—Recommend default value
`batch.administrator.email.to`—Recommend default value
`email.to`—Strongly recommended default value defined for all **non-production** systems.

Immutable vs. Mutable Properties

Some properties in the server.ini file are **immutable** when the Coupling Service is running. This means that if an immutable property is changed and saved while the Coupling Service is running, the value is not picked up until next time the Coupling Service is restarted.

Other properties are **mutable**. If a mutable value is changed and saved during runtime, the new value is picked up after a short while (a few seconds), immediately applied, and used by the coupling service right away.

There is no way you can tell if a property is immutable or mutable; you need to know from the documentation below.

Background Task Framework Runtime Setup

Background Task Administrator Login Name Setup

batch.administrator.loginname

This property defines the administrator user for the Background Task Framework. This user orchestrates the batch execution system and is also the user to execute tasks when no specific user is assigned to a task.

The Background Tasks Administrator **must** be an administrator user in the current Maconomy-system and we recommended you create a dedicated administrator user (such as: `BatchAdmin`) for this purpose.

```
batch.administrator.loginname = <administrator-number>
```

The value syntax of this property is the “number” of the batch administrator user. For example,`BatchAdmin`.

Mandatory: No, but strongly recommended for all systems

Default value: `Administrator`

Change state: Immutable (Coupling service must be restarted for a change to take effect).

System nature and short-name specific versions:

```
batch.administrator.loginname.nature.<system-nature> = …
batch.administrator.loginname.<short-name> = …
```

Background Task Administrator Locale Setup

batch.administrator.locale

This property defines the locale used by the Batch Administrator user and the default locale used when executing tasks on behalf of specific users. Override the locale for specific users by setting the locale for users in the Users workspace, or use scheduled rules and tasks by setting a specific locale on the rule/task.

The locale affects logging and error information from the Background Task Framework in both clients and emails.

```
batch.administrator.locale = <locale-token>
```

The value syntax of the this property is a locale token, such as: `en_US`, `da_DK` or `en_US_MCS`

Default value: `en_US_W`

Change state: Immutable (Coupling service must be restarted for a change to take effect).

System nature and short-name specific versions:

```
batch.administrator.locale.nature.<system-nature> = …
batch.administrator.locale.<short-name> = …
```

Background Task Execution Control Setup

batch.enabled

This property enables and disables the Background Task execution. When the Background Task Framework is disabled, no tasks or scheduled rules are executed. Changes to this property are detected at runtime and take effect right away.

If the property `system.maintenance` is set to true, that setting overrules the setting of any `batch.enabled` property and the execution is disabled.

```
batch.enabled = <boolean>
```

The value syntax of the this property is a boolean, either `false` or `true`

Default value: `true`

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.enabled.nature.<system-nature> = …
batch.enabled.<short-name> = …
```

Background Task Framework Capacity

Task Poll Frequency Setup

`batch.delay.poll`

This property defines the delay in seconds (secs) between each check by a task execution thread for tasks due for execution. When an execution thread is assigned a due task, the Background Tasks Framework wakes up other threads, which immediately check for due tasks as well.

As long as due tasks exist, execution threads request a new task immediately after a task has finished. When no more tasks are due, all the task execution threads pause for the specified amount of time (secs) before an execution thread polls for due tasks again.

The unit of the value is seconds.

```
batch.delay.poll = <integer>
```

The value syntax of the this property is an positive integer (>0)

Default value: `20` (secs)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.delay.poll.nature.<system-nature> = …
batch.delay.poll.<short-name> = …
```

Number of Execution Threads Setup

`batch.threads`

This property defines number of execution threads running for any given short-name, unless one or more scheduled execution thread configurations (see below) are defined.

If any scheduled execution threads are defined (by `batch.threads.schedule)`, this value is only used until the scheduled execution threads configuration triggers the first time. When this has happened, the scheduled execution threads definition always has precedence over this property.

When a task execution thread has an active executiontask, it occupies at most one server. This means that an execution thread has the same impact on the system as ONE (to some extent, very active) user.

For this reason, the number of threads at most use the same number of servers (unless the total number of servers is limited by `batch.server.max` [see below]).

This property can, without restarting the Coupling Service, be changed in the interval between 0 and the explicit or implicit value of `batch.threads.max` (see below). If a higher value than `batch.threads.max` is defined for this property, the resulting value is limited to the `batch.threads.max`.

```
batch.threads = <integer>
```

The value syntax of the this property is an positive integer or zero (>=0)

Default value: `3` (threads per short-name)

Change state: Mutable. (can be changed within [0; `batch.threads.max`] without restarting the Coupling Service).

System nature and short-name specific versions:

```
batch.threads.nature.<system-nature> = …
batch.threads.<short-name> = …
```

Dynamic Number of Execution Threads Setup

`batch.threads.schedule`

This property defines the number of execution threads running for any given short-name at any given time. If no value is defined, the number of active threads are defined by `batch.threads`.

The value assigned to this setting is a list of pairs of thread counts and cron time trigger patterns, where values of thread count and pattern must be separated by acolon (:) and the value pairs must be separated by a semicolon (:).

Example:

```
batch.threads.schedule = 5 : 0 7,10 * * *; 10 : 0 18 * * * ; 15 : 0 22 * * *;
                         1 : 0 8 * * mon
```

This example makes the Background Task Framework use the following number of threads around the clock.

Threads	Time active
5	**7:00 - 18:00 (exception for Mondays, see line below)**
1	8:00 - 10:00 (only Mondays)
10	18:00 - 22:00 (6 p.m. - 10 p.m.)
15	22:00 - 7:00 (10 p.m. - 7 a.m.)

Whenever one of the time-patterns are fulfilled, it triggers the associated number of threads applied to the system.

If two patterns trigger at the same time, it is undefined which value is applied.

When a trigger pattern is activated, the applied number of threads remains active until another pattern triggers. This means that there are never more than one time trigger value-set defined, as this value set always remains active after triggering the first time.

```
batch.threads.schedule = (threads : cron-pat (; threads : cron-pat)*
```

The value syntax of this property is a semicolon separated list of number of thread (integer) and a colon followed by a cron-trigger pattern.

Default value: `<empty>` (no trigger defined)

Change state: Mutable. The full specification can be changed except increasing any number of execution thread count (in any new or existing trigger) above the value defined by explicit or implicit as `batch.server.max` without restarting the Coupling Service.

System nature and short-name specific versions:

```
batch.threads.schedule.nature.<system-nature> = …
batch.threads.schedule.<short-name> = …
```

Max ExecutionThreads Setup

`batch.threads.max`

This property defines the maximum number of execution threads which can run at any given time for any short-name in the current Coupling Service.

If a value is defined for this property, the given value defines the maximum number of execution threads as defined in settings for `batch.threds` and `batch.threads.scheduled`. If the value for any of these two properties is larger the max value, they are truncated to this value, both for values defined at startup and if any of these values are changed in the server.ini file during runtime.

If no value for `batch.threads.max` is expicitly defined, the max thread count for any definition of `batch.threads` and `batch.threads.scheduled` is the defined value for `batch.threads.max`.

Example:
```
batch.threads.schedule = 5 : 0 7,10 * * *; 15 : 0 18 * * * ; 10 : 0 22 * * *
batch.threads = 12
```
The resulting value of `batch.threads.max` is: `15`

By explicitly setting this value higher than `batch.threads`, you enable the possibility of raising the number of active threads at a later time without restarting the Coupling Service.

```
batch.threads.max = <integer>
```

The value syntax of the this property is an positive integer or zero (>=0)

Default value: The max value among the thread count for any definition of `batch.threads` and `batch.threads.scheduled`.

Change state: Immutable (Coupling service must be restarted for a change to take effect).

System nature and short-name specific versions:

```
batch.threads.max.nature.<system-nature> = …
batch.threads.max.<short-name> = …
```

Server Limitation Setup

`batch.server.max`

This property defines the maximum number of concurrent servers which are permitted for use by the Coupling Service when running background tasks. The Background Task Framework never uses more than `batch.server.max` for task execution at any time.

If the Background Task Framework uses the maximum number of servers (defined by `batch.server.max`), new tasks are held back for execution until another task is done using a server, which is thereby released the Background Task Framework.

If the value of this property is 0 (zero), this means that there is no limit to the number of servers which can be used by the Background Task Framework.

This property should only be used in configurations where many short-names are running on the same Coupling Service. For example, the Coupling Services is serving 20 short-names (customers) with 3 execution threads for each short-name. The server need peeks at (3+1 [for surveillance]) * 20 = 80 servers. By setting the `batch.server.max` to 10 for this Coupling Service, the number of concurrent servers used by the Background Tasks Framework (at any given time) is limited to 10 (instead of 80).

The default value is 0 (unlimited).

`batch.server.max = <integer>`

The value syntax of the this property is an positive integer or zero (>=0)

Default value: `0` (No limitation)

Change state: Immutable (Coupling service must be restarted for a change to take effect).

System nature and short-name specific versions:

```
batch.server.max.nature.<system-nature> = …
batch.server.max.<short-name> = …
```

Priority Distribution Setup

`batch.task.standard-priority.percentage`

When defining tasks and scheduled rules in the Background Tasks Framework, youcan set them to have either low or standard priority.

This property defines the percentage by which a standard priority task is given priority in execution over a low priority task (out of 100), even if the standard priority task has a later (but still fulfilled) due time.

For the rest of the cases (100 - `batch.task.standard-priority.percentage`) all task priorities (low and standard) are picked randomly, if they have same due time.

This distribution between low and standard priority tasks is used as long as there are tasks of both low and standard priority due for execution.

If only tasks of one of the priorities are due, these will use 100% of the capacity.

The minimum value of this property is 0 (zero). This means that tasks which are due are always picked without considering the priority of the tasks. Setting this value to zero thereby overrules the setting of task priority, as all tasks with same due time are handled equally.

The maximum value for the property is 90. In this case 9 out of 10 tasks executed are standard tasks, even though low-priority-tasks with an earlier due time exist. The reason we do not allow the value of 100 is that we do not want starvation of low priority tasks, which could happen if the value is set to 100.

`batch.task.standard-priority.percentage = <integer>`

The value syntax of the this property is an integer in the range `[0;90]`

Default value: `70` (%)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.task.standard-priority.percentage.nature.<system-nature> = …
batch.task.standard-priority.percentage.<short-name> = …
```

Background Task Framework Runtime Validation

System Check Frequency Setup

batch.system-check.interval

This property defines the interval (in seconds) between system integrity checks (sanity checks) done by the Background Tasks Framework itself. A system integrity check monitors the state of the batch execution system.

If any problem is detected by the framework, an email is sent to the batch administrator, and a notification is raised in the Management panel of the Background Task-workspace.

When the system integrity check is executed, one server is occupied for this.

System integrity checks are also done whenever the server.ini file is updated.

The unit of the value is seconds.

```
batch.system-check.interval = <integer>
```

The value syntax of the this property is an positive integer (>0)

Default value: `300` (secs)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.system-check.interval.nature.<system-nature> = …
batch.system-check.interval.<short-name> = …
```

Sanity Email Control Setup

batch.sanity.email.disabled

This property can enable/disable the Background Tasks Framework sanity validation emails. These emails are sent if a system integrity check done by the framework detects any problem regarding the Background Task Framework. When a problem is detected, an email is sent (and maybe later resent) if emailing is enabled by this property. If the email announced error is fixed without shutting the system down, a new email calling the error fixed will be sent at next integrity check by the framework.

If an error is detected before a connection to the Maconomy-server can be made, the global setting (`batch.sanity.email.disabled`) is used exclusively (as the nature and/or short-name setup is not known at this time). After the connection is made, the more specific values, such as (`batch.sanity.email.disabled.nature.<system-nature->`/ `batch.sanity.email.disabled.<shortname>`) ar used if defined. For this reason, the global property should always be true if you want to disable all e-mails – event during startup.

```
batch.sanity.email.disabled = <boolean>
```

The value syntax of the this property is a boolean, either `false` or `true`

Default value: `false` (Emailing enabled)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.sanity.email.disabled.nature.<system-nature> = …
batch.sanity.email.disabled.<short-name> = …
```

Sanity Start-up Email Setup

batch.sanity.email.announce-okay-startup

This property can enable/disable a Background Framework sanity check email on startup in cases of no errors.

If this property is true, an email is sent on Coupling Service startup to the Background Task Framework administrator in all cases, even if the system is fine and no errors are detected by the system integrity check. If this property is false, an email is only sent if the system integrity system detects an error.

```
batch.sanity.email.announce-okay-startup = <boolean>
```

The value syntax of the this property is a boolean, either `false` or `true`

Default value: `true` (emailing okay startup enabled)

Change state: *This feature is only used during startup* (Coupling service must be restarted for a change to take effect).

System nature and short-name specific versions:

```
batch.sanity.email.disabled.nature.<system-nature> = …
batch.sanity.email.disabled.<short-name> = …
```

Sanity Error Email Resent Frequency Setup

batch.sanity.email.resend-error.interval

This property defines the interval (in seconds) between each mail describing an already-reported error issue from the integrity check. A mail is sent when an error is detected by the Background Task Framework integrity system validation and a new mail is resent with an interval defined by this property. This repeats until the error has been addressed and removed or the Coupling Service is shut down.

If the value of this property is set to 0 (zero), no re-sending of error mails takes place (only the initial mail is sent).

The unit is in seconds.

```
batch.sanity.email.resend-error.interval = <integer>
```

The value syntax of the this property is an positive integer (>0)

Default value: `7200` (secs) = 120 min. = 2 h.

Change state: Mutable (Can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.system-check.interval.nature.<system-nature> = …
batch.system-check.interval.<short-name> = …
```

Background Task Framework Short-name Control

Short-name Inclusion Setup

batch.database.shortnames

This property defines the set of short-names on which the Background Tasks Framework run and thereby execute tasks.

The value of this property can be a list of (enabled) short-names or one of the tokens: `<all>` or `<none>`.

The token `<all>` is equal to the list of all short-names existing in the application.

The token `<none>` indicates that the list of short-names is empty. The Background Task Framework is not running on any short-name of the application if this token is used.

The list of short-names defined (implicit or explicit) by this property can be restricted by the property `batch.database.shortnames.exclude`.

```
batch.database.shortnames = short-name (; short-name)* | <all> | <none>
```

The value syntax of the this property is an a list of short-names separated by a semicolon (;)

Default value: `<all>` (all short-names are enabled to run background tasks)

Change state: *This feature is only used during startup* (Coupling service must be restarted for a change to take effect)

Short-name Exclusion Setup

`batch.database.shortnames.exclude`

This property can prevent short-names from having tasks executed by the Background Task Framework.

The short-names in this list are excluded from the list of active short-names (given implicit or explicit by `batch.database.shortnames`).These short-names are therefore not enabled, and thereby have no tasks executed by the Background Task Framework.

The token `<all>` is equal to the list of all short-names in this context. If this token is used as value for the property, NONE of the short-names in this Coupling Service will be included in the Background Task Framework execution.

The token `<none>` means that the list of excluded short-names is empty. Therefore, all the short-names defined by `batch.database.shortnames` will have task execution enabled as none of the shortnames in this context are excluded from Background Tasks Framework execution.

```
batch.database.shortnames.exclude = short-name (; short-name)*| <all>|<none>
```

The value syntax of the this property is an a list of short-names separated by a semicolon (;)

Default value: `<none>` (all short-names in this context are enabled to run background tasks)

Change state: *This feature is only used during startup* (Coupling service must be restarted for a change to take effect)

Email Setup

Mail Server Setup

`mailserver.address`

The mail-server address property specifies the mail-server used by the Maconomy runtime. This mail-server is used for sending sanity and task-specific emails from the Background Tasks Framework as well as sending emails when the "EmailOnAction" action is used in the application.

```
mailserver.address = <server-address>:<portnumber>
```

The value syntax of the this property is: `<server-address>:<portnumber>` such as, "`172.10.8.100:25`" or "`mailserver.trifolium.com:25`"

Mandatory for all systems: Yes

Change state: Mutable (Can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
mailserver.address.nature.<system-nature> = …
mailserver.address.<short-name> = …
```

Default Sender Email Address Setup

`email.from.default`

This property is used to specify a default sender for e-mails, (such as sent by the Background Task Runtime), invoke the application action: EmailOnAction, or send emails from Java extensions.

The default sender email address is only used in cases where no specific email sender is defined.

If no default email sender address is specified, either by `email.from.default` or `batch.email.from.default` (see below), no emails without a specific sender defined, will be sent.

```
email.from.default = <default-sender-email-address>
```

The value syntax of the this property is: `<default-sender-email-address>` such as, no_reply@trifolium.com

Mandatory: No, but strongly recommended for all systems

Default value: <empty> (no email addresses defined)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
email.from.default.nature.<system-nature> = …
email.from.default.<short-name> = …
```

Default Background Tasks Framework Sender Email Address Setup

`batch.mail.from.default`

This property is used to specify a default sender for e-mails sent by the Background Task Runtime. Defining this property overwrites the default email sender (see above) when sending emails from the Background Task Framework. Emails sent from other parts of the system that the Background Task Framework will not be affected by this property.

The default sender email address is only used in cases where no specific email-sender is defined.

```
batch.mail.from.default = <default-sender-email-address>
```

The value syntax of the this property is a single email address, such as batchAdmin@trifolium.com

Default value: the value defined by `email.from.default`

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.email.from.default.nature.<system-nature> = …
batch.email.from.default.<short-name> = …
```

Background Task Administrator Email Address Setup

batch.administrator.email.to, batch.administrator.email.cc, batch.administrator.email.bcc

This property defines one or more email address(es) for background task administrators or system administrator(s) surveying the background task execution. This/these emails are used when the Batch Sanity validation system needs to report an issue to the administrator(s).

If this property is not defined, you will have a notification in the Background Task workspace, Management tab.

If an error is detected before a connection to the Maconomy-server can be made, the system nature and short-name versions of this property is ignored. For this reason, the standard property (`batch.administrator.email.to`) should always be defined, even if system nature or short-name specific definitions covers all defined Maconomy systems on the current Coupling Service.

```
batch.administrator.email.to = <default-sender-email-address>
```

The value syntax of this property is one or more email addresses separated by ‘,’ (comma) or ‘;’ (semicolon) such as: batchAdmin@trifolium.com; administrator@trifolium.com

Mandatory: No, but strongly recommended for all systems

Default value: <empty> (no email addresses defined)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
batch.administrator.email.to.nature.<system-nature> = …
batch.administrator.email.to.<short-name> = …
```

It is also possible to add Cc and Bcc receivers to receive the same mail as the Background Task Administrators. For this purpose, the following properties exist in standard, system nature, and short-name versions:

```
batch.administrator.email.cc
batch.administrator.email.bcc
```

Overruling Receiver Email Address Setup

email.to

This property is used to OVERRULE any email recipient (including Cc and Bcc) for mails sent by the Maconomy system. This property is respected by the Background Task Runtime, the EmailOnAction action, emails send by Java extensions, and others.

If this setting is defined, current email recipients arereplaced by the email address specified by this property. This may be especially useful for non-production systems. For these, a system nature-specific property is used to ensure that emails are NOT sent to customers (and others, if applicable) from such a system, but are directed to a fixed internal email address.

```
email.to = <default-sender-email-address>
```

The value syntax of this property is an email address, most like in an in-house domain, such as test@trifolium.com

Mandatory: No, but strongly recommended for **all non-production** systems

Default value: <empty> (no email addresses defined)

Change state: Mutable (can be changed without restarting the Coupling Service)

System nature and short-name specific versions:

```
email.to.nature.<system-nature> = …
email.to.<short-name> = …
```

Miscellaneous

System Nature Setup

system.nature

Setting up databases by type / nature allows you to control how background tasks are carried out in test and production databases. Since background tasks are copied as part of a database replication for testing purposes, if you do not take measures to control the background tasks, they will carry out as scheduled. This could lead to such predicaments as inadvertently synchronizing data with third party systems, or sending customer emails from the test system as well as the production system. Instead, specify that a background task is only picked up by databases of a certain nature, such as Production or Test.

To facilitate this feature, first set up the needed system natures in the configuration file, and then activate background task assignment to a system nature in Workspace Client.

Additionally, you can set up precautions for certain background tasks so that these items behave differently based on the nature to which they are assigned. Precautions exist for file location, URLs, and email accounts so, for example, that you do not inadvertently send a client an email from a test system, or sync integration data from a test system.

Setup System Nature

You must first set up database natures so that you can then assign them to background tasks.

> **Note:** The system nature names are user-defined, though we recommend using names such **test**, **dev**, and **production**.

Additionally, all recommendations in this section define the *default* settings to reference the settings you want for the test nature. You must also define a specific setting for the production nature (and possibly other natures as well).

To set up database natures:

1. In the Coupling Service, go to the configuration file **server.ini**.
2. Enter a default nature to a system shortname that defines the configuration, such as **test** or **production**.

   For example:

   ```
   # default nature is test
   system.nature=test

   # the production database short name
   system.nature.macoprod=production


   # development databases belong to nature
   # development
   ```

```
system.nature.dev1=development
system.nature.dev2=development
```

> **Note:** While you may have more than one system shortname database of *test* or *dev* nature, you must have only one system assigned *production* nature. Additionally, assigning a system as *test* ensures that, unless a background task is assigned a different nature, its default assignment will be the *test* type.

Server Descriptive Name Setup

`server.description`

You may have Multiple Maconomy server instances. The server.description property enables you to assign a meaningful name/description to your Maconomy server instance for easy identification.

The property lets you insert certain placeholders to expand.

1. Open the **server.ini** file.
2. Enter the description of the server:
   - ^{shortname}—expands to the current shortname
   - ^{nature}—expands to the nature of the current shortname
   - ^{pid}—expands to the current process ID of the coupling service
   - ^{hostaddress}—host address defined by 'server.address'
   - ^{hostname}—(os-)name of the host machine
   - ^{canonicalhostname}—canonical (os-)name of the host machine
   - ^{webport}—communication port for clients connecting

Note: This property can be specified globally, by system nature, and/or shortname. Additionally, the global value can be overwritten by the system nature value which again can be overwritten by the shortname value. If the description property is not defined, a default value is generated. The default value is:

```
"^{hostname}:^{hostaddress}:^{webport}:^{nature}"
```

The ":^{nature}" is omitted if there is no nature specified for the current shortname.

* server.description = <server-description-macro-expression>

* server.description.nature.<system-nature-id> = <server.description>

* server.description.<shortname> = <server.description.nature.<system-nature-id>>

Examples:

```
server.description=App1Server
```

Gives the (static) description "App1Server" for this server.

```
server.description=App1Server@^{webport}
```

Gives the dynamic description "App1Server@*web-port-in-use*"

For example, if the coupling service uses port 8084 as it is a web.port, then the name is expanded to: "App1Server@8084" at run-time.

server.description=App1Server@^{webport} (^{pid})

Gives the dynamic description “App1Server@web-port-in-use (process-id-for-the-running-coupling-service)

For example, if the coupling service runs with process-id 27884 on the operating system, and uses port 8084 as its web.port, then the name is expanded to: “App1Server@8084 (27884)” at runtime.

Background Tasks Framework Setup

This functionality is available in all systems (standard, CPA, and PSO). To set up background tasks, you must complete the following:

1. Background Task nature setup
2. Background Task workspace setup

Background Task System Nature Setup

Setting up databases by type / nature allows you to control how background tasks are carried out in test and production databases. Since background tasks are copied as part of a database replication for testing purposes, if you do not take measures to control the background tasks, they will carry out as scheduled. This could lead to such things as inadvertently synchronizing data with third party systems, or sending customer emails from the test system as well as the production system. Instead, specify that a background task will only be picked up by databases of a certain nature, such as Production or Test.

To facilitate this feature, first set up the needed system natures in the configuration file, and then activate background task assignment to a system nature in Workspace Client.

Additionally, you can set up precautions for certain background tasks so that these items behave differently based on the nature to which they are assigned. Precautions exist for file location, URLs, and email accounts so, for example, that you do not inadvertently send a client an email from a test system, or sync integration data from a test system.

Setup System Nature

You must first set up database natures so that you can then assign them to background tasks.

> **Note:** The system nature names are user-defined, though we recommend using names such **test**, **dev**, and **production**.

Additionally, all recommendations in this section define the *default* settings to reference the settings you want for the test nature. You must also define a specific setting for the production nature (and possibly other natures as well).

To set up database natures:

3. In the Coupling Service, go to the configuration file **server.ini**.
4. Enter a default nature to a system shortname that defines the configuration, such as **test** or **production**.

   For example:

```
# default nature is test
system.nature=test

# the production database short name
system.nature.macoprod=production


# development databases belong to nature
# development
system.nature.dev1=development
system.nature.dev2=development
```

> **Note:** While you may have more than one system shortname database of test or dev nature, you must have only one system assigned production nature. Additionally, assigning a system as test ensures that, unless a background task is assigned a different nature, it will default be assigned the test type.

Email Account Precaution

You can control email-related background tasks by associating them with systems of a certain nature, and the system nature assignment overrides the email behavior. However, as a precaution you can also set up default email accounts associated with test systems to ensure that, if a setting is inadvertently missed, emails from a test system go to a pre-defined email account, such as an internal test account. This precaution ensures that, for example, you do not send test email to customers, as it overrides whatever email address is assigned to the background task.

To set up a default email account for a test system:

1. In the Coupling Service, go to the configuration file server.ini.
2. Find the email lines where you define email properties.
3. Enter a new email line to define that emails of a test nature go to a pre-determined email address, in the format:

   ```
   email.to.nature.test=emailaddress@domain.com
   ```

To set up a specific mail server:

Set up a specific mail server for use with a specific nature. For example, set-up a mail-server that will redirect all mails to a test account.

1. In the Coupling Service, go to the configuration file **server.ini**.
2. Find the mail server lines where you define the properties.
3. Configure the setting in the following format:

   ```
   mailserver.address=<ip-address-of-mail-server>:<port>
   mailserver.address.nature.production=<ip-address-of-production-mail-server>:<port>
   # The mailserver address specifies the default mailserver used
   # by the batch runtime (may be overridden specifically for batch)
   # as well as the "EmailOnAction" action.
   # This property can be specified globally, by system nature and/or shortname.
   # The global value can be overwritten by the system nature value which again
   # can be overwritten the the shortname value.
   # If no value is specified, the system parameter MailPortAndServer will be used
   #
   # The syntax is: serveraddress:portno
   # e.g., 172.10.8.100:25 or mailserver.trifolium.com:25
   # to use the mail server at IP-address: 12.10.8.100 at port 25
   # or the mail server at mailserver.trifolium.com at port 25 respectively.
   ```

```
#-----------------------------------------------------------
# mailserver.address = <mailserveraddress>:<port>
#
# mailserver.address.nature.<system-nature-id> = <mailserver.address>
#
# mailserver.address.<shortname> = <mailserver.address.nature.<system-nature-
id>>
#-----------------------------------------------------------
```

Third Party Integration Precaution

Set up a specific URL for test integrations so that an integration such as CRM does not inadvertently sync from a test database. This set up assumes that you have at least two integration instances, where one is used for testing and the other is used for production.

To set up an integration URL for a test system:

1. In the Coupling Service, go to the configuration file **server.ini**.
2. Find the lines where you define URL properties.
3. Enter a new line to define that an integration of a test nature syncs with a pre-defined URL set up for a test purpose.

   For example (for CRM integration):

   ```
   # by default use the test crm installation
   url.detekcrmwebservice=https://testcrm.trifolium.com/vision/visionws.asmx

   # for production, use this crm installation
   url.deltekcrmwebservice.nature.prodcution=https://livecrm.trifolium.com/vision/
   visionws.asmx
   ```

   > **Note:** The specific integration must have been prepared for reading specific url-names.

File Location Precaution

Set up file locations based on system nature if you have a set up where an external process generates file to import or where output documents produced by background tasks need to be stored in different locations depending on the system nature.

To set up a default file location for a test system:

1. In the Coupling Service, go to the configuration file **server.ini**.
2. Find the lines where you define URL properties.
3. Enter a new line to define that an integration of a test nature syncs with a pre-defined URL set up for a test purpose.

   For example:

   # the default location for “postings” file reference
   filepath.postings=c:\maconomyoutput\test\postings

   # for production, the location should be different
   filepath.postings.nature.production=c:\maconomyoutput\postings

Activate Background Task Rules

After you set up the system natures in the configuration files, you can assign individual background tasks as well as individual schedule rules to a system nature in the Workspace Client. **Background Tasks » Setup » List of Schedule Rules.**

See Background Task FAQs for answers to common questions.

Background Task Workspace Setup

By default, three schedule rules are set up. These rules can be deactivated, customized, or reactivated as needed:

- **Clean-Up Completed Background Tasks** – The system deletes all background tasks (whether completed or failed) that are more than one week old.
- **Recalculate User Notifications** – For all users that are not blocked, the system automatically recalculates notifications at 2:00 a.m. on weekdays.
- **Synchronize Employees with Revisions** –The system updates all employee information daily (after midnight). This is to capture all changes or revisions to employee information that take effect after a specified date (for example, an employee's transfer to a different department).

For the system administrator or consultant, use the various sections and tabs of the Background Tasks workspace (**Setup » Background Tasks workspace**) to do the following:

1. Set up scheduled rules to periodically perform background tasks.

   You can also set up a sequence of tasks, where the system must perform one task before performing the next. For example, submit all unsubmitted time sheets, and then approve all submitted but unapproved time sheets.
2. Monitor tasks that are going to be carried out, those that have been carried out, and those that failed (that is, those that were not successfully executed because of errors encountered). Failed tasks can be ignored, deleted, or rescheduled.
3. Manage how Maconomy logs information about successful tasks, tasks run for specific users, and so on.
4. Stop the execution of background tasks temporarily, if needed. You can also stop execution for individual coupling services.
5. If needed, create one-time background tasks.

Background Task and Schedule Rules Setup

Overview

As mentioned initially, the background task framework generally works with two main concepts:

- **Background Tasks**—A single operation that is carried out *once.*
- **Schedule Rules**—A template for the background tasks being produced, used to produce a number of specific background tasks on a *recurring* basis.

Background Tasks

Background Tasks represent a single operation that is carried out once. For example, a background task is running the action “Submit Timesheet” on a specific time sheet. Such one-off background tasks carry out the actual processes that you want the system to run. While it is possible to manually create background tasks from within the Background Tasks workspace (Editing section), this rarely happens. Instead, these background tasks are made from a Schedule Rule (see below). Also, the core (or customized) extension logic may produce individual background tasks that will help driving the business processes in a suitable manner without directly involving the end-users.

Schedule Rules

Schedule Rules function as a template for how the system can produce a *number* of *specific background tasks* on a recurring basis. Typically, a system administrator/system owner sets up a number of such schedule rules representing automated processes that are carried out at specified intervals and/or points in time.

For example, you may set-up a schedule rule that automatically submits any non-submitted time sheet on every Friday at 10 p.m. Or more precisely, the schedule rule will *generate* a concrete background task for every non-submitted time sheet every Friday at 10 p.m. In this way, the rule triggers every Friday at 10 p.m., which in turn implies that a number of specific background tasks (see above) will be created. Each of these is instructed to run the action “Submit Timesheet” and each background task will reference one particular time sheet that must be submitted. Once this set of background tasks is executed, all the time sheets that were not submitted at the time when the schedule rule was triggered, will have been submitted. Hence, the schedule rule itself does not submit time sheets, it merely ensures that individual background tasks—each submitting a specific time sheet —are produced. The scheduled and recurring triggering of the schedule rules implies generations of the relevant concrete background tasks on a recurring basis, and the background task system ensures execution of these individual background tasks.

Configure a New Schedule Rule

We can think of the schedule rules as a factory that produces concrete background tasks from a recipe.

To configure a new schedule rule, you will need to specify the recipe for how to produce the concrete background tasks. The recipe roughly consists of two things:

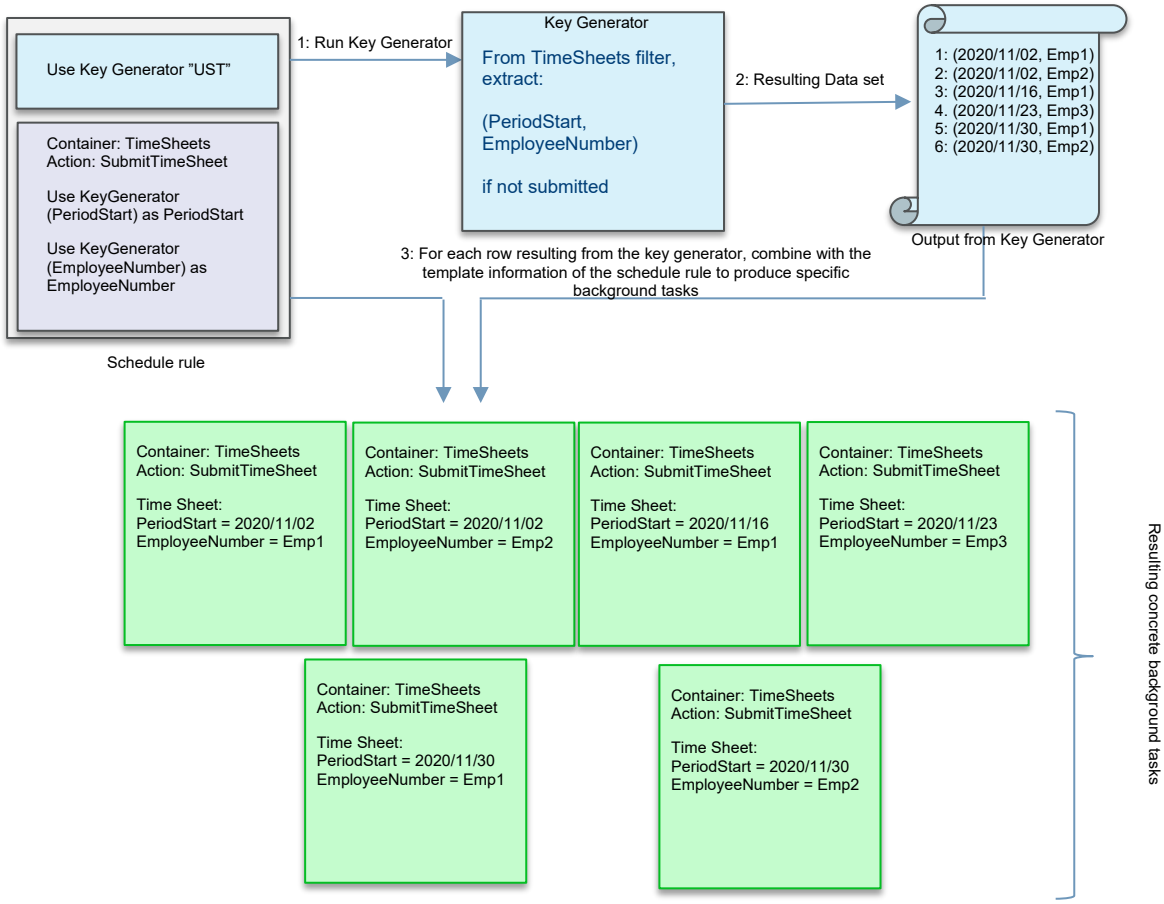
1. **A *key generator* to use.**

   You can think of the key generator as a (sub) recipe used to identify the “stuff” that we want to address. For example, identification of all the time sheets that are not submitted. The key generator extracts a specified set of fields from the database needed by a *template background task* (see below). For example, the key generator could extract the key fields of a time sheet: (EmployeeNumber and PeriodStart) for all time sheets that need to be submitted.

2. **A template of the concrete background tasks to produce**.

For example, the template could be: Run the action “Submit Timesheet” in the container “Time Sheets.” This template also defines how the information extracted by the key generator (see above) is translated into specific container-key values. This example shows how information is used to look up each individual time sheet.

Overall, a schedule rule works in the manner sketched below. In the example, the schedule rule will produce background tasks that submit non-submitted time sheets.



First, the key generator mentioned on the schedule rule runs.

In the above case, it extracts the fields PeriodStart and EmployeeNumber from the Filter pane of the TimeSheets container, where the value of the field Submitted is *False*, meaning, not submitted. This results in a set of extracted value sets. In the example, six specific time sheets were not submitted. Each of these value sets is then applied to the template information specified on the schedule rule, and a concrete background task is produced for each value set. In this case, it leads to the six concrete background tasks which all look very much alike, except they each point to a *specific and unique* time sheet. Now these background tasks can be picked up for execution by the background task system. This implies that each of these six time sheets will be submitted.

Next time the schedule rule runs, new time sheets that need to be submitted may have shown up (and the time sheets that were just submitted will no longer show up since they are now submitted). In that case, following the same recipe, concrete background tasks will be generated for *those* time sheets. Of course, if the key generator results in nothing (e.g., all time sheets were already submitted) no concrete background tasks will be produced by the schedule rule.

Key Generator Setup

The fundamental task of a key generator is to identify entities that should be in scope of a schedule rule. Examples are:

- Identify time sheets that must be submitted
- Identify customers with overdue payments
- Identify invoices that are ready for printing
- Identify journals that are ready for posting

Setup of a key generator contains these components:

- Choose filters
- Include information fields
- Indicate rows that should be in scope of the schedule rule

Filters

A key generator is always based on a *filter* pane in some container. Some filter containers have intrinsic restrictions applied. For example, the InvoiceSelection container will automatically filter away jobs that are not invoiceable. Other containers, such as the Find_JobHeader, have no intrinsic restrictions.

Generally, if you need to generate keys for use in a specific container, you should choose the same container's filter for the key generator. Exceptions may apply, depending on the specific use case. For example, if your key generator is intended for use with a schedule rule related to the InvoiceSelection container, we recommend you use the InvoiceSelection filter as the basis for the key generator. In that way, all the required intrinsic restrictions are automatically applied. If you used the filter of Find_JobHeader instead, you would need a manual configuration of the key generator to filter away (for example, jobs that are not invoiceable).

Information Fields

In addition to choosing which container to use as a filter, the key generator must know which information fields should be passed on to the schedule rule. Typically, you will be interested in the formal key fields of the database table, which is the basis for your key generator. When you enter a filter container, Maconomy automatically derives these and suggests these fields under the "Extract Fields" section.

The fields that will be passed on to (and can therefore be used by) the schedule rule are:

- **Fields listed under the "Extract Fields" group**—Here you can list any of the fields available in the filter container mentioned on this key generator. You don't *have* to extract the key fields, but usually it will make sense to do so. You can extract up to 10 different fields.
- **Fields listed under the "Define Fields" group**—Here you can define custom fields that will be made available to the schedule rule. For each define field that you give a name, you must also specify the value and the type of that field. The value may be defined by an expression, or it may be a constant value. If you enter an expression, you can make use of *any* of the fields available in the filter container of the key generator, you can make use of the standard functions of the expression language (e.g., `userDate()` which returns the current date) as well as the standard arithmetic operators such as addition (+), subtraction (-), multiplication (*) and division (/). A typical use case for a define field is to specify the user name or "role instance" of a specific user that should be used to carry out the background tasks produced by the schedule rule. Other examples could be multiplying unit price and a quantity, or merely use the define fields to

increase the number of exported fields from 10 to 15 by setting the expression to the value of a particular field.

Notice that the schedule rules can use these extracted fields for many different purposes,including identifying the data entity being targeted by the container mentioned on the schedule rules. This explains the need for exporting more than the formal key fields.

Example

To better understand how the define fields work, let us use an example:

Suppose our key generator extracts the fields: JobNumber and CustomerNumber. In addition, suppose we have defined the following custom fields:

- `RunAsUser [constant, String] = SystemUser2`
- `OpenMarkupPct [expression, Real] = if OpenCostpriceEnterprise = 0 then 0.0 else (OpenBillingPriceEnterprise – OpenCostpriceEnterprise) / OpenCostPriceEnterprise * 100.0`

Hence, the first, `RunAsUser`, is merely a constant value of type String: “SystemUser2”

The second, `OpenMarkupPct`, is a field of type Real defined as an expression. Essentially, it defines the value as the ratio between the open mark-up and the open cost price (multiplied by 100%). In case the divisor is 0, the value is set to 0.

Assume that the rows that are picked up through the filter container of the key generator are:

JobNumber	CustomerNumber	OpenBillingPriceEnterprise	OpenCostPriceEnterprise
10251002	770-112-228	200.00	100.00
10251070	663-991-656	0.00	0.00
10252031	775-982-152	350.00	140.00

The rows generated by the key generator would be the following:

JobNumber	CustomerNumber	RunAsUser	OpenMarkupPct
10251002	770-112-228	SystemUser2	100.00
10251070	663-991-656	SystemUser2	0.00
10252031	775-982-152	SystemUser2	150.00

This was made by extracting the “extract fields” for each row obtained by the key generator and by computing the field OpenMarkupPct based on the values of the fields of the corresponding row. So, for the first line, the value for OpenMarkupPct is computed as:

```
if 100.00 = 0 then 0.0 else (200.00 – 100.00) / 100.00 * 100.0
```
which evaluates to
```
(200.00 – 100.00) / 100.00 * 100.0
```
which evaluates to
```
100.00 / 100.00 * 100.0
```
which evaluates to
```
1.0 * 100.0
```

which evaluates to
`100.0`

The custom field RunAsUser has the same constant value (SystemUser2) on all rows.

A schedule rule based on this key generator can now use the fields: JobNumber, CustomerNumber, RunAsUser and OpenMarkupPct.

Filter Rows

The final thing you need to take into consideration when defining a key generator is which of the rows available in the filter container should translate to a background task.

For example, if your schedule rule is supposed to submit time sheets, it would be inefficient to create a background task for *every* time sheet: we only need to identify the time sheets that are not already submitted. And we probably don't want to consider time sheets with a closing date in the future.

For this reason, a key generator is associated with an expression determining the rows to let through. This condition is specified in the field **Extracted Rows Match this Condition**. This expression is defined using the same kind of expressions as the ones used for defining custom fields. There are, however, some limitations on the use of functions, including that you are not able to apply standard functions to the field values in the database. You can, however, use functions to compute values that you compare to the database fields.

Example

Following the example, to only include time sheets that are not submitted, you would write something like:

```
not Submitted
```

This would, however, not take into account that we only want time sheets from the *past*. To handle that, we need to expand the condition by stating that the value of the field PeriodEnd must be in the past (i.e., *before* today's date). Today's date can be obtained through the function userDate. Our expression would then look like:

```
not Submitted
and PeriodEnd < userDate()
```

If, additionally, we would want to only consider "recent" time sheets (for example, time sheets covering a period no earlier than 60 days prior to today's date), we would again make use of functions to compute the date as it was 60 days earlier and require that the field PeriodStart must be at least that date. We can combine the functions addDays and userDate to compute this. Our condition would become:

```
not Submitted
and PeriodEnd < userDate()
and PeriodStart >= addDays(userDate(), -60)
```

As an example, let us have a look at how the above expression would apply to a small sample data set. Suppose the database contains the following time sheets:

EmployeeNumber	PeriodStart	PeriodEnd	Submitted
Emp1	2020/11/02	2020/11/08	No
Emp2	2020/11/02	2020/11/08	Yes
Emp1	2020/11/16	2020/11/22	No
Emp3	2020/11/23	2020/11/29	No

EmployeeNumber	PeriodStart	PeriodEnd	Submitted
Emp1	2020/11/30	2020/12/06	No
Emp2	2020/11/30	2020/12/06	No

Now, depending on the *date* the key generator/schedule rule runs, the resulting rows will differ. The only row that would *never* be included is the second row, because this time sheet is already marked as submitted.

Imagine that the key generator runs on November 28, 2020: Then the following rows would be extracted:

EmployeeNumber	PeriodStart	PeriodEnd	Submitted
Emp1	2020/11/02	2020/11/08	No
Emp1	2020/11/16	2020/11/22	No

The reason is that the rows where the PeriodEnd has a value that is after November 28, 2020, will *not* be considered, since the expression would be false for those rows.

Similarly, suppose the query was run on January 2, 2021. In that case, the result would be:

EmployeeNumber	PeriodStart	PeriodEnd	Submitted
Emp1	2020/11/16	2020/11/22	No
Emp3	2020/11/23	2020/11/29	No
Emp1	2020/11/30	2020/12/06	No
Emp2	2020/11/30	2020/12/06	No

Now, the oldest time sheet (dating from November 02, 2020) falls beyond the 60 day rule and will not be included. The remaining ones are all in the past and will be included.

In general, you will benefit from narrowing the number of rows generated by the key generator. In this way, the number of background tasks produced by the corresponding schedule rule is smaller and more specific, reducing the amount of work needed to be done by the Background Task system.

Tip

You can make a test-run of your key generator by running the **Test Key Generation** action. This produces an Excel spreadsheet with the data to be extracted by the key generator if it was run now. Since it may be difficult to get a sense of whether the data seems reasonable, you may want to include additional fields in the test run. You can add up to 10 additional fields that will be included in the test run output under the header “Show Fields During Test.” For example, sometimes you might want to get an employee number, a customer name or some of the fields included in the filtering conditions, such as Submitted. Run a test to ensure that you didn’t make an error or typo in your expression (such as writing “or” instead of “and” or “>” instead of “<”.

Schedule Rule Setup

A schedule rule defines a template background task and describes how the fields extracted from the key generator are used to populate the template. In addition, it also specifies *when* the schedule rule should trigger.

First, let us consider the template information.

The schedule rule must specify:

- The **container** in which an operation should be performed—This information is fixed and cannot vary. An example could be “TimeSheets”
- The **pane** in scope for the operation (e.g., card or table) —This information is fixed an cannot vary (an example could be “Card”)
- The **action** to run—This information is fixed and cannot vary. An example could be “SubmitTimeSheet” or “Update”
- How to identify the **container key** (e.g., a specific job, a specific time sheet, a specific customer) —This information can vary depending on the result from the key generator
- A possible **guard expression**—This information can vary depending on the result from the key generator
- How to handle possible **output documents**—This information can vary depending on the result from the key generator
- How to handle possible **input document** requests—This information can vary depending on the result from the key generator
- What **data fields** to modify as part of performing the operation (“wizard fields” for actions or field values to update or set during creation) —This information can vary depending on the result from the key generator
- Whether the action needs to be **parameterized** (may be used with Java extensions) and how—This information can vary depending on the result from the key generator

All this information is found on one-off background tasks background tasks as well. The difference here is that for many of these values, we can refer to the information from the *key generator*. In order to avoid confusion with fields of the container related to the background tasks being produced, the information from the key generator is referred to by the prefix “`context.`” So, if the key generator extracts a field called EmployeeNumber, you refer to that field as `context.EmployeeNumber` in the schedule rule.

Each of the fields provided by the key generator are displayed in the schedule rule, and for each you can choose how/if the corresponding value should be used to map into a key field on the concrete background tasks produced. Typically, the mapping is trivial, and includes a straight-forward mapping of a context field to the corresponding field in the container (without the context prefix).

Example

As an example, suppose our schedule rule references a key generator that extracts the fields EmployeeNumber and PeriodStart from time sheets. Now, how do we map these fields into a specific time sheet?

```
context.EmployeeNumber -> EmployeeNumber
context.PeriodStart    -> PeriodStart
```

That is, for each row generated by the key generator: produce a background task where the keyfield EmployeeNumber (right-hand side) gets the value from `context.EmployeeNumber` (the specific value extracted as “EmployeeNumber” in the key generator). And similar for the key field PeriodStart.

The field mapping is not necessarily trivial, though. For example, suppose the key generator extracts the field ProjectManagerNumber from jobs and that you want to use this number to identify an employee in the Employee container on the schedule rule. The mapping would then be:

```
context.ProjectManagerNumber -> EmployeeNumber
```

In general, one background task is produced for every row extracted by the key generator. However, if the schedule rule detects that duplicate background tasks will be generated, it will only produce one of these. For example, if you want to run some action of every employee who is acting as a project manager on some job, your key generator would extract the field ProjectManagerNumber from the Jobs filter. Since a given employee could be managing many projects, this gives the potential for multiple background tasks. Since the schedule rule automatically detects this, only one background task would be generated for each employee.

Context Fields

For each context field, you can choose whether its value should be used to identify which user(-role) the produced background task should be run as. You can choose that for *one* of the context fields by specifying that the value should translate into a user(-role). The possible options are:

- **Employee**—The concrete value is interpreted as an employee number. The user to run-as will be a user which is associated with this employee. If several user profiles are associated with the employee one of them will be used. A non-administrator profile will be chosen over the others, though. The background task will run as the user-role which is specified as the default background task role of the chosen user.
- **User**—The concrete value is interpreted as a user name. The background task will run as the user-role which is specified as the default background task role of that user.
- **Role-Instance**—The concrete value is interpreted as a user-role instance key. The background task will run as the user-role having the specified instance key value.

If none of the context fields are marked with a “Run-as” option, the background tasks produced by this schedule rule will run as the background task administrator user specified for your system.

The field “Record Condition” is used to specify a “guard” condition. For actions running in a table pane where multiple records may be found, it can also be used to identify a specific (or a few) lines that are affected by the operation. The produced background task will contain this Record Condition (expanded with the context-values from the key generator). When the background task runs, it will perform the specified operation on all rows where the resulting expression evaluates to true. In card panes there is only one row, so a Record Condition on a card pane will determine whether to *really* apply the operation. This is very useful in several cases.

Example

For example, assume that our schedule rule produces background tasks that submits time sheets. Assume that the key generator extracts only time sheets that are not submitted. When the schedule rule triggers, it will produce a number of background tasks, one for each time sheet. Later (maybe only a short while later) the concrete background tasks will be picked up for execution. But what happens if some user manually submitted one of these time sheets in between? If a time sheet has already been submitted, an error “to the end user” would occur when the time sheet is attempted submitted. Instead of ending up with

background tasks that report a failure because it is no longer possible to submit a given time sheet, it is better to instruct the background task to simply ignore running the action if a certain condition holds. For example, we could specify the record condition:

```
not Submitted
```

When the generated background task is run, this condition is evaluated immediately before a SubmitTimesheet is attempted. If the time sheet *is* already submitted at that time, the above condition would evaluate to false, and it will not run. The exact condition can be made dependent on values from the key generator by referring to the `context.`*fieldname* in that expression.

For table-panes, the record condition can be used to choose for which (if any) of the lines the action should be performed. For example, suppose we want our schedule rule to delete all time sheet lines with 0 hours. The schedule rule would specify the action to be Delete on the TimeSheet (table). If we didn't specify any record condition, all lines would be deleted! Since that is not what we want, we would specify as record condition:

```
WeekTotal = 0
```

With that, only lines where the WeekTotal is 0 will be deleted.

If an action specified by ta background task produces a document/file as a side-effect, the background task system should know what to do with the document. Normally, and end-user would receive a document and decide what to do about it. But there are no end-users involved with background tasks. You can therefor specify what to do about such possible documents:

- Explicitly don't care and let the document be thrown away
- Choose a file-server location where the output document is stored
- Choose an e-mail recipient for the documents; an email with the documents attached will be sent to that recipient if the action resulted in documents
- Choose a document archive to store the output documents

You can specify several of these options depending on the file pattern. For example, you can store PDF-files in one fileserver location, and send Excel spreadsheets to a specified e-mail address.

For the e-mail and document archive options, you can specify these as expressions. And the expressions can reference to the values of the key generator by referencing the `context.`*fieldname* values.

For example, suppose the key generator extracts a field called ElectronicMailAddress, and assume that this contains the email-recipient that we would want to have PDF-documents sent to.

We can then specify a file-output handler (using e-mail) as:

```
Handle Files Like: *.pdf
E-mail To        : if context.ElectronicMailAddress = "" then
"maconomyowner@yourcompany.com" else context.ElectronicMailAddress
```

Hence: if the email-address that we get from the key generator is empty we choose some static internal email address. Otherwise, we use the value extracted by the key generator. It is also possible to just specify a static e-mail address. Maconomy will automatically figure out if the value is an expression or a static e-mail address.

Similarly, you can specify output handlers that makes use of document archives as well as handlers for input files, specifying a possible document archive from which o fetch requested input documents.

Adding User Input Lines

Sometimes you need to enter information when running an action. For example, the "Copy Budget" action in the JobBudgets container lets you choose the jobnumber, budget type and revision number of the

budget you want to copy into the targeted budget. Other times, you want the generated background tasks to make an update (i.e., changing some field values) or you want to create a new instance (e.g., a new time sheet or a new time sheet line). In such cases, you need to tell the system what values should be entered by the background tasks. This is done by adding lines of type “User Input” in the table underneath the schedule rules. For each field you want to change/set, you must specify:

- The name of the field
- The value of the field, either as a constant value or as an expression. The expressions can refer to the values of the key generator by prefixing the field with `context.`*fieldname*.

For example, suppose you want a schedule rule to generate background tasks for every non-submitted time sheets where there are days with too few hours. In that case, you want the background task to create a new time sheet line on a special “Missing Time Sheet” job and task, and enter the missing number of hours on the Monday.

Assuming that the key generator has extracted the missing hours as a field, MissingHours, we can make a schedule rule that performs a “Create” in the table. The following User Input would then be specified:

```
FieldName: JobNumber      Value: J-MissingTime          (constant)
FieldName: TaskName       Value: MissingTime            (constant)
FieldName: NumberDay1     Value: context.MissingHours   (expression)
```

In a similar way, you can specify parameters (possibly used by extensions) to the background task action. The only difference is that the line type will be “Parameter” instead of “User Input”.

The idea behind schedule rules is that they are configured to run at scheduled and recurring intervals. This is triggered through the “Trigger Time Pattern”. It has the format of a UNIX cron pattern. Basically, such a pattern consists of five parts:

- **Minute pattern**. For example “5” to mean “five minutes past”, “/5” meaning every fifth minute. The “catch all” pattern “*” is strongly discouraged: that would make the rule run every minute when all other patterns match.
- **Hour pattern**. For example “8” meaning when the hour is 8am. “20” meaning when the hour is 8pm, “8-17” meaning all hours from 8am to 5pm. The catch all pattern “*” means every hour.
- **Days of month pattern**. For example “1” meaning the first day of the month, “15” meaning the fiftheenth day of the month. “L” means the last day of the month, independent of the number of days in the given month. The catch all pattern “*” means every day of the month.
- **Month pattern**. “1” or “jan” meaning January, “4-6” meaning april, may and june. The catch all pattern “*” means every month.
- **Days of week pattern**. For example “0” or “sun” means Sunday, “3-5” or “wed-fri” means Wednesday, Thursday and Friday, “2,6” means Tuesday and Saturday. The catch all pattern “*” means all days of the week.

These patterns are used in combination to specify time intervals. Examples are:

- /15 8-17 * * mon-fri means: every fifteenth minute from 8am to 5pm, but only from Monday to Friday,
- 0 22 * * * means: at 10pm every day
- * 22 * * * means: every minute from 10:00pm to 10:59pm. **Not recommended! You probably meant 0 22 * * ***
- 0 22 L * * means: at 10pm on the last day of each month

In addition, you can combine several patterns by separating them by the pipe (“|”) character.

Sometimes you only want the schedule rule to run from a certain date, or in a specific date interval. You can do that by specifying a validity date interval on the schedule rule. For example, specifying the interval as 2021/06/01 – 2025/12/31 means that, no matter the pattern, the schedule rule will only be considered from June 1, 2021 until December 31, 2025.

Other times you want a rule to run a few times, maybe only once. You can obtain that by specifying that the schedule rule should only run a maximum number of times. Once you do that, you can enter the number of runs left. Every time the schedule rule runs, the number will decrease. A value of 0 means that it will not run again. If you wish, you can always edit this number to increase or decrease it.

Chained Tasks

By default, the concrete background tasks produced by a schedule rule will be made in a way that allows all of them to be executed in parallel. The actual limit on how many can be executed in parallel depends on the number of coupling services and how many background task execution threads they each have. Sometimes the background tasks will need access to shared resources such as a specific database record. It could by a system number series or a specific customer. This means that even if the concrete tasks are executed in parallel, they will really just be forced into being executed in sequence: for example, three such concreate background tasks may all need access to a shared resource. One of them gets access first, and the other two will be held back by a database lock (but this wait will still be done by one of the available background task execution threads). Eventually, the first one is done, and the database lock is released. Then one of the two others will get the lock, and can continue. The last one is still awaiting that the second one finishes and releases the lock. And the, at last, the third one can continue. This is obviously a bad use of the background task execution threads that might as well have executed other tasks that would not need to wait for the first. For this reason, it is possible to set the flag “Chained Tasks” on the schedule rule. When this flag is set, the concrete background tasks will be created in a way that enforces the background task system to only pick up one of them at a time. You can also have the schedule rule produce a *number* of such sequences, depending on some value from the key generator. For example, you could make the chaining group on `context.CompanyNumber` to make one such “chain” of concrete background tasks for each associated value of the “CompanyNumber” field extracted by the key generator.

Low Priority Tasks

Finally, you can specify that the background tasks produced by the schedule rule should be marked as having “low priority”. Doing so implies that – with a certain probability – other background tasks that are marked with normal priority may be picked up for execution even though its due date/time is later than that of a low-priority background task. Low-priority background tasks will eventually be executed, but they will not completely block other normal-priority background tasks that may have been scheduled to run at a slightly later date/time.

Different Kinds of Schedule Rules

The above settings are those that are used to produce background tasks from a schedule rule. The type of such schedule rules are called “Single Schedule Rule”. There are other kinds of schedule rules:

- **Single Schedule Rules** (as described above)
- **Sub Schedule Rules**—These are like the Single Schedule Rules, but they cannot be scheduled to run at a certain time pattern. Sub rules are used together with Sequential Schedule Rules (see below)
- **Sequential Schedule Rules**—Such rules can be scheduled with a time trigger pattern, but they do not themselves produce background tasks. Instead, when such a rule triggers, it will run a number of associated *sub schedule rules* (see above) in sequence: For sequential schedule

rules, you can enter an ordered list of sub schedule rules. This means that the first sub schedule rule will be run when the sequential schedule rule triggers. Once all the concrete background tasks of the first sub schedule rule have all been executed, the next sub schedule rule will be triggered. And so on.

- **Interleaved Schedule Rules**—This kind of schedule rule is deprecated and will be removed. This kind of schedule rule should no longer be used; use the sequential schedule rule instead.

As an example of a sequential schedule rule, suppose we want to automatically submit all time sheets that needs to be submitted, and – once that has been done – we want to approve all time sheets that have been submitted, but not approved.

This can be achieved by making a sequential schedule rule, which in turn lists two sub schedule rules:

1. The first sub schedule rule is configured (as explained earlier) to submit time sheets.
2. The second sub schedule rule is configured (in a similar way) to approve time sheets.

There are no requirement that the sub schedule rules must relate to the same container. It is perfectly possible to have one sub rule relate to invoicing and a subsequent one relate to posting journals. The only thing is that the sub rules are carried out in order, and each one is triggered immediately after the last background task of the preceding sub rule has been carried out. It is much better to schedule dependent schedule rules in this way than choosing “wide apart” schedules; doing so could lead to either too much time in between the schedule rules – or in maybe too little time in between.

Using Expressions in Input Fields

E-mail (and other) fields used for document handling on rules and tasks handle as expression fields as well as e-mail addresses. Here, you can use data from the context fields to complete these fields.

For the fields **E-mail From** and **E-mail To**, the system can automatically determine if the field contains a valid e-mail address expression. However, for the Document Archive number and subject / body fields in the e-mail, you must explicitly specify if it is an expression or not (using the **Yes/No** expression field).

Background Shading

Background shading offers at-a-glance information for a field type. Expressions are indicated via purple background color. Without color on the field, it is an email.

This convention is used in these areas:

- Input Document Handling (Using Doc. Archives)
- Output Document Handling (Using E-mail)
- Output Document Handling (Using Doc. Archive
- Send Email on Completion (fields with **Expr.** column/field to the right of them).

To use expressions in Output Document Handling / Email:

1. Go to **Background Tasks > Management > List of Scheduled Rules**.
2. In the **E-mail To** field, select an expression from the Context Field Name field.
3. If this is a Document Archive number field, or subject / body field in the email notifications, then you must set the **Expr.** value to **Yes** to explicitly say that this is an expression.

Using Extended Test Bed

The Test Bed functionality enables you to test your rule without actually activating it so that you can see in action how it works and if it produces a result that you expect. For ease of use and versatility, the Test Bed includes color-coded format for at-a-glance status, as well as key actions and a wizard. Additionally, now you can test the rules before the rule is activated.

Features include:

- **Origin** field on a task defines the rule from which it generated. This field is also used when checking if any pending task from a rule exists. In this column, you can select a new value called Test Bed, so that you can filter by this type or by Manual, Schedule Rule, API, or Core Application
- If needed, generate tasks from a disabled rule.
- New Schedule Rule column in the Generated Background Tasks sub-tab enables you to see where the background task started.
- The Generated Background Tasks sub-tab shows you the tasks that were generated when the rule was executed.
- You can test a rule before you activate it.

To test background tasks in the Test Bed:

1. Go to **Background Tasks > Setup > List of Scheduled Rules > Test Bed**.
2. Click **Test Schedule Rule.** A wizard displays.
3. In the **Max No. of Test Tasks** field, enter the number of tests you would like to generate. This is helpful if you have thousands of tasks, but for example you would like to only test 50 or so to

ensure that the rule is working correctly. Specify 0 (zero) for no limit / to generate all in the database.

4. Select **Activate Generated Test Tasks** to specify whether tasks must be activated. If you do not want the tasks activated, deselect this field.
5. Section **Generate Test Tasks in the Background** to continue to have access to the UI and related functionality while background tasks run.

Test Scheduled Rule Time Pattern

In the Test Bed assistant, used for testing scheduled rules, a new **Show Next Scheduled Rule** action shows the next fifty (50) execution times for the current rule. This action is beneficial for debugging, and is used to verify that the time pattern entered for the rule results in the wanted executions. For ease of use, you can also run / test the rule prior to activating the rule.

Note: Take into consideration the data in the **No. of Runs Left** field and the valid interval when evaluating execution times.

The new functionality also enables you to test scheduled rule time patterns before you activate them.

In the Test Bed area, a new **Test Schedule Rule** action triggers a wizard which enables you to test schedule rules.

Use this action to verify that the time pattern entered for the rule resolves in the needed executions.

Note the 'No. of Runs Left" and the valid interval should also be taken into consideration when execution times are resolved.

Low Priority Tasks

There are times where it makes sense to run tasks with a lower priority than usual. A good example of this is a notification calculation, where a lot of time is spent on non-critical tasks, which are generated to execute at the same time. This load can block all Background Task resources for hours and prohibit other more critical tasks from being executed.

This is resolved by supporting tasks of two priorities (standard and lower).

In the default configuration, standard priority tasks are picked approximately twice as often by a polling thread as low priority tasks when both standard and low priority tasks exists and are due – even though the due time of the low priority tasks are clockwise before the standard priority tasks.

The priority ratio between standard and low priority tasks (default: 70%) can be configured, but cannot be raised over 90%. This is done to avoid starvation of low priority tasks. On the other hand, setting a value of 0% will make the Maconomy ignore the setting of low/standard on tasks/rules and thereby make the system as before this feature was introduced.

For at-a-glance use, low priority tasks have a light blue foreground color status panes.

To set tasks as low priority:

1. Go to **Background Tasks > Setup > List of Schedule Rules > Schedule Rule**.
2. In the Background Task area, select the **Low Priority** check box.

To set tasks as low priority:

3. Go to **Background Tasks > Setup > List of Schedule Rules > Schedule Rule**.

4. In the Background Task area, select the **Low Priority** check box.

Hold Tasks Rule If Tasks Are Still Pending

You can hold tasks even if tasks from the previous generation are still waiting to execute. A popup field indicates if a rule should run now, or later.

To create rule to hold tasks:

1. Go to **Background Tasks > Schedule Rules**.
2. In the **Name** field, select the name of the Schedule Rule to update.
3. In the **When Last Run Not Complete** field, choose:
   - **Run Anyway** — If you choose this option then when the rule is triggered, it will run regardless if there are tasks that have not yet finished from the previous run of this rule.
   - **Skip** — Select this option to skip the current run of the rule if there exist tasks that have not yet finished from the previous run of this rule. Additionally, the Next Due Date and Time are updated as if this rule ran successfully. For example, if the rule is set up to run every minute and the next run is at 3:01 p.m., but there are tasks from previous runs that have not yet finished, then the rule is skipped and the Next Due Time is updated to 3:02 p.m.
   - **Postpone** — Similar to Skip, except the Next Due Time is NOT updated. If we take the example above then the rule will be skipped, but the Next Due Time will NOT be updated and will stay as 3:01 PM. Select this option to skip if there are existing tasks from the previous run, but you want to make sure the next due time is not updated, so that the rule can be run as soon as possible.

Notes:

1. The time in the **Next Due** field is not updated when a rule is rejected due to existing tasks.
2. The **Schedule Rule** field tells the name of the rule from which each task originates. Select to show Tasks with the reference to the rule generating them. This enables you to easily check if un-executed tasks still exist. If the task was created manually (and not from the rule) this field is empty
3. Do-not-generate is default for new rules created.

Requests for Pending Tasks

We have reduced the number of request for pending tasks from N times a minute (where N is the number of background task threads) to once a minute. This reduces the load on the system and makes it possible to configure a large number of background task threads without wasting resources in idle situations.

The following items improve efficiency:

- A queue holds all pending threads. When the poll interval is up, the queue releases a thread (X), which will ask for a pending task.
  - If no pending tasks exists, the thread X returns to the queue and new thread is released when a new poll period is up.

- If, on the other hand, a task (A) is pending, the thread (X) signals to the queue that there may be more pending tasks before starting to execute the task (A). This signal leads the queue to release another thread (Y), which might pick-up another pending tasks (B) – and release yet another thread (Z) from the queue, before execution of the tasks (B) begins.

  In this way, threads picking up tasks which lead to new active threads keep the process running until no more tasks are pending and thereby the threads queue up again.

- If no threads are in the queue when a release signal is received, the queue counts needed threads and a new thread is released as soon as it enters the queue. This process repeats as long as the needed thread counter is larger than zero (0).

- There is a thread queue-instance for each short-name in a Coupling Service as short-name are handled independently in relation to the batch framework threads.

Retry Scheduling

For efficiency in retry scheduling, and to avoid tasks scheduled at the same time, the default number of retries is five (5) and the amount of time between retries is randomized.

The randomized time-pattern is calculated in the following way. There is a parameter defining the rescheduling delay (default 30 seconds). A task is rescheduled a number of delays plus a random amount of time, giving a result in the interval between the current rescheduling and the (possible) next rescheduling.

For example:

Rescheduling Count	**1**	**2**	**3**	**4**	**5**	**6, 7, ...**
Delay Period	1	3	5	7	9	9
Randomized Delay %	100 - 299	300 - 499	500 - 699	700 - 899	900 - 1099	900 -1099

When a task is rescheduled, all existing log-lines are removed from the task and a single line states that the task is rescheduled for the [number] time.

Document Archives

A Document Archive is created automatically by the system when you execute a task, if needed, for ease of use.

On tasks and rules, callback-handlers using document archive automatically create the document archive if it does not exist. This is similar to the functionality when using folders in the file system, as these are also created if they do not exist.

Chained Background Tasks

At times it is necessary to specify that a number of background tasks (defined by some properties) should be executed such that only one task from the group is executed at any time. The **Chained Task Execution** field and a grouping field are introduced on rules.

If the grouping field is assigned with field name from the rule context, tasks generated by the rule are grouped in a number of chains, where all the tasks in one chain have the same value for the context field.

For example, one group-field is CompanyNumber. If CompanyNumber is defined as the group field, all tasks with the same context CompanyNumber generated from the current rule are chained and executed in a sequence, in such a way that all the tasks (for a given company) are executed alone, (meaning, without any other task from the same company being executed at the same time). If the grouping field is not assigned (but the chained execution is enabled), all tasks generated by the rule are linked in one chain and all these tasks are executed without overlap.

The **Recalculate User Notifications** rule supports this feature. If a user has multiple user roles, then notifications for those roles are recalculated in a chain / succession.

Expressions in Background Tasks

Some of the previous chapters included several examples of using expressions in various fields in background task related containers. It should be clear by now that the difference between a fixed value and an expression is that a fixed value is essentially a value that is taken “as is” without the need to evaluate it. On the other hand, an expression is something that may be evaluated at some point to determine its final value.

The advantage of using expressions over fixed values is that they allow to encode more complex workflows. I.e. expressions allow to produce background tasks that depend on the underlying context they run in. For example, instead of always sending an email to a fixed email address, you might want to send it to the underlying employee that the background task executes on.

Generally speaking, we could categorize supported expressions into 4 types:

1) Container Field References
2) Context Field References
3) Standard Functions
4) Combined Expressions

Container Field References

The simplest supported expression is a container field reference. Whenever we create a new background task, we must always specify an underlying container the task should execute in. Once we have done that, we will be able to reference all the underlying fields that exist in that container, including any custom fields and variables. This is particularly useful when trying to filter out records based on some condition or when trying to update a field based on another field in the same container.

As an example, let us assume that we want to create a single background task that will submit all the time sheets that have not been submitted yet and belongs to the employee with number ‘11’. In this example we will have to reference at least 2 fields on the time sheets: the “Submitted” field and the “EmployeeNumber” field. An expression like that could look as follows in our “Record Condition” field:

```
not Submitted and EmployeeNumber = '11'
```

When such background task runs, each timesheet is evaluated against this condition. First, the “Submitted” and “EmployeeNumber” values are substituted with the values from the current timesheet and then the expression is evaluated. If the expression evaluates to true, the timesheet is submitted and otherwise it is not, i.e. the timesheet is skipped and no action is performed on it.

As another example, let us say that we (for some valid reason) would like to transfer the value from “EmployeeNumber” field to “Remark3” field and the value from “Name1” field to “Remark4” field in our Employees container. As shown in the table below, setting this up is fairly straightforward. The key thing to remember here is that you must explicitly mark the “Expr.” field to indicate that the specified value should be interpreted as an expression and not as a fixed value. If we had not marked the expression field then the “Remark3” and “Remark4” fields would have simply been set to textual values “EmployeeNumber” and “Name1” respectively, and not evaluated against the underlying employee in order to extract the values from these fields.

Line Type	Field/Parameter Name	Value	Expr.
User Input	Remark3	EmployeeNumber	Yes
User Input	Remark4	Name1	Yes

In general, most fields that support expressions allow to reference container fields. As shown in the first example you may reference multiple container fields in an expression or a single container field in an expression as shown in the second example.

Context Field References

Context fields are fields that are extracted from “Key Generators” and made available on “Schedule Rules”. As it was mentioned in previous chapters, key generators can extract fields from the chosen container. Moreover, it allows you to define your own five custom fields with your chosen name, field type, and value. Both, extracted fields and defined fields can be referenced in expressions on your schedule rule. Context fields can be recognized easily as they must always be prefixed with the “context.” prefix.

As an example, let us assume that our key generator extracts a list of employees, including their name and email address, that are missing to submit their timesheets. The schedule rule then creates a background task for each employee that will send an email, which will remind the employee to submit their time sheets.

To achieve this, we could utilize the “Send Email on Completion” functionality on our schedule rule, since all of the fields support expressions. It would look as follows:

Field	Value	Expr.
To	context.ElectronicMailAddress	Yes
Subject	Please submit your timesheet	No
Body	'Dear ' + context.EmployeeName + ' you are missing to submit your timesheet.'	Yes

As you can see the fields “To” and “Body” are marked as expressions, meaning that any references to the “context.” fields will be evaluated and substituted with the value of the underlying employee.

An important difference between container field references and context fields references is that container field references are only evaluated when the background task is executed, while the context field references are evaluated when the schedule rule runs. In this particular example it means that in the tasks created from this schedule rule the references to “context.ElectronicMailAddress” and “context.EmployeeName” will already be substituted with the correct values.

Standard Functions

Standard functions are a collection of useful Maconomy built-in functions that can be used in the fields that support expressions in key generators, schedule rules, and background task containers.

As an example, let us assume that we want to delete all the employees that are blocked and have not been changed for 2 or more years. One way to achieve this is to compute the difference between the changed date on employee and the current date. If we assume that our background task is set up to run the “Delete” action in the Employees container, then our “Record Condition” might look as follows:

```
Blocked and yearsBetween(ChangedDate, currentDate()) >= 2
```

“Blocked” and “ChangedDate” are field references in the underlying Employees container. yearsBetween() is standard function that returns the number of years between the two given dates, while currentDate() is a standard function that returns today’s date.

Maconomy supports a lot of standard functions for working with dates, times, strings, numeric types (i.e. amounts, reals, integer), and much more. A comprehensive list of all the available standard functions with their descriptions and examples of how to use them is provided in the “Supported Standard Functions” chapter.

Combined Expressions

Combined Expressions in itself is not a special type of expression category, but rather an attempt to clarify that all of the previously mentioned expression types (i.e. container field references, context field references, and standard functions) can be combined and used in a single expression in certain fields. As a rule of thumb, if a field supports referencing container fields and/or context fields, it is very likely that it also supports standard functions. We have already seen an example of this in the previous chapter, where we referenced both, container fields and standard functions, in our record condition.

In fact, in the Schedule Rules container, certain fields even support referencing container fields, context fields, and standard functions in the same expression! For example, perhaps one day you might need to specify the following restriction on your Schedule Rule:

```
not isAdministrator() and not contains(uppercase(NameOfUser), 'PATRICK') and
yearsBetween(ChangedDate, currentDate()) >= context.MaxInactivityYears
```

The parts highlighted with red color are standard functions; the green parts are container field references; and the blue part is a context field reference.

Expressions Supported in Different Containers

The tables below describe which fields in different containers support expressions and what type of expressions.

Key Generators

		Supported Expressions		
Internal Field Name	External Field Name	Standard Functions	Container Field References	Combined Expressions
SemanticFieldName1	Extract Field Name 1	✗	✓	✗
SemanticFieldName2	Extract Field Name 2	✗	✓	✗
SemanticFieldName3	Extract Field Name 3	✗	✓	✗
SemanticFieldName4	Extract Field Name 4	✗	✓	✗
SemanticFieldName5	Extract Field Name 5	✗	✓	✗
SemanticFieldName6	Extract Field Name 6	✗	✓	✗

		Supported Expressions		
SemanticFieldName7	Extract Field Name 7	✗	✓	✗
SemanticFieldName8	Extract Field Name 8	✗	✓	✗
SemanticFieldName9	Extract Field Name 9	✗	✓	✗
SemanticFieldName10	Extract Field Name 10	✗	✓	✗
DefinedFieldValue1Var	Defined Field Value 1	✓*	✓*	✓*
DefinedFieldValue2Var	Defined Field Value 2	✓*	✓*	✓*
DefinedFieldValue3Var	Defined Field Value 3	✓*	✓*	✓*
DefinedFieldValue4Var	Defined Field Value 4	✓*	✓*	✓*
DefinedFieldValue5Var	Defined Field Value 5	✓*	✓*	✓*
InfoSourceField1Name	Information Field 1	✗	✓	✗
InfoSourceField2Name	Information Field 2	✗	✓	✗
InfoSourceField3Name	Information Field 3	✗	✓	✗
InfoSourceField4Name	Information Field 4	✗	✓	✗
InfoSourceField5Name	Information Field 5	✗	✓	✗
InfoSourceField6Name	Information Field 6	✗	✓	✗
InfoSourceField7Name	Information Field 7	✗	✓	✗
InfoSourceField8Name	Information Field 8	✗	✓	✗
InfoSourceField9Name	Information Field 9	✗	✓	✗

		Supported Expressions		
InfoSourceField10Name	Information Field 10	✗	✓	✗
Restriction	Extracted Rows Match This Condition	✓	✓	✓

* Values entered in these fields will only be interpreted as expressions when the "Expr." field is set to Yes.

Schedule Rules

		Supported Expressions			
Internal Field Name	External Field Name	Standard Functions	Context Field References	Container Field References	Combined Expressions
LinkToContainerKeyField1Var	Link To Container Key Field 1	✗	✗	✓	✗
LinkToContainerKeyField2Var	Link To Container Key Field 2	✗	✗	✓	✗
LinkToContainerKeyField3Var	Link To Container Key Field 3	✗	✗	✓	✗
LinkToContainerKeyField4Var	Link To Container Key Field 4	✗	✗	✓	✗
LinkToContainerKeyField5Var	Link To Container Key Field 5	✗	✗	✓	✗
LinkToContainerKeyField6Var	Link To Container Key Field 6	✗	✗	✓	✗
LinkToContainerKeyField7Var	Link To Container Key Field 7	✗	✗	✓	✗

		Supported Expressions			
LinkToContainerKeyField8Var	Link To Container Key Field 8	✗	✗	✓	✗
LinkToContainerKeyField9Var	Link To Container Key Field 9	✗	✗	✓	✗
LinkToContainerKeyField10Var	Link To Container Key Field 10	✗	✗	✓	✗
LinkToContainerKeyField11Var	Link To Container Key Field 11	✗	✗	✓	✗
LinkToContainerKeyField12Var	Link To Container Key Field 12	✗	✗	✓	✗
LinkToContainerKeyField13Var	Link To Container Key Field 13	✗	✗	✓	✗
LinkToContainerKeyField14Var	Link To Container Key Field 14	✗	✗	✓	✗
LinkToContainerKeyField15Var	Link To Container Key Field 15	✗	✗	✓	✗
RecordConditionExpr	Record Condition	✓	✓	✓	✓
OutputEmailTo1	Output E-Mail To 1	✓**	✓**	✗	✓**
OutputEmailFrom1	Output E-Mail From 1	✓**	✓**	✗	✓**

		Supported Expressions			
OutputEmailTo2	OutputEmailFrom2	✓**	✓**	✗	✓**
OutputDocArchNumber1	Output Doc. Arch. No. 1	✓*	✓*	✗	✓*
OutputDocArchNumber2	Output Doc. Arch. No. 1	✓*	✓*	✗	✓*
InputDocArchNumber1	Input Doc. Arch Number 1	✓*	✓*	✗	✓*
EmailToOnCompletion	Email To	✓**	✓**	✗	✓**
EmailFromOnCompletion	Email From	✓**	✓**	✗	✓**
EmailSubjectOnCompletion	Email Subject	✓*	✓*	✗	✓*
EmailBodyOnCompletion	Email Body	✓*	✓*	✗	✓*
GroupByContextFieldName	Group By Context Field Name	✗	✓	✗	✗
Table Pane					
ParameterValueVar	Value	✓*	✓*	✓*	✓*

* Values entered in these fields will only be interpreted as expressions when the "Expr." field is set to Yes.

** Values entered in these fields will only be interpreted as expressions if the entered value is not a valid email address(-es).

Background Tasks

		Supported Expressions		
Internal Field Name	External Field Name	Standard Functions	Container Field References	Combined Expressions
KeyField1Name	Key Field 1 Name	✗	✓	✗

		Supported Expressions		
KeyField2Name	Key Field 2 Name	✗	✓	✗
KeyField3Name	Key Field 3 Name	✗	✓	✗
KeyField4Name	Key Field 4 Name	✗	✓	✗
Table Pane				
ParameterValueVar	Value	✓*	✓*	✓*

* Values entered in these fields will only be interpreted as expressions when the "Expr." field is set to Yes.

Supported Standard Functions

In this chapter we described all the different type of standard functions that are supported in expression fields. Note that all of these functions are also supported in the MDML language.

Date and Time Functions

The table below describes date and time functions for manipulating Maconomy's Date and Time values. Note that operations that take Date or Time might throw an exception if an empty date or time is provided i.e. NullDate or NullTime.

Function	Description	Examples
currentTime(): Time	Returns the current server time.	currentTime() = time(12,58,18)
currentDate(): Date	Returns the current server date.	currentDate() = date(2020,12,11)
userTime(): Time	Returns the current time on the client. Note that the userTime() and currentTime() will give the same results when using them in background task expression fields, since background tasks are always run on the server side.	userTime() = time(12,58,18)
userDate(): Date	Returns the current date on the client. Note that the userDate() and currentDate() will give the same results when using them in background task expression fields, since background tasks	userDate() = date(2020,12,11)

Function	Description	Examples
	are always run on the server side.	
NullDate: Date	Returns a date that is smaller than any other date. This function can be used to check whether a date field is empty.	NullDate < date(1200, 1, 1) = true
NullTime: Time	Returns a time that is smaller than any other time. This function can be used to check whether a time field is empty.	NullTime < time(3,20,55) = true
date(year: Integer, month: Integer, day: Integer): Date	Constructs a new date from the given year, month and day and returns it.	date(2015,11,7) = 2015-11-07 date(98,5,16) = 1998-05-16
year(date: Date): Integer	Returns the year part of the given date.	year(date(2015,11,7)) = 2015 year(date(98,5,16)) = 98
month(date: Date): Integer	Returns the month part of the given date.	month(date(2015,11,7)) = 11
day(date: Date): Integer	Returns the day part of the given date.	day(date(2015,11,7)) = 7
week(date: Date): Integer	Returns the week number of the given date.	week(date(2017,12,31)) = 52 week(date(2020,12,31)) = 53
intWeekday(date: Date): Integer	Returns the week day (as an Integer) of the given date.	intWeekday(date(2015,11,7)) = 6
stringWeekday(date: Date): String	Returns the week day (as a String) of the given date.	stringWeekday(date(2015,11,7)) = Saturday
yearsBetween(date1: Date, date2: Date): Integer	Returns the number of years between two dates.	yearsBetween(date(1998,5,16), date(2015,11,07)) = 17 yearsBetween(date(2015,11,07), date(1998,5,16)) = -17 yearsBetween(date(2015,11,07), date(2015,05,11)) = 0
monthsBetween(date1: Date, date2: Date): Integer	Returns the number of months between two dates.	monthsBetween(date(2015,05,11), date(2015,11,07)) = 5
daysBetween(date1: Date, date2: Date): Integer	Returns the number of days between two dates.	daysBetween(date(2015,10,30), date(2015,11,07)) = 8

Function	Description	Examples
		daysBetween(date(2015,11,11), date(2015,11,07)) = -4
addYears(date: Date, years: Integer): Date	Adds years to the given date and returns the resulting date.	addYears(date(98,5,16), 2) = 2000-05-16 addYears(date(2000,11,08), -5) = 1995-11-08
addMonths(date: Date, months: Integer): Date	Adds months to the given date and returns the resulting date.	addMonths(date(98,5,16), 8) = 1999-01-16
addDays(date: Date, days: Integer): Date	Adds days to the given date and returns the resulting date.	addDays(date(98,5,16), 17) = 1998-06-02
addPeriod(date: Date, years: Integer, months: Integer, days: Integer): Date	Adds the given years, months, and days to the given date and returns the resulting date.	addPeriod(date(2012, 05, 10), 2, 4, 16) = date(2014,09,26)
time(hour: Integer, minute: Integer, second: Integer): Time	Constructs a new time from the given hour, minute and second and returns it.	time(22,5,15) = 22:05:15 time(3,20,55) = 03:20:55
time(hour: Integer, minute: Integer): Time	Constructs a new time from the given hour and minute and returns it.	time(23,15) = 23:15:00
hour(time: Time): Integer	Returns the hour part of the given time.	hour(time(22,5,15)) = 22
minute(time: Time): Integer	Returns the minute part of the given time.	minute(time(22,5,15)) = 5
second(time: Time): Integer	Returns the seconds part of the given time.	second(time(22,5,15)) = 15
addHours(time: Time, hours: Integer): Time	Adds hours to the given time and returns the resulting time.	addHours(time(22,5,15), 3) = 01:05:15 addHours(time(22,5,15), -4) = 18:05:15
addMinutes(time: Time, minutes: Integer): Time	Adds minutes to the given time and returns the resulting time.	addMinutes(time(22,5,15), 25) = 22:30:15 addMinutes(time(22,5,15), -10) = 21:55:15
addSeconds(time: Time, seconds: Integer): Time	Adds seconds to the given time and returns the resulting time.	addSeconds(time(22,5,15), 45) = 22:06:00

Function	Description	Examples
		addSeconds(time(22,5,15), -14) = 22:05:01

String Functions

String functions provide a number of helpful functions for manipulating strings.

Function	Description	Examples
length(str: String): Integer	Function that returns the length of the given string str.	`length('Hello World!') = 12`
startsWith(str1: String, str2: String): Boolean	Function that returns true if the given string str1 starts with string str2 and false otherwise.	`startsWith('Hello World!', 'Hell') = true` `startsWith('Hello World!', 'World') = false`
endsWith(str1: String, str2: String): Boolean	Function that returns true if the given string str1 ends with string str2 and false otherwise.	`endsWith('Hello World!', 'World!') = true` `endsWith('Hello World!', 'Hello') = false`
firstIndexOf(str1: String, str2: String): Integer	Function that returns the index of the first occurrence of string str2 in string str1. Returns -1 if str2 does not occur in str1.	`firstIndexOf('No former performer performed this performance', 'for') = 3` `firstIndexOf('No former performer performed this performance', 'of') = -1`
lastIndexOf(str1: String, str2: String): Integer	Function that returns the index of the last occurrence of string str2 in string str1. Returns -1 if str2 does not occur in str1.	`lastIndexOf('No former performer performed this performance', 'for') = 38` `lastIndexOf('No former performer performed this performance', 'of') = -1`
indexOf(str1: String, str2: String, fromIndex: Integer): Integer	Function that returns the index of the first occurrence of string str2 in string str1 starting from position specified by the fromIndex. Returns -1 if str2 does not occur in str1.	`indexOf('No former performer performed this performance', 'for', 4) = 13`
charAt(str: String, index: Integer): String	Function that returns the character at position specified by the given index in string str.	`charAt('hello', 4) = 'o'` `charAt('hello', 10) = ''`

Function	Description	Examples
contains(str1: String, str2: String): Boolean	Function that returns true if str1 contains str2 and false otherwise.	`contains('No former performer performed this performance', 'former') = true` `contains('No former performer performed this performance', 'hello') = false`
substring(str: String, startIndex: Integer): String	Function that returns a new string containing the substring of the given string str from the specified index startIndex (inclusive).	`substring('Hello World!', 6) = 'World! '` `substring('Hello World!', 12) = ''`
substring(str: String, startIndex: Integer, endIndex: Integer): String	Function that returns a new string containing the substring of the given string str from the specified startIndex (inclusive) to the specified endIndex (non-inclusive).	`substring('Hello World!', 6, 8) = 'Wo'`
trim(str: String): String	Function that returns a new string by trimming the trailing and leading spaces, tabs, and new lines from the given string str.	`trim('  Hello World !  ') = 'Hello World!'`
uppercase(str: String): String	Function that returns a new string by uppercasing the given string str.	`uppercase('Hello World!') = 'HELLO WORLD!'`
lowercase(str: String): String	Function that returns a new string by lowercasing the given string str.	`lowercase('Hello World!') = 'hello world!'`
replaceFirst(str: String, replace: String, replaceWith: String): String	Function that returns a new string by replacing the first occurrence of the **replace** text with **replaceWith** in the given string **str**.	`replaceFirst('No former performer performed this performance', 'form', 'corni') = 'No cornier performer performed this performance'`
replaceAll(str: String, replace: String, replaceWith: String): String	Function that returns a new string by replacing all occurrences of the **replace** text with **replaceWith** in the given string **str**.	`replaceAll('No former performer performed this performance', 'form', 'corni') = 'No cornier percornier percornied this percorniance'`

Function	Description	Examples
replaceFirstRegEx(str: String, regex: String, replaceWith: String): String	Function that returns a new string by replacing the first occurrence matched by the given regex with the **replaceWith**.	`replaceFirstRegEx('No former performer performed this performance', ' ([a-z]{3})', ' $1$1') = 'No forformer performer performed this performance'`
replaceAllRegEx(str: String, regex: String, replaceWith: String): String	Function that returns a new string by replacing all occurrences matched by the given regex with the **replaceWith**.	`replaceAllRegEx('No former performer performed this performance', ' ([a-z]{3})', ' $1$1') = 'No forformer perperformer perperformed thithis perperformance'`
matchRegEx(str: String, regEx: String): Boolean	Function that returns true if the given **regEx** matches the given string **str** and false otherwise.	`matchRegEx('555-2368', '\\d{3}-\\d{4}') = true` `matchRegEx('5555-2368', '\\d{3}-\\d{4}') = false`

asReal(str: String): Real	Function that converts the given string str to a real. Throws an exception if the given string cannot be converted to real.	`asReal('3.14159265359') = 3.14159265359`
		`asReal('1.000.000,25') = 1000000.25`
		`asReal('1\'000\'000.30') = 1000000.3`

stringToInteger(str: String): Integer	Function that converts the given string str to an integer. Throws an exception if the given string cannot be converted to integer.	`stringToInteger('123') = 123`
stringToReal(str: String): Real	Function that converts the given string str to real. Throws an exception if the given string cannot be converted to real	`stringToReal('3.14159265359') = 3.14159265359`

Popup Functions

Popup functions provide a number of functions for working with popups.

Function	Description	Examples
popupTitle(popup: Popup): String	Function that returns the	`popupTitle(JournalTypeType'Time_Sheet) = 'Time Sheet'`

Function	Description	Examples
	title of the given popup.	`popupTitle(JournalTypeType'nil) = ''`
popupLiteralName(popup: Popup): String	Function that returns the popup literal name of the given popup.	`popupLiteralName(JournalTypeType'Time_Sheet) = 'Time_Sheet'` `popupLiteralName(JournalTypeType'nil) = 'nil'`
popupOrdinal(popup: Popup): String	Function that returns the ordinal value of the given popup.	`popupOrdinal(JournalTypeType'Time_Sheet) = 4`
popup(popupType: String, literalName: String, ordinalValue: Integer): Popup	Function that constructs and returns a popup based on the given type name, literal name, and ordinal value.	`popup('JournalTypeType', 'Time_Sheet', 4) = JournalTypeType'Time_Sheet`

Numeric Type Conversion and Formatting Functions

These functions provide the ability to convert between different Maconomy numeric types, i.e. Integers, Reals, and Amounts, as well as print them out using custom formatting rules.

Function	Description	Examples

toInteger(number: (Integer | Real | Amount)): Integer	Function that converts the given number to integer. Will throw an exception if the given number is too large or too small to fit into integer.	`toInteger(amount(1.13)) = 1`
		`toInteger(3.14159265359) = 3`
		`toInteger(4.9) = 4`

toAmount(number: (Integer | Real | Amount)): Amount	Function that converts the given number to amount. Will throw an exception if the given number cannot be converted to amount.	`toAmount(2) = amount(2.00)`
		`toAmount(3.14159265359) = amount(3.14)`
		`toAmount(13.349) = amount(13.34)`

toReal(number: (Integer | Real | Amount)): Real	Function that converts the given number to real. Will throw an exception if the	`toReal(2) = 3.0` `toReal(amount(12.49)) = 12.49`

Function	Description	Examples
	given number cannot be converted to real.	

formatNumber(number: (Integer | Real | Amount), format: String): String	Function that formats the given number according to the given format.	`formatNumber(123456, "#,##0.00") = '123,456.00'`
		`formatNumber(123456, "#,##0") = '123,456'`
		`formatNumber(amount(25), '#.00 USD') = '25.00 USD'`

formatNumber(number: (Integer | Real | Amount), format: String, groupingSep: String): String	Function that formats the given number according to the given format and the grouping separator.	`formatNumber(123456, "#,##0", " ") = '123 456'` `formatNumber(123456, "#,##0", "-") = '123-456'`
formatNumber(number: (Integer | Real | Amount), format: String, groupingSep: String, decimalSep: String): String	Function that formats the given number according to the given format, the grouping separator, and decimal separator.	`formatNumber(1234, "#,##0.00", " ", "^") = '1 234^00'` `formatNumber(1234567.12, "#,##0.00", "-", "/") = '1-234-567/12'`

Math Functions

Math functions provide a number of helpful functions for working with Maconomy numbers, i.e. Integers, Reals, and Amounts. Besides the functions described below, you may also use regular mathematical operators (+, -, *, /) in your expressions.

Function	Description	Examples
min(numbers: (Integer | Real | Amount)*): (Integer | Real | Amount)	Function that returns the smallest number out of all the numbers provided.	`min(2, 10, 25, 1, 8, 3) = 1` `min(-2, 5, 10, amount(-7.13), asReal('20.14159')) = amount(-7.13)`
max(numbers: (Integer | Real | Amount)*): (Integer | Real | Amount)	Function that returns the largest number out of all the numbers provided.	`max(2, 10, 25, 1, 8, 3) = 25` `max(-2, 5, 10, amount(-7.13), asReal('20.14159')) = 20.14159`

abs(number: (Integer | Real | Amount): (Integer | Real | Amount)	Function that returns the absolute value of the given number.	`abs(-123) = 123`
		`abs(amount(-7.13)) = amount(7.13)`
		`abs(asReal('-3.14159265359')) = 3.14159265359`

Function	Description	Examples

sign(number: (Integer | Real | Amount): Integer	Function that returns 1 if the given number is positive, -1 if the given number is negative and 0 if the number is 0.	`sign(-3) = -1`
		`sign(3.13) = 1`
		`sign(0) = 0`

round(number: (Integer | Real | Amount): (Integer | Amount)	Function that rounds the given number to the nearest value.	`round(10.49) = 10`
		`round(10.51) = 11`
		`round(amount(-10.51)) = amount(-11.00)`
		`round(asReal('-11.56')) = -12`

floor(number: (Integer | Real | Amount): (Integer | Amount)	Function that returns the largest value that is less than or equal to the given number.	`floor(10.99) = 10`
		`floor(amount(-13.49)) = amount(-14.00)`
		`floor(asReal('3.14159')) = 3`

ceiling(number: (Integer | Real | Amount): (Integer | Amount)	Function that returns the smallest value that is greater than or equal to the given number.	`ceiling(10.99) = 11`
		`ceiling(amount(-13.49)) = amount(-13.00)`
		`ceiling(asReal('3.14159')) = 4`

User Functions

Similar to previously mentioned functions, user functions are standard functions that can be used in expression fields. However, unlike previous functions, in most cases these functions would only produce a useful output when using them on background tasks that run as a specific user. This is the case because these functions return information about the current user. If these functions are used on background tasks where “Run As User” is not specified then they will run in the context of the default batch administrator user specified in the server.ini file.

Full list of user functions can be seen in the table below.

Function	Description	Examples
username(): String	Returns the username of the current user.	`if username() = 'John' then 'Hi John!' else 'You are not John!'`
isAdministrator(): Boolean	Returns true if the current user is an administrator and false otherwise.	`if isAdministrator() then 'Access allowed' else 'Access forbidden'`

Function	Description	Examples
hasRole(groupName: String): Boolean	Returns true if the current user has the given group groupName (a String) assigned and false otherwise.	`if hasRole('ProjectManager') then 'Hello Project Manager!' else 'Hello!'`
hasRole(groupName: GroupNameType): Boolean	Returns true if the current user has the given group groupName (a Popup) assigned and false otherwise.	`if hasRole(GroupNameType'ProjectManager) then 'Hello Project Manager!' else 'Hello!'`
userEmployeeNumber(): String	Returns the employee number assigned to the current user.*	`if userEmployeeNumber() = ' ' then 'No employee assigned to this user!' else 'Hello ' + userEmployeeNumber() + '!'`
userRole(): String	Returns the user role of the current user.	`if userRole() = 'Standard' then 'Standard Role' else 'Non-standard Role'`
companyNumber(): String	Returns the company number on the employee assigned to the current user.*	`if companyNumber() = '1' then 'Company 1' else 'Another company'`
accountNumber():String	Returns the account number on the employee assigned to the current user.*	`if accountNumber() = '2110' then 'P&L' else 'Balance Sheet'`
locationName(): String	Returns the location name on the employee assigned to the current user.*	`if locationName() = 'France' then 'Bonjour!' else if locationName = 'Germany' then 'Guten Tag!' else 'Hello!'`
entityName(): String	Returns the entity name on the employee assigned to the current user.*	`if entityName() = 'E1' then 'Entity 1' else 'Another entity'`
projectName(): String	Returns the project name on the employee assigned to the current user.*	`if projectName() = 'P1' then 'Project 1' else 'Another project'`
purposeName(): String	Returns the purpose name on the employee	`if purposeName() = 'P1' then 'Purpose 1' else 'Another purpose'`

Function	Description	Examples
	assigned to the current user.*	
specification1Name(): String	Returns the specification 1 name on the employee assigned to the current user.*	`if specification1Name() = 'Spec1' then 'Specification 1' else 'Another specification'`
specification2Name(): String	Returns the specification 2 name on the employee assigned to the current user.*	`if specification2Name() = 'Spec2' then 'Specification 2' else 'Another specification'`
specification3Name(): String	Returns the specification 3 name on the employee assigned to the current user.*	`if specification3Name() = 'Spec3' then 'Specification 3' else 'Another specification'`
localSpec1Name(): String	Returns the local specification 1 name on the employee assigned to the current user.*	`if localSpec1Name() = 'LocalSpec1' then 'Local Specification 1' else 'Another local specification'`
localSpec2Name(): String	Returns the local specification 2 name on the employee assigned to the current user.*	`if localSpec2Name() = 'LocalSpec2' then 'Local Specification 2' else 'Another local specification'`
localSpec3Name(): String	Returns the local specification 3 name on the employee assigned to the current user.*	`if localSpec3Name() = 'LocalSpec3' then 'Local Specification 3' else 'Another local specification'`

* **NOTE**: the output will be an empty string if no employee has been assigned to the current user.

System Table and Environment Functions

Maconomy has two tables for storing system-wide information, namely System Information and System Parameters. Sometimes it is useful to acquire the information from these tables and use it on a background task. Maconomy provides a number of helpful functions that can ease this task. For these functions to work correctly the CouplingConfiguration.mcsl.xml file must be updated correspondingly.

To be able to reference fields from the System Information table you must add them to the “SystemInformation” entity binding in the CouplingConfiguration file. E.g. if you would like to be able to get the value of the field “YearEndResultAccount” then you would modify your coupling configuration as follows:

```
<Binding entity="SystemInformation" namespace="system:info">
  <Fields>
    <Field source="YearEndResultAccount" />
  </Fields>
</Binding>
```

Similarly, to be able to reference fields from the System Parameter table you must add them to the "SystemParameter" entity binding in the CouplingConfiguration file. E.g. if you would like to be able to get the values of the fields "BPMServerURL" and "UseVATSubLevels" then you would modify your coupling configuration as follows:

```
<Binding entity="SystemParameter"
namespace="system:parameters:^{envVar('InternalName')}">
  <Restriction condition="InternalName = 'BPMServerURL' or
                          InternalName = 'UseVATSubLevels'" />
  <Fields>
    <Fields ref="CommonSystemParameters" />
    <Field source="TheGroup" />
    <Field source="Description" />
    <Field source="Format" />
    <Field source="AllowCompanyParameter" />
  </Fields>
</Binding>
```

Making these changes will require the restart of the Coupling Service.

The supported functions are documented in the table below.

Function	Description	Examples

sysInfo(fieldName: String): T	Function that returns a system information value of type T for the given fieldName. Type T depends on the type of the field name.	`if sysInfo('MarkUpPercentage') >= 2 then 'Markup Percentage is 2 or higher' else 'Markup percentage is less than 2'`
		
		`if sysInfo('CreateNewOrderAtJobAssignment') then 'New Ord. when Transf. to Sales' else 'No new orders'`

amountSysInfo(fieldName: String): Amount	Function that returns a system information value of type amount for the	`There are currently no fields of type Amount on the System Information table.`
		
		`amountSysInfo('SomeAmountField')`

Function	Description	Examples
	given fieldName.	
booleanSysInfo(fieldName: String): Amount	Function that returns a system information value of type boolean for the given fieldName.	`if booleanSysInfo('CreateNewOrderAtJobAssignment') then 'New Ord. when Transf. to Sales' else 'No new orders'`
dateSysInfo(fieldName: String): Date	Function that returns a system information value of type date for the given fieldName.	`dateSysInfo('CreatedDate')`
decimalSysInfo(fieldName: String): Real	Function that returns a system information value of type real for the given fieldName.	`decimalSysInfo('MarkUpPercentage')`
integerSysInfo(fieldName: String): Integer	Function that returns a system information value of type integer for the given fieldName.	`integerSysInfo('MaximumFileSize')`
popupSysInfo(fieldName: String): Popup	Function that returns a system information value of type popup for the given fieldName.	`popupSysInfo('EnterpriseExchangeRateTable')`

Function	Description	Examples
stringSysInfo(fieldName: String): String	Function that returns a system information value of type string for the given fieldName.	`stringSysInfo('StandardVATCode')`
timeSysInfo(fieldName: String): Time	Function that returns a system information value of type time for the given fieldName.	`There are currently no fields of type Time on the System Information table.` `timeSysInfo('SomeTimeField')`

sysPar(systemParameter: String): T	Function that returns a system parameter value of type T for the given systemParameter. Type T depends on the type of the system parameter.	`if sysPar('BPMServerURL') = ' ' then 'No BPM server defined' else sysPar('BPMServerURL')`
		
		`if sysPar('UseVATSubLevels') then 'Using VAT Sub Levels' else 'Not using VAT Sub Levels'`

amountSysPar(systemParameter: String): Amount	Function that returns a system parameter value of type amount for the given systemParameter.	`There are currently no system parameters of type amount.` `amountSysPar('SomeAmountParameter')`
booleanSysPar(systemParameter: String): Boolean	Function that returns a system parameter value of type boolean for the	`booleanSysPar('UseVATSubLevels')`

Function	Description	Examples
	given systemParameter.	
dateSysPar(systemParameter: String): Date	Function that returns a system parameter value of type date for the given systemParameter.	`There are currently no system parameters of type date.` `dateSysPar('SomeDateParameter')`
decimalSysPar(systemParameter: String): Real	Function that returns a system parameter value of type real for the given systemParameter.	`decimalSysPar('NumberOfHoursPerManDay')`
integerSysPar(systemParameter: String): Integer	Function that returns a system parameter value of type integer for the given systemParameter.	`integerSysPar('MaxNumberOfEntriesJobJournals')`
popupSysPar(systemParameter: String): Popup	Function that returns a system parameter value of type popup for the given systemParameter.	`There are currently no system parameters of type popup.` `dateSysPar('SomePopupParameter')`
stringSysPar(systemParameter: String): String	Function that returns a system parameter	`stringSysPar('BPMServerURL')`

Function	Description	Examples
	value of type string for the given systemParameter.	
timeSysPar(systemParameter : String): Time	Function that returns a system parameter value of type time for the given systemParameter.	`There are currently no system parameters of type time.` `timeSysPar('SomeTimeParameter')`

As an alternative to system information and system parameter functions there also exist environment functions. In fact, system information and system parameter functions are simply wrappers around the environment functions. You can achieve exactly the same result using environment functions with only difference being that environment functions take a fully qualified name (namespace + field name) of the field name. Moreover, environment functions are not restricted to only system information fields and system parameters and can be invoked on any field from any binding defined in the coupling configuration file.

As an example, let us assume that we want to define a new, custom binding with a Boolean field “UseLocalAccounts”. To do this we would define a new binding in the coupling configuration file as follows:

```
<Binding namespace="our.custom.binding">
  <Fields>
    <Field name="UseLocalAccounts" value="true" />
  </Fields>
</Binding>
```

After the restart of the coupling service, we should now be able to reference the value of this field via the following expression:

```
booleanEnvVar('our.custom.binding.UseLocalAccounts')
```

The complete list of environment functions is provided in the table below.

Function	Description	Examples

envVar(fieldName: String): T	Function that returns an environment value of type T for the given fieldName. Type T depends on the type of the field name.	`if envVar('system.info.MarkUpPercentage') >= 2 then 'Markup Percentage is 2 or higher' else 'Markup percentage is less than 2'`
		
		`if envVar('system.info.CreateNewOrderAtJobAssignment') then 'New Ord. when Transf. to Sales' else 'No new orders'`

isEnvVarDefined(fieldName: String): Boolean	Returns true if the given environment variable is defined and false otherwise.	`isEnvVarDefined('user.info.AccessToAllTimeSheets') = true` `isEnvVarDefined('custom.binding.AbcField') = false`
amountEnvVar(fieldName: String): Amount	Function that returns an environment value of type amount for the given fieldName.	`amountEnvVar('custom.binding.SomeAmountField')`
booleanEnvVar(fieldName: String): Boolean	Function that returns an environment value of type boolean for the given	`booleanEnvVar('system.parameters.UseVATSubLevels.Boolean1')` `booleanEnvVar('user.info.AccessToAllTimeSheets')`

Function	Description	Examples
	fieldName.	
dateEnvVar(fieldName: String): Date	Function that returns an environment value of type date for the given fieldName.	`dateEnvVar('custom.binding.SomeDateField')`
decimalEnvVar(fieldName: String): Real	Function that returns an environment value of type real for the given fieldName.	`decimalEnvVar('custom.binding.SomeRealField')`
integerEnvVar(fieldName: String): Integer	Function that returns an environment value of type integer for the given fieldName.	`decimalEnvVar('custom.binding.SomeIntegerField')`
popupEnvVar(fieldName: String): Popup	Function that returns an environment value of type popup for the given fieldName.	`popupEnvVar('custom.binding.SomePopupField')`

Function	Description	Examples

stringEnvVar(fieldName : String): String	Function that returns an environment value of type string for the given fieldName .	`stringEnvVar('system.parameters.BPMServerURL.String1')`
		`stringEnvVar('user.roles.NameOfRole')`
		`stringEnvVar('user.info.Specification8Name')`

timeEnvVar(fieldName: String): Time	Function that returns an environment value of type time for the given fieldName .	`timeEnvVar('custom.binding.SomeTimeField')`

Other Functions

A number of additional functions that can be used for various purposes.

Function	Description	Examples
shortName(): String	Function that returns the shortname of the current system.	`shortName() = 'mac210test'`

isNullOrEmpty(value: T): Boolean	Function that returns true if the given Maconomy type value is null or empty and	`isNullOrEmpty(NullDate)) = true`
		`isNullOrEmpty(NullTime) = true`
		`isNullOrEmpty(date(2012,10,02)) = false`
		`isNullOrEmpty(GRPType'nil) = true`
		`isNullOrEmpty('') = true`

Function	Description	Examples
	false otherwise.	
hasAddon(addonNo: Integer): Boolean	Function that returns true if the given addon is installed on the current system and false otherwise.	`hasAddon(3) = true` `hasAddon(131) = false`
urlEncode(str: String): String	Function that encodes the given string so that it can be used in URL.	`'https://somesite.com?q=' + urlEncode('String with spaces') = 'https://somesite.com?q=String%20with%20spaces'`

format(maconomyType: T, formatting: String): String	Function that can format various Maconomy types according to the specified formatting.	`format(date(2010, 03, 15), 'dd.MM.yyyy') = '15.03.2010'`
		`format(date(2010, 03, 15), 'E, dd MMM yyyy HH:mm:ss zzzz') = 'Mon, 15 Mar 2010 00:00:00 Central European Time'`
		`format(2010, '#,###,###.00') = '2.010,00'`
		`format(time(13, 46), 'HH:mm:ss a') = '13:46:00 PM'`
		`format(10.1234, '#.###') = '10,123'`

Fields and Actions

Note: The information in this section is in DRAFT form and is not yet updated to reflect 2.5.2.

Fields

Setup » Background Tasks workspace » Editing section » Background Task tab

Background Task island

Field	Description
Id	This field displays the unique ID of the task. When you click the **New Background Task** action, Maconomy automatically assigns an ID number to the created task.
Container	In this field, enter the internal name of the container that hosts the task to be carried out. For example: “maconomy:TimeSheets” or “TimeSheets” (since the name space “maconomy:” is assumed by default).
Pane Type	In this field, specify the pane type that contains the action. Select one of the following values from the drop-down list: ▪ Filter ▪ Card ▪ Table The default value is **Card**.
Action Name	In this field, enter the internal name of the named action that this background task should carry out. You can also specify the internal name of a standard action. To do this, enter a value in the **Standard Action** field.
Standard Action	In this field, you can specify the standard action to be carried out. If you leave this blank, it means the action to be carried out is a named action. If you select a value from the drop-down list, the **Action Name** field automatically assumes the corresponding internal name of the standard action. The available values are: ▪ **Create** – Select to create exactly one record. ▪ **Update** ▪ **Delete** ▪ **Print**

Task Properties island

Field	Description
Activation	This field displays either of two values: “Active” or “Inactive”, depending on whether the task is currently active. Maconomy obtains this value from the **Active** check box which is not part of the default layout, and which changes its own value when the user clicks the **Activate** or **Deactivate** action for the task. Active tasks can be picked up for execution once the due date and the due time have passed and there are no unfinished background tasks belonging to the collection ID indicated in the **Awaits Collection ID** field. You can only edit inactive tasks.
Task Description	In this field, enter a description of the task.
Access Level	In this field, specify an access level for the task.
Collection Id	In this field, you can specify a Collection ID. This is optional. By default, Maconomy sets the collection ID to a value corresponding to the ID of this task, thereby making the task the only member of the collection. Tasks with the same collection ID belong to the same collection. If at least one of these tasks is pending or running, any task that depends on that collection ID cannot be picked up for execution.
Identification	In this field, enter identifying information to indicate the purpose of the task. For example, you can enter the ID of the programmatic extension or task generator that initiates this background task on behalf of the user.
System Task	If the task is a system task, this field displays a **Yes** value. System tasks are those created and maintained by Maconomy to keep the processing of background tasks active. You cannot delete or deactivate pending system tasks. If the task is not a system task, this field is left blank.
System Nature	In this field, you can specify a system nature. System natures are defined in the configuration file server.ini file on the Maconomy Server. If you fill out this field, Maconomy only carries out the background task on database short names that belong to the specified system nature. This is useful when data from a production database is copied to test-database environments, and vice versa. In cases like this, the background tasks and schedule rules for generating background tasks are also copied. By specifying a system nature, you ensure

Field	Description
	that production tasks are carried out in production environments only, and test tasks in test environments only.
Max. Duration	In this field, you can specify the maximum duration of the background task. This is used to compute the length of time that should pass before the execution thread running the task times out. Even if you filled out the **Default Max. Task Duration** field for the execution thread, you can fill out the **Max. Duration** field for specific tasks in the thread. This allows you to specify a longer time duration for a long-running task, preventing Maconomy from timing out and prematurely aborting the task. Note that you can also specify the maximum duration for tasks generated by a background task generator.
Require Execution	Select this check box if you want Maconomy to consider the task as “failed” if it is unable to carry out the specified action for at least one record. For example: You have a background task that is supposed to post a journal marked with a special remark. If such a journal does not exist when the task is picked up for execution and you have this check box selected, Maconomy marks the tasks as “failed”, thereby bringing it to the attention of the one in charge of monitoring background tasks. By default, this check box is not selected. This means that if there are no records on which the action is performed, Maconomy simply marks the task as successfully run.
Delete Automatically	Select this check box if you want Maconomy to automatically delete the background task after successful execution. Note that if the task is either failed or partially successful (with an “Incomplete” status), it is not deleted. The one in charge of monitoring can still investigate why the task was not completed successfully.

Run As island

Field	Description
User Name	In this field, you can specify the user on whose behalf Maconomy will carry out the background task. You can only specify a user with administrator credentials. If a non-administrator user creates a background task, that task assumes the user name specified in this field. Note that ordinary

Field	Description
	users cannot create a background task for an operation they cannot perform manually. If you leave this field blank, Maconomy carries out the task on behalf of the background task user configured for the system.
Empl. No.	In this field, you can enter the employee number on whose behalf Maconomy will carry out the background task. If you fill out this field, Maconomy automatically updates the **User Name** field with the corresponding user and then clears this field. If several users are associated with the specified employee, Maconomy either selects the non-admininistrator user or (if all users are administrators) selects one of the users arbitrarily.
Role Name	In this field, you can specify the role name of the user on whose behalf Maconomy will carry out the background task. Note that several roles can be assigned to or associated with a user. If you leave this field blank, Maconomy carries out the task on behalf of whichever role is specified as the “Default for background tasks” for the user.
Login Name	In this field, you can specify the login name of the user/role on whose behalf Maconomy will carry out the background task.

Due island

Field	Description
Due Date	Click the down arrow, and select a due date from the calendar. Maconomy will generate a background task which will only be carried out after the specified date. If you leave this blank, Maconomy interprets the task as due immediately.
Due Time	In this field, enter a specific time of day following the HH:mm:ss 24-hour clock format. If you specify a date in the **Due Date** field, you must also fill out this field. Maconomy will generate a background task which will only be carried out after the specified time on the specified due date. If you leave the **Due Date** field blank, leave this field blank as well.
Await Collection Id	In this field, enter a collection ID if you want Maconomy to not consider the task as due when there are scheduled or running tasks associated with the specified collection ID.

Re-Attempts on Data Busy island

Field	Description
Max. Execution Re-Attempts	In this field, you can specify the number of times the system is allowed to re-attempt a task if execution fails because data is locked to another user. For example, a background task will fail if another user is editing the record that the task needs. If you specify a value of **3** in this field and the first attempt to carry out the task fails, the system automatically reschedules a second attempt, and a third if needed. If the third attempt still fails, the task is marked as failed. If you specify a value of **0**, the system does not reattempt the task at all. If you want to specify default values for an entire execution thread, fill out the **Default Max. Exec. Re-Attempts** field for that thread.
Secs. Between Re-Attempts	In this field, you can specify the number of seconds between system re-attempts to carry out a task. For example, if you specify a value of **10**, the system reattempts the task ten seconds after the preceding attempt. The value must fall within the range of 0 – 3600 seconds (or one hour). If you want to specify default values for an entire execution thread, fill out the **Default Secs. Between Re-Attempts** field for that thread.

Container Key island

Field	Description
Match Any	Select this check box if you want the task to apply to all possible keys. For example, if you specified the Employees container, selecting this check box means the action will apply to all employees. Always select this check box for singletons (for example: Import, Print, and some selection criterion containers). Do not select this check box if: ▪ You specified one or more key field value constraints. ▪ The specific background task invokes the **Create** action in a Card pane. ▪ The specific background task invokes the **Create** action in a Table pane, unless you want to create lines for all table sets (that is, tables belonging to all Card pane keys). For example, specifying the **Create** action and selecting this check box for Time sheet tables means Maconomy will create a time sheet line in all existing time sheets. When you select this check box, Maconomy displays a warning message since this is not usually a preferred setting.

Field	Description
Key Field 1 Name	In this field, enter the name of the first key field of the container. In principle, you can specify any field. It does not have to be a formal key field of the container. If you do not use any of the formal key fields when specifying the key value, Maconomy derives the set of formal key fields when applying the field/value restriction to the filter list of the container. You do not need to specify key field constraints if: ▪ You specified singletons. ▪ The specific background task invokes the **Create** action in a Card pane. ▪ The specific background task invokes the **Create** action in a Table pane, and you want to create lines for all table sets (that is, tables belonging to all Card pane keys).If this is the case, specify the **Create** action and select the **Match Any** check box, but do not specify key field constraints.
Key Field 1 Type	In this field, specify the key field type. You can specify any of the following values: ▪ Amount ▪ Boolean ▪ Integer ▪ Real ▪ String ▪ Date ▪ Time ▪ Enumeration (or popup)
Key Field 1 Value	In this field, enter the value of the first key field of the container. You enter the value as a string, then the system interprets it relative to the type you specify. The only exceptions are: ▪ **Enumeration-type fields** – Enter the value using the format `TypeName'literalValue.` For example: `CountryType'United Kingdom` or `OrderTypeType'Sales Order.` ▪ **Boolean-type fields** – Enter the value as true/false or Yes/No.
Key Field 2 Name	This field is similar to **Key Field 1 Name**, but you use this for the second key field for containers with two or more key fields.
Key Field 2 Type	This field is similar to **Key Field 1 Type**, but you use this for the second key field for containers with two or more key fields.

Field	Description
Key Field 2 Value	This field is similar to **Key Field 1 Value**, but you use this for the second key field for containers with two or more key fields.
Key Field 3 Name	This field is similar to **Key Field 1 Name**, but you use this for the third key field for containers with three or more key fields.
Key Field 3 Type	This field is similar to **Key Field 1 Type**, but you use this for the third key field for containers with three or more key fields.
Key Field 3 Value	This field is similar to **Key Field 1 Value**, but you use this for the third key field for containers with three or more key fields.
Key Field 4 Name	This field is similar to **Key Field 1 Name**, but you use this for the fourth key field for containers with four key fields.
Key Field 4 Type	This field is similar to **Key Field 1 Type**, but you use this for the fourth key field for containers with four key fields.
Key Field 4 Value	This field is similar to **Key Field 1 Value**, but you use this for the fourth key field for containers with four key fields.

Record Conditions island

Field	Description
Rec. Cond. Expr.	Use this field to specify the condition. You can express the condition based on the fields available in the pane. If the pane is of the Table type, you may refer to values in the corresponding Card part by prefixing “card” to the Card field names.
Ignore Disabled Action	Select this check box if you want Maconomy to treat this record-condition expression as false when the background task’s action is disabled. You can enable this if you do not need to know the actual conditions under which the action is disabled. For example: you have a background task for approving a specific time sheet. If this time sheet has not yet been submitted or has already been approved, Maconomy disables the **Approve Time Sheet** action. When the background task attempts to approve the time sheet using this action, an error occurs. However, if you selected this check box during setup, Maconomy disregards the disabled action. Some actions are preceded by an update. In such cases, Maconomy ignores the operation if either the update or the actual action is disabled. Note that in cases where several container keys fall within the scope of the background task, selecting this check box results in the task failing. This happens because the system cannot take such situations into account when multiple records/container keys are involved.

User island

Field	Description
Created	This field displays the date the background task was created, as well as the name of the user who created it. This information is mainly used for tracking purposes. Background tasks are most often created programmatically rather than directly by a user. The user specified in this field is rarely aware of the task creation.
Changed	This field displays the date the background task was last updated, as well as the name of the user who made these updates.
Version	This field displays the number of times the background task has been edited.

Output Document Handling (Using Files) island

Field	Description
Handle Files Like (Option 1)	If your background task involves document creation or generation, use this field to specify the filter for the output documents that Maconomy should handle using option 1. Use simple wild-card notation. ▪ An asterisk (*) means zero or more characters. ▪ A question mark (?) means exactly one character. ▪ For example: Enter the value ***.pdf** to match all output files that end in ".pdf". Enter **p_invoice.pdf** to match files named "p_invoice.pdf". Enter an ***** to match all files. If you do not specify a value, no output files are filtered and saved in your specified location.
File Location Ref. (Option 1)	In this field, specify a named reference to the file directory where you want to store the output file. To configure the actual directory, use the server.ini file on the Maconomy server. For example, you can specify different output locations for production and test systems. When data is copied from the production system into the test system, the background tasks store output data in different locations (depending on whether the tasks are run by the production system, or the test system).
File Location (Option 1)	In this field, specify the base-file location (for example, a file server) where Maconomy should place the output documents that match the specified filter. The actual documents can be stored in a sub-directory.
Handle Files Like (Option 2)	This field is similar to **Handle Files Like (Option 1)**, but this allows you to specify a different filter for output documents.

Field	Description
File Location Ref. (Option 2)	This field is similar to **File Location Ref. (Option 1)**, but this allows you to specify a named reference to a different file directory.
File Location (Option 2)	This field is similar to **File Location (Option 1)**, but this allows you to specify a different base-file location.

Output Document Handling (Using E-mail) island

Field	Description
Handle Files Like (Option 1)	This field is similar to the **Handle Files Like (Option 1)** field in the Output Document Handling (Using Files) island, but files that match the filter you specify here are e-mailed. Files that match both a file-handler and an e-mail handler are handled by both (that is, saved in a file directory and sent via e-mail).
E-mail To (Option 1)	In this field, specify the e-mail address (for example, a mailbox or administrator user) to which Maconomy should send the output files.
E-mail From (Option 1)	In this field, specify the e-mail address of the sender, as it will appear in the e-mail that Maconomy sends out. You do not have to specify a real e-mail address. For example: maconomy-background-tasks@yourorg.com. If you set up a default reply-to address using the `email.from.default` property in the server.ini file, you do not need to specify an address here. Maconomy uses the default address you specified in the server.ini file. If you enter an address in this field, Maconomy uses this value instead of the default address in the server.ini file.
Handle Files Like (Option 2)	This field is similar to **Handle Files Like (Option 1)**, but this allows you to specify a different filter for output documents.
E-mail To (Option 2)	This field is similar to **E-mail To (Option 1)**, but this allows you to specify a different recipient e-mail.
E-mail From (Option 2)	This field is similar to **E-mail From (Option 1)**, but this allows you to specify a different sender e-mail.

Input Document Handling (Using Files) island

Field	Description
Input Files Like (Option 1)	If your background task runs an action that requires document input, use this field to specify the filter for the files that Maconomy should use for the action.

Field	Description
	Use simple wild-card notation. All files that match your specified filter (and are found at the specified file location) will be used as input. If you do not specify a value, no files are filtered and used as input.
Input File Location Ref. (Option 1)	In this field, specify a named reference to the file directory from which Maconomy should get the input files. To configure the actual directory, use the server.ini file on the Maconomy server. For example, you can specify different input file loacations for production and test systems. When data is copied from the production system into the test system, the two systems carry out similar background tasks differently by picking up input files from different locations.
Input File Location (Option 1)	In this field, specify the base-file location (for example, a file server) where Maconomy should search for input files.
Store Output Files Like (Option 1)	If the background task using input files also generates output files, you can choose to store both file types in the same directory. Use this field to specify the filter that Maconomy should use for the output files. Use simple wild-card notation. For example: Enter the value ***.log** to match all files that end in ".log". These filtered output files are then stored in the same directory as the input files used for the task.

Remarks island

Field	Description
Remarks	You can use this field to enter remarks about the task.

Action	Description
New Background Task	Click this action to start creating a background task.
Delete Background Task	After selecting a background task from the filter list, click this action to delete that task. Note that you cannot delete running tasks or pending system tasks.
Deactivate	After selecting a background task from the filter list, click this action to deactivate that task. This prevents the system from picking up the task for execution. You can then edit the deactivated task.

Action	Description
Activate	After selecting a background task from the filter list, click this action to activate that task. Once you activate a task, the system may pick it up for execution. You cannot edit active tasks. However, you can edit the **Remarks** field of an active background task if you navigate to **Background Tasks workspace » Status section » Background Task tab**.
Copy as Template	After selecting a background task from the filter list, click this action to use the selected task as a template for a new task. The new task is marked as deactivated, and is therefore editable.

Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab

In this sub-tab, enter information about parameters and user-changes that are associated with the action of the background task. Each line represents either a field edit or a parameter with an associated value. For example, if the background task represents an update to a record, the sub-tab indicates the fields to be updated. Another example is when the action uses parameters to indicate e-mail settings and the like; the parameter values are stated in the sub-tab.

If you specify user-input parameters for actions that do not take user-changes, this means that the system should perform an update (with those changes) before performing the actual action. For example, before the system performs the **Copy Budget** action in the Job Budgets container, it may be necessary to set the source budget type, and so on. This is done by an update, and can therefore be specified by providing user-inputs for such an action.

Field	Description
Line Type	In this field, select one of three available values: ▪ **Parameter** – The line represents a parameter used by the action. ▪ **User Input** – The line represents a field that is edited as part of a **Create** or **Update** action. ▪ **Read Parameter** – The line represents a parameter passed to the READ event of the container, which takes place before performing the action. The default value of this field is **User Input**.
Field/Parameter Name	In this field, enter the name of the field/parameter that the line represents.
Type	In this field, specify the field/parameter type. For example, the field type of “JobNumber” in the Time Sheets container (sub-tab) is **String**, while the field type of “PermanentLine” in the same container is **Boolean**.

Field	Description
Expr.	Select this check box if the field value is entered as an expression. Expression values are resolved at execution time and may depend on the data in the current record. If you do not select this check box, the field value must represent a constant value of the appropriate type.
Value	In this field, enter the value of the field/parameter. This is independent of the actual type. The value you enter should be a String representation of the actual value. For example: Use **true** or **false** values for Booleans, and enter Strings as is. If you select the **Expr.** check box, enter a value as an expression with the result type you specified in the **Type** field. If the value you enter represents an expression, you may refer to fields in the record (that is, a record in the pane related to the background task) by their names. If that pane is of the Table type, you can refer to values of the fields in the corresponding Card pane by prefixing “card” to the Card field names. If you need to specify a value for each type, you can add the fields for these to the default layout.

Action	Description
Add Parameter/Field	Click this action to add a line for a parameter/field.
Insert Parameter/Field	Click this action to add a line for a parameter/field between existing lines.
Delete Parameter/Field	After selecting a parameter/field line, click this action to delete that line.
Move Parameter/Field Up	After selecting a parameter/field line, click this action to move that line up.
Move Parameter/Field Down	After selecting a parameter/field line, click this action to move that line down.

Setup » Background Tasks workspace » Setup section » List of Key Generators tab

Action	Description
New Key Generator	Click this action to create a key generator.

Setup » Background Tasks workspace » Setup section » List of Key Generators tab » Key Generator sub-tab

Key Generator island

Field	Description
Name	In this field, enter the name of the key generator.
Description	In this field, you can enter an optional description for the key generator.
Container Name	In this field, specify the name of the container which will provide key information. This container must contain a filter list. Each record read through this container's filter list corresponds to one container key. When the system reads these records, it applies the restrictions specified in the **Extracted Rows Match This Condition** field.
Pane Name	In this field, specify the name of a pane in the container. Since the pane type is always "Filter", the default pane name is generally "Filter" as well. In case the the filter pane is named differently, specify that name here.

Extract Fields island

Field	Description
Extract Field Name 1-10	In these fields, you can specify up to 10 field names for extraction from the filter that the key generator addresses. Maconomy uses the extracted field/s to map to container keys, record conditions, and/or run-as information in background task generators associated with this key generator.

Extracted Rows Match This Condition island

Field	Description
Extracted Rows Match This Condition	In this field, specify a Boolean expression for Maconomy to apply when reading keys. This is a required field. If you want all rows to match the condition, enter the value **true**. You may use simple functions such as **userDate()** (to access the current date), **addDays()**, and so on. However, you should only specify functions that can resolve a value independently of data. This means you cannot apply functions to field values in the container. For example: **addDays(SomeField, 2)** is not valid, while **addDays(userDate(), 2)** is valid since it can be computed without reference to any field values from the underlying container. You can also refer to field values directly (that is, not using function invocations). For example, you can refer to time sheets that are submitted but not yet approved by specifying the following condition: `submitted and not approved`.

Show Fields During Test island

Field	Description
Field Names	In these fields, you can specify up to 10 additional field values for extraction during testing. While these fields are optional, they are useful for providing the key generation with a meaningful context. For example: If you have a key generator that extracts information about customers with certain properties, you will most likely specify **CustomerNumber** in one of the fields in the Extract Fields island. While you may not need to extract actual customer names, extracting names (and other fields) helps you verify that your test is working.

Remarks island

Field	Description
Remarks	You can use this field to enter remarks about the key generator.

User island

Field	Description
Created	This field displays the date the key generator was created, as well as the name of the user who created it.
Changed	This field displays the date the key generator was last updated, as well as the name of the user whom made these updates.
Version No.	This field displays the number of times the key generator has need edited.

Action	Description
New Key Generator	Click this action to create a key generator.
Delete Key Generator	After selecting a key generator from the filter list, click this action to delete that key generator. If you select and try to delete a key generator currently in use, Maconomy displays an error message.
Test Key Generation	Click this action to generate an Excel file containing the following information: ▪ All keys generated by the rule at the time you run the action ▪ Data extracted from fields specified in the Extracted Fields island

Action	Description
	▪ Data extracted from fields specified in the Show Fields During Test island ▪ When running this action, you can limit the number of rows extracted. For example, you can limit the output to show only the first 500 rows extracted.

Setup » Background Tasks workspace » Setup section » List of Schedule Rules tab

Action	Description
New Background Task Generator	Click this action to create a background task generator.
Import	Click this action to launch the wizard that allows you to import key generators and background task generators using the Import Background Task Generators import program.

Setup » Background Tasks workspace » Setup section » List of Schedule Rules tab » Generate Default Rules sliding panel

If the default rules are deleted or heavily modified, the system administrator may lose track of the rules' original specifications. In such instances, use this sliding panel to regenerate and revert to the standard rules.

Field	Description
Title	This field displays the name of the default rule corresponding to the current line.
Description	This field displays a brief description of the default rule.
Marked for Gen.	If you select this check box and then click **Generate Marked Default Rules**, Maconomy generates the rule represented by the line.
New Key Generator Name	In this field, enter the name of the key generator that is used for the rule. This name should not match that of any current key generators.
New Task Generator Name	In this field, enter the name of the task generator. This name should not match that of any current task generators.
Generated By	This field displays the name of the user who last invoked rule generation from the line.
Generated On	This field displays the date when the user last invoked rule generation from the line.

Action	Description
Mark All	Click this action to select the **Marked for Gen.** check box on all lines.
Unmark All	Click this action to clear the **Marked for Gen.** check box on all lines.
Generate Marked Default Rules	Click this action to generate task generators and corresponding key generators for all the lines that are selected for generation. After clicking this action, Maconomy clears the **Marked for Gen.** check box on all lines. Maconomy does not automatically activate the new task generators. The designated user or system administrator has to activate them manually.

Setup » Background Tasks workspace » Setup section » List of Schedule Rules tab » Schedule Rule sub-tab

Generator island

Field	Description
Name	This field displays the name of the task generator, or the name you entered in the New Schedule Rule wizard.
Description	In this field, you can enter a description of the task generator.
Access Level	In this field, specify an access level. Maconomy applies this access level to all generated tasks.
Type	In this field, specify a task generator type. Select one of the following values from the drop-down list: ▪ **Single Task** – The task generator generates a single task type. Such task generators specify an action that is carried out in a specific container, along with a trigger time pattern that dictates when this action is performed. ▪ This is the default value. ▪ **Sub Task** – The task generator generates a single task type. However, you do not specify a trigger time pattern, since sub tasks are meant to be referenced by sequential and interleaved tasks.

- **Sequential Tasks** – The task generator serves as an umbrella for other task generators (of the Sub Task type) which are invoked when this generator is triggered.
  - Maconomy generates the corresponding tasks such that each task type is carried out in sequential order (that is: Maconomy generates all tasks related to task generator 1, then all tasks related to task generator 2, and so on). This means Maconomy does not generate tasks related to generator 2 until after it completes all tasks related to generator 1.
  - **Interleaved Tasks** – This is similar to sequential tasks, but Maconomy carries out the tasks in an interleaved manner rather than in a sequential one. That is, Maconomy:
    1. Invokes generator 1 and carries out its first task
    2. Invokes generator 2 and carries out all resulting tasks
    3. Carries out generator 1's second task
    4. Invokes generator 2 and carries out all resulting tasks, and so on.
  - **Note:** Interleaved tasks have a greater overhead than sequential tasks, and may lead to situations where the same operation is produced multiple times. **Therefore, unless you have very specific reasons for using interleaved task execution, consider using sequential task execution whenever possible.**
  - In addition, interleaving of tasks is not strict. Several background processes can work in parallel. That is, Maconomy can execute several tasks from the generator 1 before carrying out subsequent tasks from generator 2.

Trigger Time Pattern

In this field, specify the time pattern that Maconomy should use for triggering the task generator. The syntax for this pattern is similar to that of cron jobs.

Some examples:

`10 * * * *` triggers the task on the 10th minute of each hour daily. That is: 00:10, 01:10, 02:10, 03:10, and so on.

`*/15 * * * *` triggers the task on the 15th minute of each hour daily. That is: 00:00, 00:15, 00:30, 00:45, 01:00, 01:15, and so on.

`20 15 * * *` triggers the task at 20 minutes past 15:00 daily. That is: 3:20 P.M.

`20 15 * * Tue` triggers the task at 20 minutes past 15:00 on Tuesdays. That is: 3:20 P.M. on Tuesdays.

	`20 15 * * Mon-Fri` triggers the task at 20 minutes past 15:00 on weekdays. That is: 3:20 P.M. on Mondays through Fridays. **`*/15 9-17 * * *`** triggers the task every 15 minutes from 9:00 to 17:00 daily. That is: 09:00 A.M., 09:15, 09:30, 09:45, 10:00, 10:15, … 5:45 P.M. **`*/15 9-10,16-17 * * *`** triggers the task every 15 minutes from 9:00 to 10:00 and from 16:00 to 17:00 daily. That is: 09:00 A.M., 09:15, 09:30, 09:45, 10:00, 10:15, … 10:45, … 4:00 P.M., 4:15, …5:45 P.M. **`0 6 1 * *`** triggers the task at 6:00 A.M. on the *first* day of each month. **`0 23 L * *`** triggers the task at 23:00 on the *last* day of each month. That is: 11:00 P.M. on January 31, February 28 (or 29 during a leap year), March 31, April 30, …December 31. **`0 8 * * * | 0 18 * * *`** triggers the task at 8:00 A.M. *and* 18:00 (6:00 P.M.) daily. Patterns can be separated by a |. Leave this field blank only when the schedule rule is of the Sub Task type. For more information about trigger time patterns, refer to http://www.sauronsoftware.it/projects/cron4j/manual.php#p02.
Last Run	This field displays the date and time that Maconomy last ran the task generator.
Next Due	This field displays the date and time this task generator is next due. This field takes the specified Trigger Time Pattern into account.
Valid	In these fields, you can specify a validity date interval. If you fill out both fields, Maconomy only invokes the task generator within this date range (regardless of the specified Trigger Time Pattern). If you leave the first field blank, this means you do not want to specify a start date. If you leave the second field blank, this means you do not want to specify an end date.
Run a Max. Number of Times	Select this check box if you want Maconomy to run the schedule rule a maximum number of times. By default, this check box is not selected.
No. Runs Left	When you select the **Run a Max. Number of Times** check box, Maconomy automatically displays this field. Here you can specify the maximum number of times Maconomy should run the schedule rule. Every time Maconomy triggers the rule, this number is decreased.

Background Task island

The fields in this island are only applicable to Single Task and Sub Task types.

Field	Description
Container Name	In this field, specify the internal name of the container that hosts the task to be carried out. For example: “maconomy:TimeSheets” or simply “TimeSheets”, since the namespace “maconomy” is the default value.
Singleton Container	This field indicates whether the specified container is a singleton container (or one that cannot be referenced by a specific container key).
Pane Type	In this field, select the type of pane where Maconomy invokes the background task. If you fill out this field, you can leave the **Pane** field blank. Maconomy then automatically fills out that field with the default name for the pane type you specify here.
Pane	This field shows the internal name of the pane that hosts the task to be carried out. This field is not displayed by default, but you can add it to a custom layout.
Action Name	In this field, specify the internal name of the named action that the background task is meant to carry out. If the action is a Standard action, leave this field blank.
Standard Action	In this field, select the standard action that the background task is meant to carry out. You can select one of the following values from the drop-down list: ▪ Create ▪ Update ▪ Delete ▪ Print If the action is a named action, leave this field blank.
System Nature	This field is similar to the **System Nature** field found in the Task Properties island of the

	Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**).
Max. Duration	This field is similar to the **Max. Duration** field found in the Task Properties island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**).
Tasks Require Execution	Select this check box if you want Maconomy to consider it an error whenever an action cannot be carried out on a record. This applies in the case of an unfulfilled record condition or when you select the **Ignore Disabled Action** check box. By default, this check box is not selected.

Container Keys island

Field	Description
Container-Key Generator	Use these fields to specify the generator you want to use for generating container keys and additional information. You can specify either the name or the number of the container key generator. When the task generator is triggered, Maconomy runs this container key generator. For each of the generated entries, Maconomy also generates a task with the properties specified in the Background Task island (depending on how you define the mapping to container key fields).

The following fields in this island are structured as a “grid” composed of up to 10 rows: one for each of the fields extracted by the specified Container-Key Generator. If only one field is extracted, only one row is displayed; if two fields are extracted, two rows are shown; and so on. While all the Source Field names extracted by the Key Generator are listed in these rows, you are not required to use all of them for mapping.

Field	Description
Context Field Name	This field displays the value of the extracted field.
Run As	This field displays additional information about the value of the **Context Field Name** field. Specifically, this field displays one of the following values:

- **<Blank>** - The **Context Field Name** field does not indicate the user on whose behalf Maconomy runs the task. The system runs the task on behalf of whoever is specified as the background user. Ideally, this background user should have Administrator rights. Otherwise, “Run As” information is disregarded, and Maconomy runs all generated tasks as the background task administrator user (specified in the server.ini file).
- **User** - The value of the **Context Field Name** field represents a user name. The system runs the generated background task on behalf of this user.
- **Employee Number** - The value of the **Context Field Name** field represents an employee number. The system runs the generated background task on behalf of the user (or one of the users) associated with this employee number.
- **Role Instance** – The value of the **Context Field Name** field represents the internal identification (instance key) of the user role on whose behalf this task should be run.

Only one row should have a value for this field. If you try to fill out this field in another row, Maconomy deletes the first value you entered.

Record Condition island

Field	Description
Record Condition	In this field, enter an expression for Maconomy to apply as a record condition to each of the generated tasks. For example: You specify a rule for approving all submitted time sheets. If a user approves a submitted time sheet after the system generates the background task and before the system picks up this same task for execution, you could receive an error report. This is because Maconomy cannot approve a time sheet that is already approved. To avoid getting an error, you could specify a record condition of “Submitted and Not Approved”. If you do this, the system picks up the task for execution and carries out the approval action only for time sheets that are submitted but not yet approved.

	The record condition you specify here can also refer to context field names (for example, to specify actions that should take place on specific lines in a sub-tab). In this case, refer to the value of the context field names using the context.-prefix indicated in the **Context Field Name** field for the row.
Ignore Disabled Action	This check box is similar to the **Ignore Disabled Action** check box found in the Record Conditions island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**).

Output Document Handling (Using Files) island

Field	Description
Handle Files Like (Option 1)	This field is similar to the **Handle Files Like (Option 1)** field in the Output Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
File Location Ref. (Option 1)	This field is similar to the **File Location Ref. (Option 1)** field in the Output Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
File Location (Option 1)	This field is similar to the **File Location (Option 1)** field in the Output Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.

Handle Files Like (Option 2)	This field is similar to **Handle Files Like (Option 1)**, but this allows you to specify a different filter for output documents.
File Location Ref. (Option 2)	This field is similar to **File Location Ref. (Option 1)**, but this allows you to specify a named reference to a different file directory.
File Location (Option 2)	This field is similar to **File Location (Option 1)**, but this allows you to specify a different base-file location.

Output Document Handling (Using Email) island

Field	Description
Handle Files Like (Option 1)	This field is similar to the **Handle Files Like (Option 1)** field in the Output Document Handling (Using Email) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
E-mail To (Option 1)	This field is similar to the **Email To (Option 1)** field in the Output Document Handling (Using Email) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
E-mail From (Option 1)	This field is similar to the **Email From (Option 1)** field in the Output Document Handling (Using Email) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
Handle Files Like (Option 2)	This field is similar to **Handle Files Like (Option 1)**, but this allows you to specify a different filter for output documents.
E-mail To (Option 2)	This field is similar to **E-mail To (Option 1)**, but this allows you to specify a different recipient e-mail.
E-mail From (Option 2)	This field is similar to **E-mail From (Option 1)**, but this allows you to specify a different sender e-mail.

Input Document Handling (Using Files) island

Field	Description
Input Files Like (Option 1)	This field is similar to the **Input Files Like (Option 1)** field in the Input Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
Input File Location Ref. (Option 1)	This field is similar to the **Input File Location Ref. (Option 1)** field in the Input Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
Input File Location (Option 1)	This field is similar to the **Input File Location (Option 1)** field in the Input Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.
Store Output Files Like (Option 1)	This field is similar to the **Store Output Files Like (Option 1)** field in the Input Document Handling (Using Files) island of the Background Task tab (**Setup » Background Tasks workspace » Editing section » Background Task tab**). The system copies this information to each background task generated by the task generator.

Remarks island

Field	Description
Remarks	You can use this field to enter remarks about the task generator.

User island

Field	Description
Created	This field displays the date the background task generator was created, as well as the name of the user who created it.
Changed	This field displays the date the background task generator was last updated, as well as the name of the user who made these updates.

Field	Description
Version	This field displays the number of times the background task generator has been edited.

Action	Description
New Background Task Generator	Click this action to start creating a background task generator.
Delete Background Task Generator	After selecting a background task generator from the filter list, click this action to delete that task generator. Note that you cannot delete a task generator if other task generators refer to it.
Activate	After selecting a background task generator from the filter list, click this action to activate that task generator. Once you activate a task generator, the system may pick it up for execution within the validity period you specified. Note that you cannot edit active task generators. If you activate a rule, you also activate all the sub tasks that the rule references.
Deactivate	After selecting a background task generator from the filter list, click this action to deactivate that task generator. This prevents the system from invoking the task generator. You can then edit the deactivated task generator. Note that deactivating a sub task also deactivates any rule that references that sub task.
Run Generator	Click this action to start generating tasks manually. This action only generates tasks if: ▪ The background task generator has just passed its “next due date/time”. ▪ The background task executor is within its validity interval. ▪ The **Run a Max. Number of Times** check box is selected, and the **No. Runs Left** field’s value is greater than zero. Note that you usually should not manually invoke this action. The background task execution engine automatically invokes it.

Setup » Background Tasks workspace » Setup section » List of Schedule Rules tab » Schedule Rule sub-tab » Test Bed sliding panel

Generator island

Field	Description
Name	This field displays the name of the background task generator.
Description	This field displays a description of the background task generator.
Access Level	This field displays the access level specified for the background task generator.
Type	This field displays the background task generator type. It could have one of the following values: ▪ **Single Task** ▪ **Sub Task** ▪ **Sequential Tasks** ▪ **Interleaved Tasks**
Trigger Time Pattern	This field displays the trigger time pattern specified for the background task generator.
Last Run	This field displays the date and time Maconomy last ran this task generator.
Next Due Date	This field displays the date this task generator is next due. This field takes the specified Trigger Time Pattern into account.
Next Due Time	This field displays the time this task generator is next due. This field takes the specified Trigger Time Pattern into account.
Valid	This field displays the validity date interval specified for the background task generator.

Action	Description
Run Generator Unconditionally	This is similar to the **Run Generator** action in the Schedule Rule sub-tab, except clicking this action generates tasks unconditionally (that is: independent of whether the task generator is due/is within the validity period/has been run the maximum number of times).

Setup » Background Tasks workspace » Setup section » List of Schedule Rules tab » Schedule Rule sub-tab » Parameters panel

Field	Description
Line Type	This field is similar to the **Line Type** field found in **Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab**.
Field/Parameter Name	This field is similar to the **Field/Parameter Name** field found in **Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab**.
Type	This field is similar to the **Type** field found in **Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab**.
Expr.	This check box is similar to the **Expr.** field found in **Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab**.
Value	This field is similar to the **Value** field found in **Setup » Background Tasks workspace » Editing section » Background Task tab » Parameter/Field sub-tab**.

Action	Description
Add Parameter/Field	Click this action to add a line for a parameter/field.
Insert Parameter/Field	Click this action to add a line for a parameter/field between existing lines.
Delete Parameter/Field	After selecting a parameter/field line, click this action to delete that line.
Move Parameter/Field Up	After selecting a parameter/field line, click this action to move that line up.
Move Parameter/Field Down	After selecting a parameter/field line, click this action to move that line down.

Actions

Note: The information in this section is in DRAFT form and is not yet updated to reflect 2.5.2.

Show Next Scheduled Rule

In the Test Bed assistant, used for testing scheduled rules, the **Show Next Scheduled Rule** action shows the next fifty (50) execution times for the current rule. This action is beneficial for debugging, and is used to verify that the time pattern entered for the rule results in the wanted executions. For ease of use, you can also run / test the rule prior to activating the rule.

Note: Take into consideration the data in the **No. of Runs Left** field and the valid interval when evaluating execution times.

Export Selected Lines

This action creates an Excel export of the background tasks specified in the selection criteria of the tab. This is an alternative to the **Show Lines** action and works for more than 5000 lines.

Test Schedule Rule

Click on this action and a new wizard displays enabling you to test the schedule rule

Export Selected Tasks

This action exports to Excel all the tasks with the Selection Criteria that you've selected. Leave the Selection Criteria fields blank to export everything in the database. Alternatively, filter by container or origin or any Selection Criteria field to narrow your list. This action allows you to verify lines before you delete them.

Delete Selected Tasks

Use this action to delete selected tasks. You no longer need to select "Show Lines." This is helpful when you have numerous tasks (sometimes more than 5,000). Previously, you wouldn't be able to delete from this window. With the new updates, you can delete from this window. If you do not specify any criteria, then the action acts like a "delete all lines" action and will delete all lines.

Export Selected Tasks

Use the **Export Selected Tasks** action and to export and review material prior to deletion.

Pause Execution

The **Pause Execution** action facilitates a smooth break in background task execution so that any background tasks currently running can successfully complete and that no further tasks are picked up. The Pause action is used on nodes only, and it sets all the execution threads underneath this node to a state where they suspend picking up new background tasks for execution. The threads complete background tasks already in progress, but do not pick up additional background tasks.

Using this action enables current tasks to complete without failure then gives you time to perform whatever action you must do, such as shutting down a coupling service. When you resume activity, such as after a coupling service restart, the node is no longer "paused," and it remembers where it left off and resumes background tasks from that point.

Note: This action impacts the selected node only. In the instance that there are several coupling services, the other coupling services can continue to pick up new background tasks.

Delete Inactive Test Tasks

If you have tested and found that everything is okay, and you don't want to activate the tasks, you can instead delete them.

Activate Inactive Test Tasks

Generally, activating inactive / generated test tasks is used as a second phase of testing. For the first phase, generate the test tasks and investigate them. To perform further testing, use this action to activate them again and evaluate if they are being executed correctly.

Show Next Scheduled Rule Execution

Click to show the next 50 scheduled runs so you can check the time pattern as needed.

Framework Admin and Maintenance

Monitoring Background Tasks

Maconomy provides numerous ways to monitor background tasks, including:

- Email reporting
- Status information in the Background Tasks workspace
- Color coding

Email Reporting

An automatic reporting system provides transparency into the system, enabling system owners to have better insight into and control over what is occurring in the system. Upon startup or after a task is resolved, an auto-email reports to confirm status. If an error occurs, the email report immediately flags it along with diagnostics and tips on resolution to facilitate user self-sufficiency.

For example, an email report is sent when the system is put in maintenance mode, or if background task execution is switched off.

The email report is supported by a new sliding panel within the Background Task workspace in case email is unavailable for whatever reason.

The report includes the following information:

- Server information, including name in heading and detail at the bottom of email
- Summary of error
- Tips on resolving the error
- Name of background task administrator

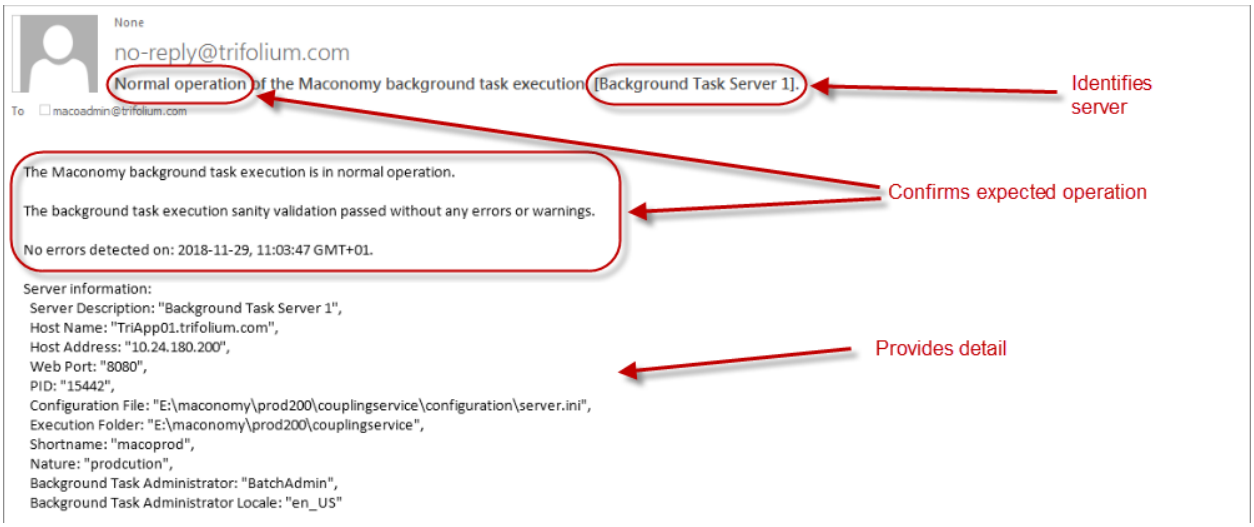
When errors are unresolved, a reminder email is sent every two hours until the issue is resolved. When the issue is resolved, an email report is sent confirming a return to normal operations.

Note: By default the email timing is every two hours. To configure the timing, see Configure Email Timing.

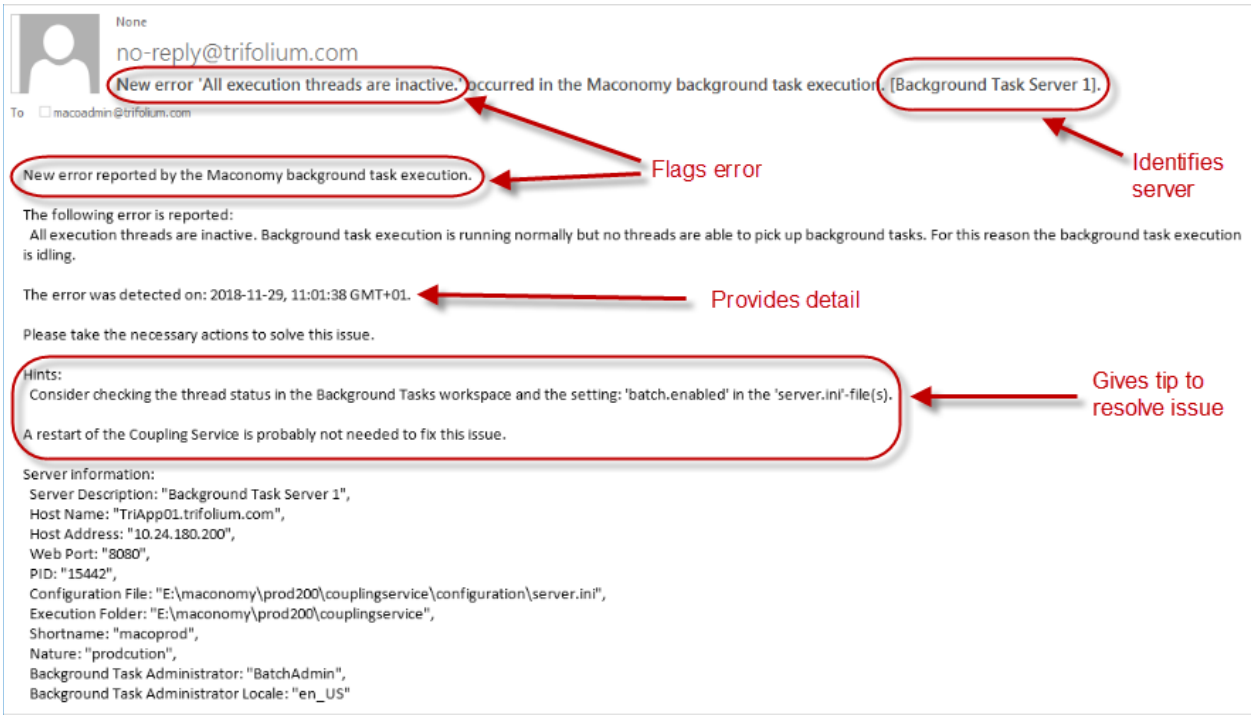
Email reports are sent automatically:

- Upon startup.
- If an error prevents background tasks from being executed.
- When the system goes back into to confirm normal operations, after an error has occurred.
- When the system is put into maintenance mode.
- Each hour (or as configured), to flag any unresolved issues.

Email Report—Normal Operation



Email Report—Task Error



Sliding Panel for Error Reporting

Background Tasks » Management » Execution Runtime » Execution Nodes » Coupling Service sliding panel

If an error occurs with background tasks that cannot be reported via the email reporting system, such as due to an invalid/non-responding e-mail server, an incorrect email recipient, or network issues, these errors are reported in the Coupling Service sliding panel.

The Coupling Service sliding panel supports the email reporting system, and provides information regarding issues detected while attempting to send email reports. If an error occurs and a message is available on the sliding panel, a red message icon displays in the Msg column on the Execution Nodes sub-tab for each coupling service impacted (node of type PID). This indicates you should check the sliding panel for information.

Error Types

There are various types of errors that may occur and impact background task performance. Below are the general types. In all cases, refer to the email report for details.

- **User Access** — Some errors, such as an unknown user specified in batch.administrator.loginname or insufficient access for that user, make it impossible for the system to access the background tasks in Maconomy. These errors are sent by email.
- **Coupling Service** — Coupling service errors relate to one specific coupling service that is unable to execute background tasks. These errors are sent by email.
- **System** — Errors may relate to the entire SYSTEM as unable to execute background tasks, or indicate that the entire SYSTEM is “hanging.” These errors are sent by email.
- **Monitoring** — Some errors, such as sanity, configuration, and runtime error messages, relate to monitoring whether it is possible to send error reporting emails at all. If email reporting is unavailable, such as due to an invalid email address, the messages are displayed in the Coupling Service sliding panel. A red message icon displays in the Msg column on the Execution Nodes sub-tab for each coupling service impacted (node of type PID). These errors are reported in the Coupling Service sliding panel. These errors are reported in the Coupling Service sliding panel.

Status Information in Background Task Workspace

Background Tasks > Management > Execution Runtime

The Execution Runtime tab provides quick access to detail regarding your background tasks. The view enables you to quickly see status as well as errors or “hanging” tasks. Additionally, color coding indicates the status at-a-glance.

The workspace includes these areas:

- **Status**—This section tells you whether or not the background task execution is on or not, as well as:
  - **No. Running Tasks**— This field displays the number of currently running background tasks. Click the **Refresh** action to monitor how the number changes over time.
  - **Last Refreshed**—Indicates the last time this view (including the list of execution nodes shown in the table part) was last refreshed. This works in conjunction with the “Execution Runtime” part of the workspace, which offers a snapshot of what the system looks like at a given time. Often the system will look slightly differently even a few seconds after the snapshot. The “Last Refreshed” details when this snapshot was taken.
- **Hanging Tasks**—This section flags if a background task is running indefinitely (in other words, “hanging”). If hanging tasks are detected, an error message displays in purple in this section, and the number of hanging tasks detected displays in red. Additionally, an error message displays in red, and an email report is sent to flag the error. If all is running as expected, the message “No hanging tasks” displays.
- **Error Messages**—This section includes details of error messages, if any. If not, the message “No error message” displays.

Actions on Execution Nodes Sub-Tab

Background Tasks > Management > Execution Runtime

- **Pause Execution Action**— Use the Pause action to bring the threads of a node into a state where they complete existing tasks and will not pick up new tasks. This is helpful when you need to shut down a coupling service, as once all threads have finished executing their current tasks, you know that they cannot take new ones, so then it is a safe time to shut down a coupling service. When the coupling service is started once again, the threads on the node resume picking up new tasks.
- **Stop Execution Action**—When you use this action, the node (and underlying threads) are no longer able to pick up *new* background tasks, but when the coupling service is stopped and started once again, these nodes will *remain* in the Stopped state.
- By default, the layout of the workspace allows the user to invoke the "Pause" actions for coupling-service nodes (type PID) and "Stop" for nodes of type Host and System. By default none of these actions are available for the individual threads, but can be configured by updating the layout.
- **Termination status**—A "Termination in progress" status on the Execution Nodes sub-tab enables you to easily see when a background task is in the process of being cancelled.

Color Coding

The color coding uses conventions of text color and text highlight to communicate several messages at once and at-a-glance. Threads (and the parent nodes of threads) may be color-coded using highlight coloring, as described below.

Note: The color codes can be configured in the layout, per usual process for layout changes.

Within the Background Task workspace, click the sliding panel to see the color legend while in Maconomy.

Color	Indicates
For background tasks that may run, text is colored and indicates:	
Black text	All is as expected and these background tasks may run.
Blue text	Background task is currently being run on the thread shown.
Red text	A background task has been cancelled (either by the end-user or by the system), and the task termination is in progress.
Purple text	Background task is hanging / running indefinitely.
For background tasks that will NOT run, text is highlighted and indicates:	
Purple highlight	Purple indicates that *all* execution threads beneath this node are hanging. This is only used for nodes of type PID, Host, and System.
Yellow / orange highlight	Yellow/orange means that the node is prevented from picking up new background tasks because one of its parent nodes is stopped.
Red highlight	Red means that the given node is "stopped" by the end-user, which means that it is prevented from picking up *new* background tasks

	(currently-running tasks will continue as usual and their foreground color-code will reflect the state).
Black highlight	Black highlight with yellow text indicates Maintenance Mode / background tasks are OFF:
Light gray highlight	A bold light gray color for nodes of type PID indicates that the node has been deliberately excluded from executing background tasks.

Color Code Flow Chart

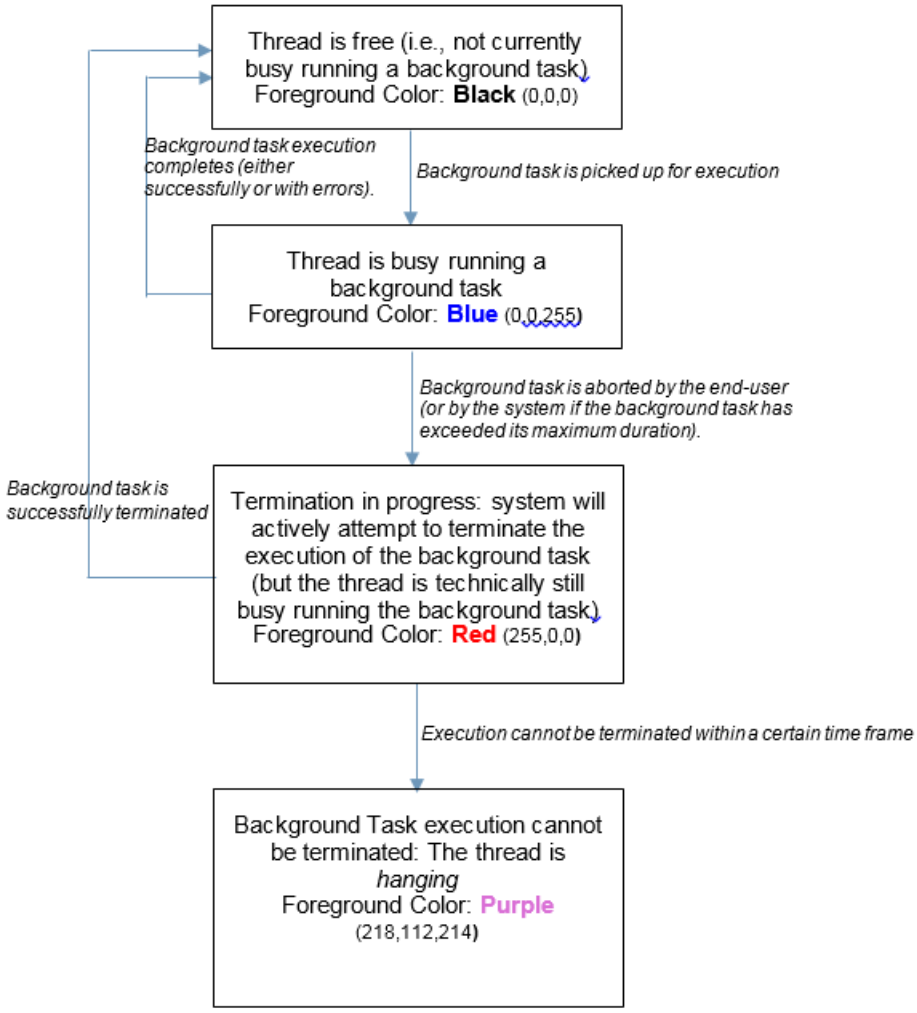


Figure 1: How nodes of type "Thread" make transitions and what the corresponding color coding means.

Verify and Delete Tasks

To verify and delete tasks:

1. Go to **Background Tasks > Management > Selection & Clean-Up.**
2. In the Selection Criteria tab, enter information in any fields you choose to use as task filters.
3. Click **Export Selected Tasks**.
4. Review the exported information.
5. When verified, in the Selection Criteria tab, again enter information for fields you choose to filter.
6. Click **Delete Selected Tasks**.

Clear “Override Auto-Deletion of Tasks”

With the new **Override Auto-Deletion of Tasks** field, you can set up system defaults to override auto-deletion, as well as specify dates on which tasks should be deleted. This is useful in testing and to improve monitoring.

Sometimes if you want to debug or test, it is helpful to keep tasks even if they have finished successfully. While you could deselect the Delete Automatically field on the task, since some tasks are created from within the system, you do not have control over them and cannot deselect auto-deletion for those tasks.

Instead, use the **Ignore Until** field. Here you can specify date and time of deletion, meaning that once task has finished successfully, it is not deleted automatically until the specified date, giving you time to investigate the task.

To Override Auto-Deletion of Tasks:

1. Go to **Background Tasks > Management > System Defaults**.
2. In the **Override Auto-Deletion of Tasks** group, **Ignore Until** field, enter the date and the time on which the tasks must be deleted.

   After this date and time, the system resumes auto-deletion of tasks.

Admin FAQs and Troubleshooting

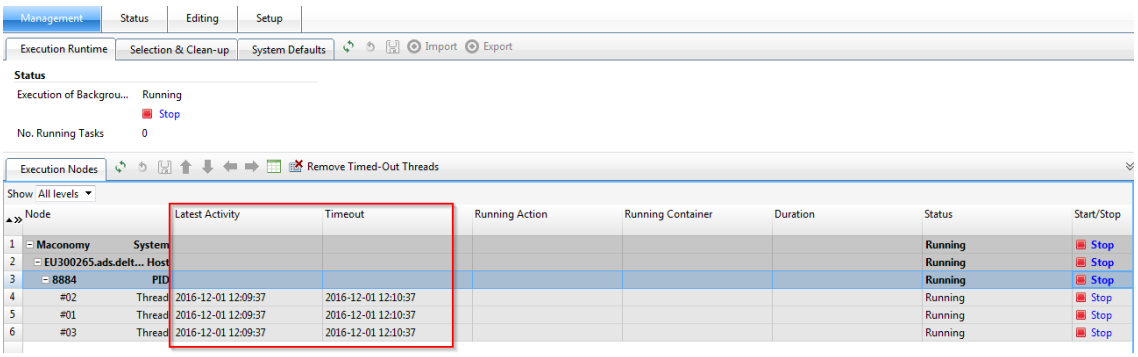
This section provides Frequently Asked Questions for Scheduled Background Tasks.

Is the Background Task System Running?

You need to check what background execution threads are running, and verify that they are “alive”.

To do that, go to **Set Up » Background Tasks » Management » Execution Runtime » Execution Nodes**.

This view shows all execution nodes “recently known” as well as their activity level:



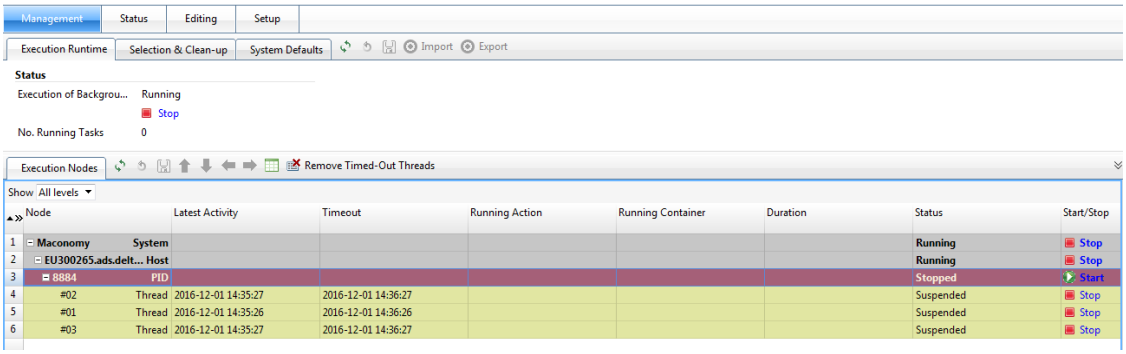
Each node of type “Tread” represents an execution thread that can run a task. The threads are grouped into running processes represented by the PID, grouped again under the host name of the computer hosting that process.

Every time a thread makes itself known (to query for background tasks) the “Latest Activity” time stamp is updated. Hence, using this field, you can see when a given thread was latest making itself known. The “Timeout” timestamp means that if the thread does not make itself known again before that time, the system may consider the thread as “timed out”, and may remove it from this view. If – at that time – the thread is believed to be running an actual background task, that background task will automatically be “aborted” by the system.

It is possible to “Stop” a given execution node. That means that no new background task will be assigned that node.

In the picture below, the node with PID 8884 is “stopped”, meaning that background tasks will not be assigned to any execution node belonging to this PID. The threads may still be rendered as “active” (their “Latest Activity” is updated), but no matter what background tasks will not be assigned to those nodes.

> **Note:** If all threads are Suspended/Stopped, no background tasks are processed!



I Need to Shut Down the Coupling Service – Should I Do Anything?

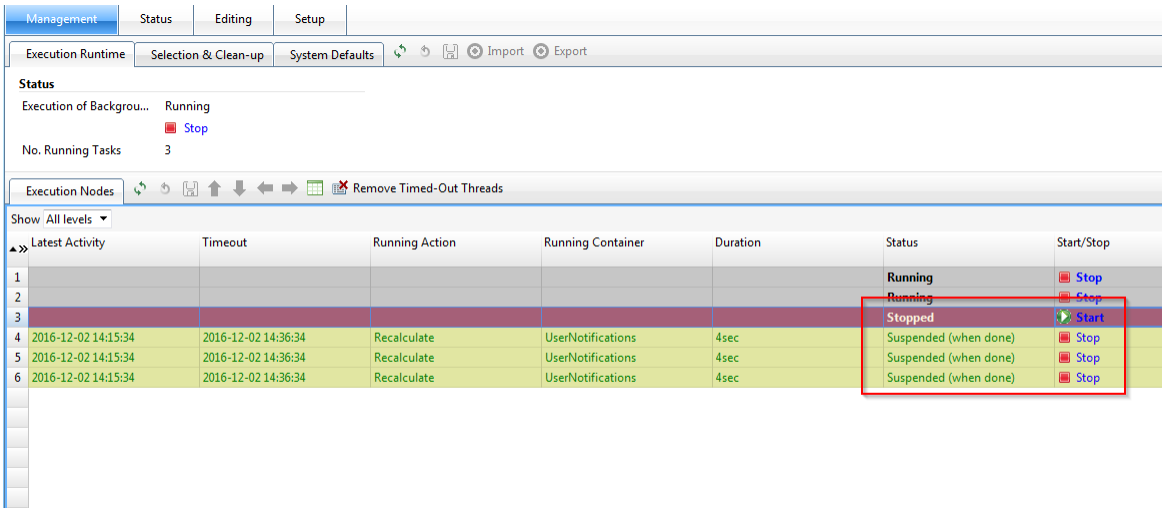
You don't have to do anything. But if the coupling service is executing a background task while you shut it down, that task will time out and be aborted by the system.

In order to avoid this situation, a "nice" behavior is to stop background task execution for the specific process ID, and once all underlying task threads are done executing tasks, stop the coupling service.

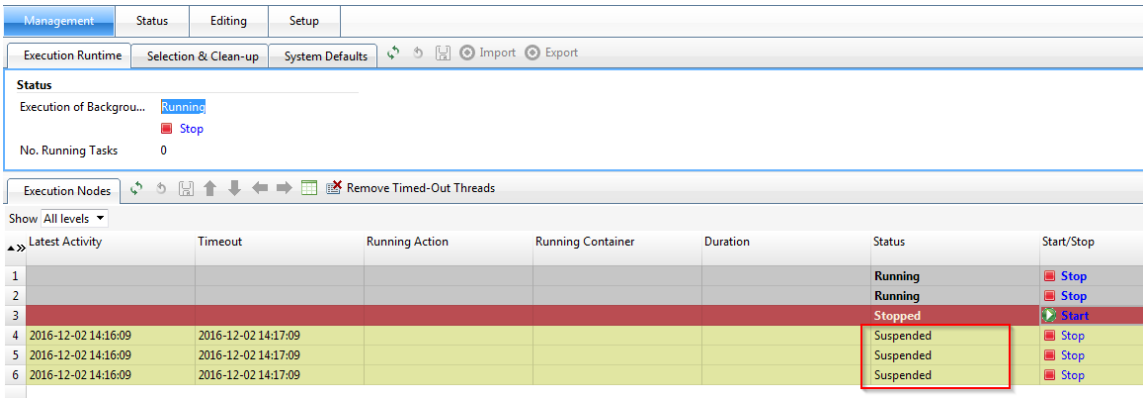
Note: you cannot get "damaged data" simply by shutting tasks down, but you may end up having tasks that are marked as failed, because they are aborted by the system. Such tasks can be rescheduled, but the decision to do so is a manual one.

You can stop execution of background tasks for specific processes in Set Up / Background Tasks, Management » Execution Runtime » Execution Nodes.

After pressing the "Stop" button for the process ID corresponding to the coupling service to be stopped, the view may change into something like (assuming that tasks were running):



By refreshing the view, you can see when the tasks are done:



Now that process can be stopped without leading to abortion of background tasks.

I Need to Copy the Production Database to Test – Can I Do So Safely?

When a database copy is done, the background tasks are also being copied (in their current state) since they are part of the data. For many tasks this is not a problem at all.

However, if some background tasks communicate with external systems (mail servers, file system, web services) it may be relevant to ensure that the production and test systems are configured to handle these external resources differently. And/or to ensure that background tasks that should only run on one of these systems are annotated with a proper “nature”.

Natures are defined in the server`.ini` file. As are named file- and URL references as well as mail servers. You should look into the documentation in the `server.ini` file for these properties.

If this has been set up properly, database copies can be done without issues.

Alternatively, you may choose to disable background task execution for the short name to which to database is copied, and manually examine the pending background tasks/background task schedule rules before enabling background task execution for such systems.

What is the Status of the Background Tasks?

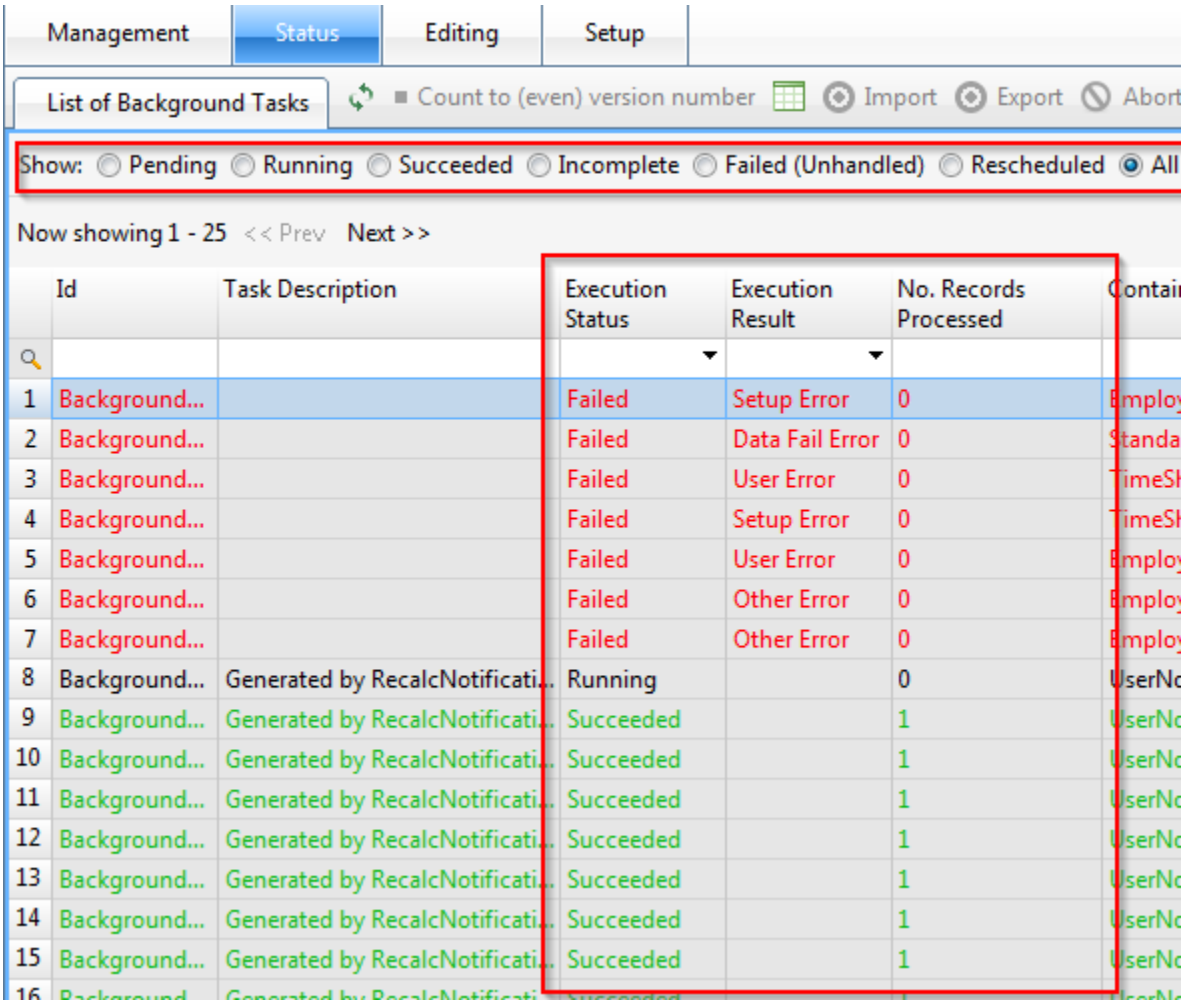
Background tasks can essentially be in one of the following states:

- Pending - The task is ready to be picked up for execution (when it’s active, it’s due time is up and there’s nothing to await)
- Running - The task is currently being executed
- Committed - The “logic” of the background task has succeeded and has been committed, handling of output documents has not been finalized yet
- Incomplete - After the task was “Committed” the handling of output documents failed for some reason. For example, it was not possible to send mails, or not possible to store files
- Succeeded - The task has succeeded: the “logic” has been done and committed, and all handling of output documents (if any) has been successfully completed.
- Failed - The execution of the background task failed. The overall reason “kind” can be seen in the background task filter, along with a more detailed description in the log associated with the task.

In case of a failed task, the Execution Result can have one of the following values:

- Aborted - The execution of the background task was aborted by a user, or automatically by the system because the background task timed out.
- User Error - This happens when the background task attempts to do something that does not make sense or is not allowed--just like an end user can experience an error message when entering data that is not accepted by the system.
- Internal Application Error - This happens in case of an internal error in the application. Situations like this should typically lead to support issues with customer care.
- Data Fail Error - This may happen if the background task targets a record that does not exist, or if the background task declares the creation of an entry that already exists. For example, if you try to update an employee that has been deleted, this kind of error may occur.
- Data Busy Error - This may happen if the data being targeted is in use or being modified by other users while the background task is running. Usually this error will trigger a re-attempt a while later. If this error keeps occurring, the task will fail with this error code at the end.
- Access Error - This may happen if the user executing the background task does not have access to the container in question.
- Setup Error - This may happen, for example, if the task has been set-up to submit a time sheet that is already submitted.
- Other Error - This may happen for other kinds of errors.

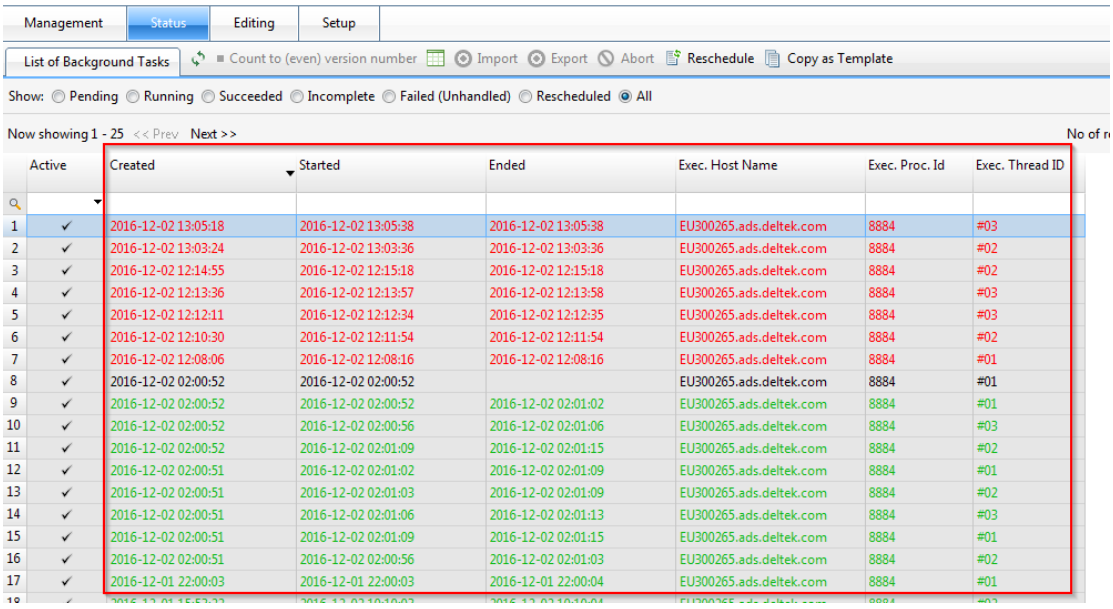
If you go to Set Up / Background Tasks, Status » List of Background Tasks there are many possibilities of overviewing tasks of various categories.



	Id	Task Description	Execution Status	Execution Result	No. Records Processed	Contai
1	Background...		Failed	Setup Error	0	Employ
2	Background...		Failed	Data Fail Error	0	Standa
3	Background...		Failed	User Error	0	TimeSh
4	Background...		Failed	Setup Error	0	TimeSh
5	Background...		Failed	User Error	0	Employ
6	Background...		Failed	Other Error	0	Employ
7	Background...		Failed	Other Error	0	Employ
8	Background...	Generated by RecalcNotificati...	Running		0	UserNo
9	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
10	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
11	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
12	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
13	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
14	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo
15	Background...	Generated by RecalcNotificati...	Succeeded		1	UserNo

In addition to the actual status, you can trace:

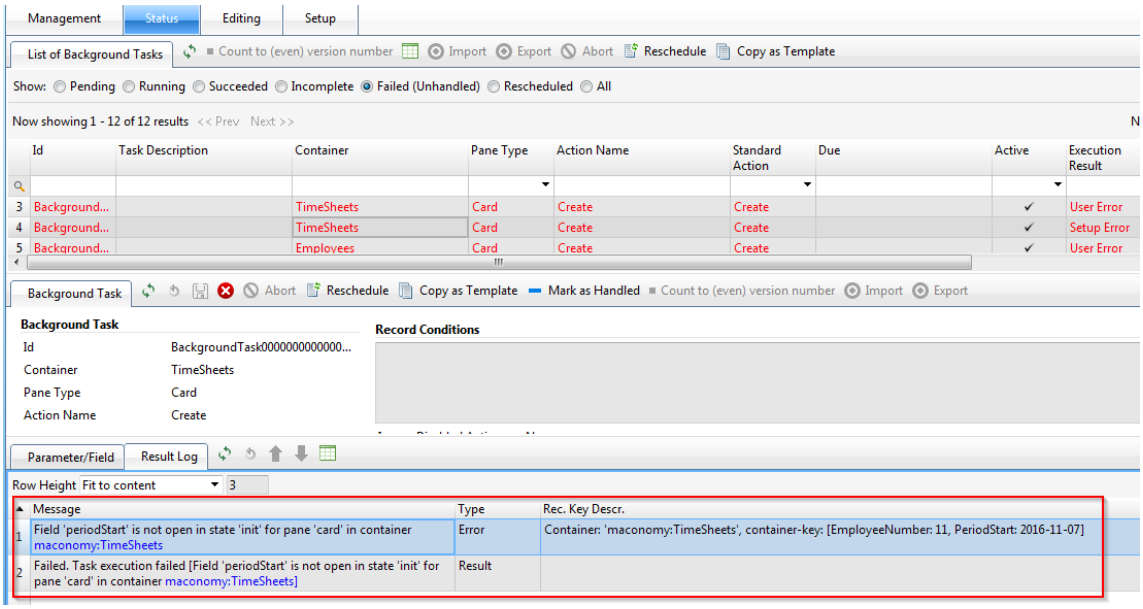
- When was the task created
- When was the task due
- When was the task started
- When was the task ended
- Which host machine / PID / Thread ID executed the task



Why Did a Background Task Fail?

A background task has failed. To understand why, examine the explanations provided in the Log of the background task. The log is found in: Set Up / Background Tasks, Status » List of Background Tasks » Background Task » Result Log.

In the case below, a time sheet was attempted created with the input: EmployeeNumber = '11' and PeriodStart = 2016-11-07, but the field PeriodStart is not open for data entry.



How Can I Limit the Size of the Background Task Database Table?

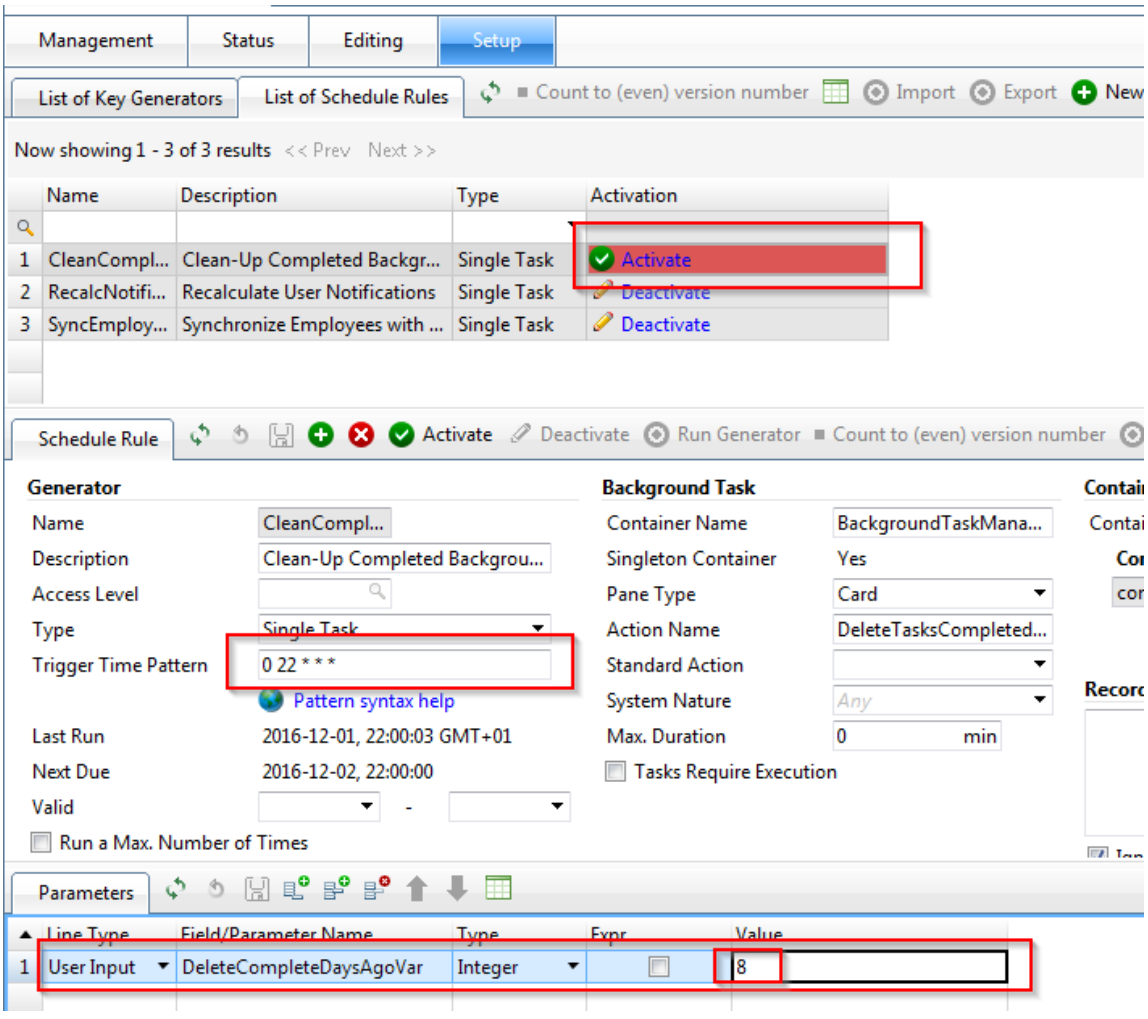
Background tasks make up an important trace of things that has been done, letting a system administrator monitor the status of things.

It's possible to set up a schedule rule that automatically deletes completed background tasks that are older than X number of days. This is even a default rule. By default this rule deletes all completed background tasks that are 8 or more days old.

It is possible to configure:

- Whether this should happen
- What the time range should be
- When the rule triggers

This can be done from: Set Up / Background Tasks, Setup » List of Schedule Rules » Schedule Rules » Parameters.



By editing the rule CleanCompletedBackgroundTasks you can alter this. In this example, if you change the Value field in the table "Parameters" from 8 to 15, tasks will have to be at least 15 days old to be deleted.

Just remember that the longer the duration, the larger the size of the Background Tasks database table.

Can I See Tasks That Were Run as Another User?

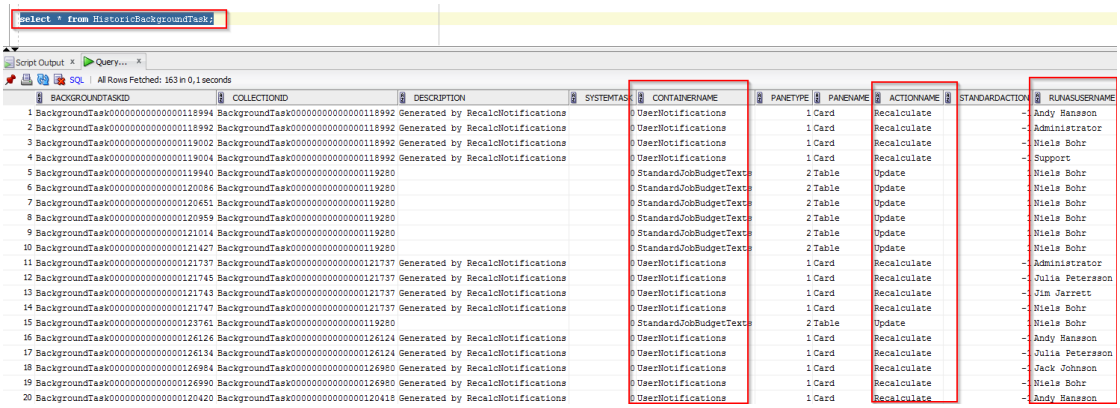
In the background task status filter, you can see whether a background task was run on behalf of another user. If background tasks have been deleted, there may still be a chance that you can dig up some information.

By default any background task that is set-up to run on behalf of some specific user (i.e., not as the background administrator user) is stored in a condensed form in the database table HistoricBackgroundTask. You can choose:

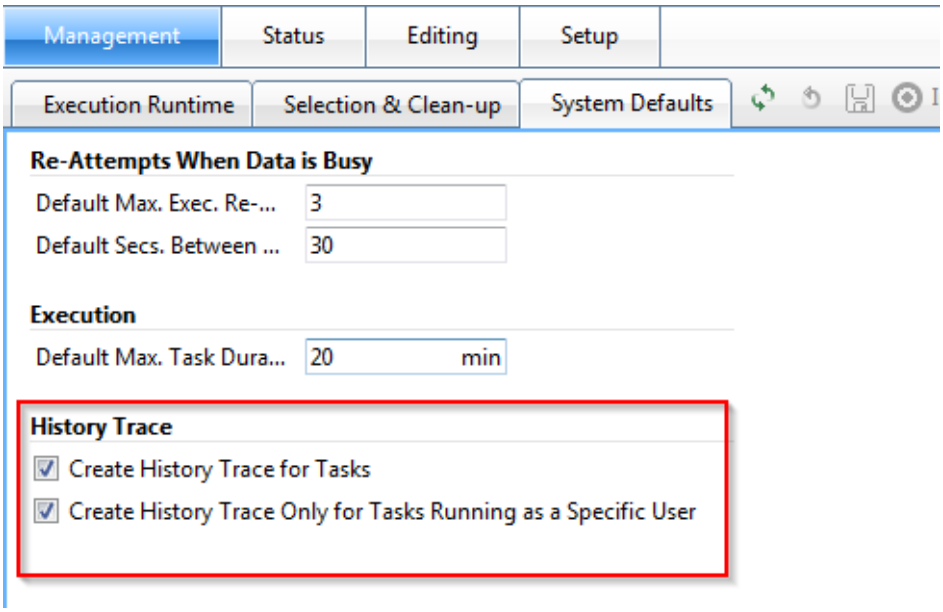
- Whether completed non-failing background tasks are stored in condensed form or not
- If they are, whether only background tasks running on behalf of a specific user are stored like that (in order to keep the size of the table down)

By default, there is no place where you can inspect this historic information in a workspace. And the table is not indexed. This means finding data might take a long time.

Basically, you need to do it using SQLPlus or another database administration tool.



The settings declaring to what extend (if at all) background tasks are logged in the `HistoricBackgroundTask` table, can be altered from Set Up / Background Tasks, Management » System Defaults.



Whenever any of these fields are changed, it is logged (also in HistoricBackgroundTask) who did the change, and what the implication of the change was.

What is the Purpose of the HistoricBackgroundTask table?

The purpose is to resolve a situation where an entity says, for example, “Approved by User name”, but the user refuses any knowledge of that. In this case, we can search for a HistoricBackgroundTask that has done the approval action on behalf of that user. If such a HistoricBackgroundTask exists, the situation can be explained. Obviously this can only be done if history trace is enabled.

How Can I Limit the Size of the HistoricBackgroundTask Table?

You can only limit if/to what extend data is stored in HistoricBackgroundTask. It is not possible to delete any records from this table from within Maconomy. Hence, the table will keep growing as time goes by.

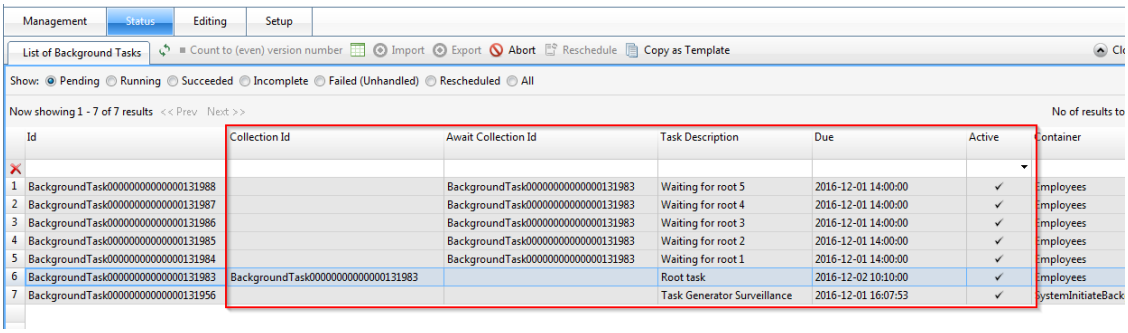
For this reason, it may eventually be necessary to delete “old” portions of this table. Doing so will have to be a manual task done by a DBA, for example using SQLPlus.

Execution Threads are Active; But No Tasks are Being Executed. Why?

You need to figure out whether the background tasks are ready for execution. This includes checking:

- What is the due date/time for the background tasks?
- Are the background tasks active or inactive?
- Are the pending tasks awaiting some other task that isn’t completed, or isn’t due?
- Are the tasks associated a nature different from the nature of the current system?
- Are there errors being reported by the task execution engine (see server log)
- Does the background administrator have adequate access rights?

Properties of the tasks can be seen in: Set Up / Background Tasks, Status » Background Task.



In the above screenshot, you can see that the “Waiting for root n” tasks cannot be picked up for execution because they await the “Root task”. The “Root task” awaits the due date/time before it can be picked up.

Another cause could be related to more critical errors experienced by the background task execution engine. This status of this can be tracked by enabling logging in the `logback.xml` file found in the `configuration` folder in the coupling service installation folder.

Assuming that you have an appropriate “appender” defined (it could be the `FILE` appender, or it could be one dedicated for background tasks execution), you can enable all logging concerning the background task execution engine by adding this to the logback.xml file:

```
<logger name="com.maconomy.coupling.service.batch" additivity="false">
```

```
    <level value="DEBUG"/>
    <appender-ref ref="FILE" />
  </logger>
```

Level TRACE will give an even higher amount of information from the background execution engine. Errors could include, not being able to login, background task user not having adequate access etc.

My Background Tasks Are Not Run at an Adequate Pace – Why?

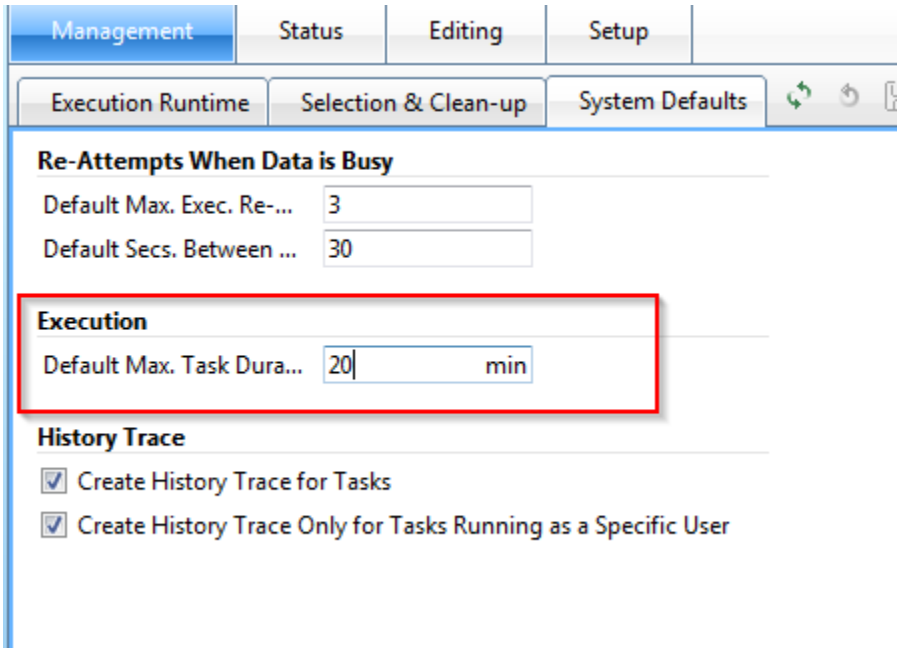
If background tasks are not picked up "soon enough" it may indicate that the execution engine is too busy performing background tasks.

You may need to consider:

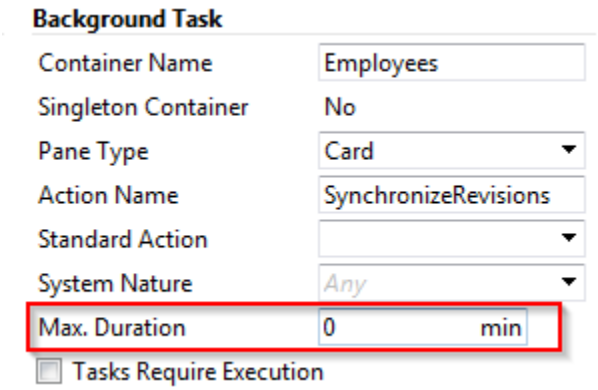
- How many background task execution threads are enabled (across all server machines)
- Are any of the execution threads stopped or suspended?
- Are the pending tasks awaiting other tasks?
- Are the tasks being picked up for execution, but time out? In that case, is the maximum duration of the task set to an adequate amount of time?
- Are any of the expected task assigned a nature different from the current systems?
- Are the tasks activated?

Most of these topic are mentioned elsewhere in this FAQ.

The Maximum duration can be set specifically on a task, as well as on task generator rules. The default maximum duration can be configured in Set Up / Background Tasks, Management » System Defaults.



Maximum duration for tasks created by a schedule rule can be configured in: Set Up / Background Tasks, Setup » List of Schedule Rules » Schedule Rule.



A value of 0 means “apply the default max. duration”.

My Background Task is Green, But It Didn’t Run – Why?

If the task is “green” (i.e., “Succeeded”) it means that the execution framework believes (and has no reason not to) that the task was successfully run.

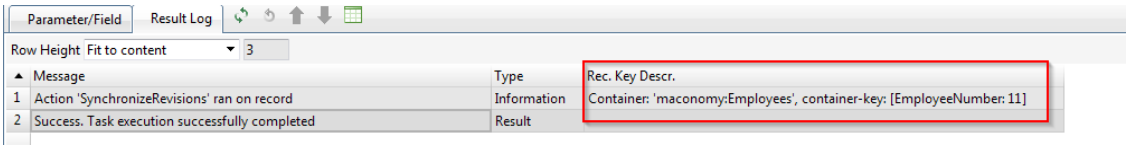
There are a couple of things you can check:

- Is the record key(s) referencing the expected record(s)? If not, the action was executed on something other than what you expected.
- Is the record condition (if any) ruling out the expected record(s)? If so, the records you thought was being processes has been skipped.
- How many records have been executed? If no records was actually executed, this is by default considered a successful background task. It is possible to flag a background task so that if the specified action is not applied for any record, it will be considered an error.

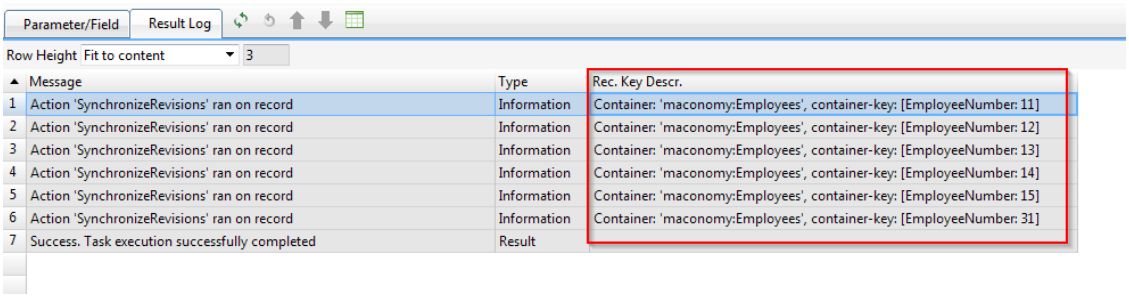
Notice:

In the log for the background task (in the Workspace) you can see exactly which record keys the action has been applied to! Go to: Set Up / Background Tasks, Status » Background Task » Result Log.

In this case, the action ran on one record: (EmployeeNumber = 11)



In this case, the action ran on several records: (EmployeeNumber in {11, 12, 13, 14, 15, 31})



My Background Tasks Generator Hasn't Run - Why?

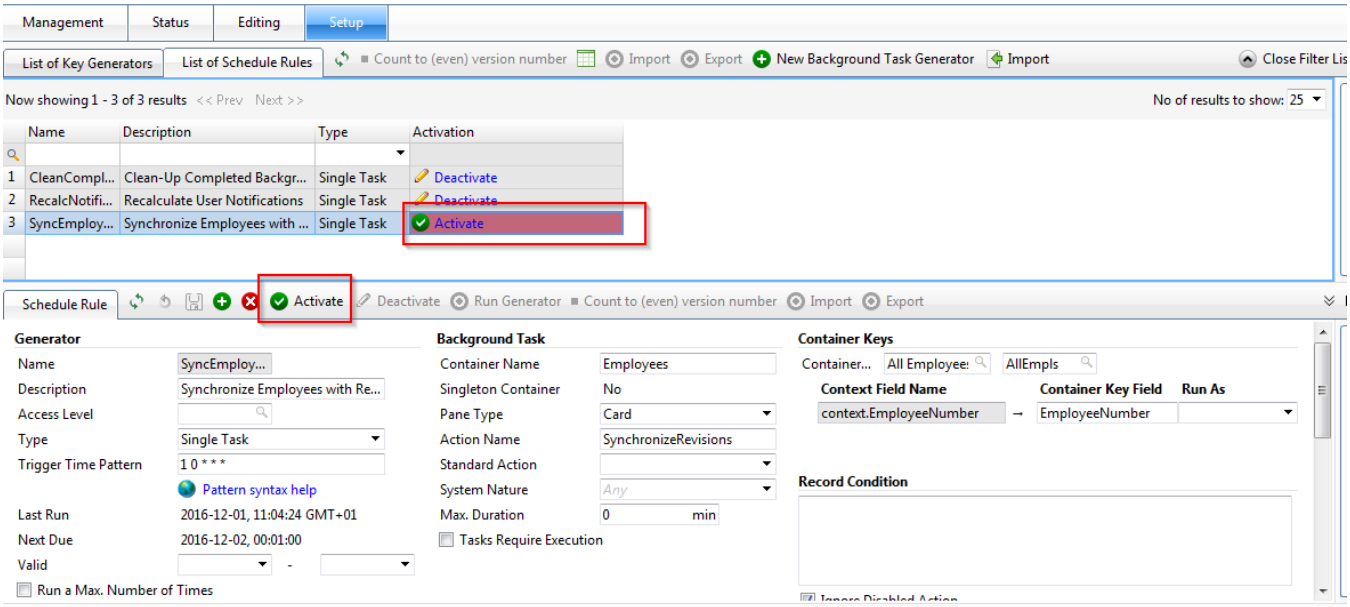
You should investigate:

- When is the next due date/time of the background task generator? If this is sometime in the future, maybe the time trigger pattern is not what you think, and the rule really shouldn't have run
- When was the task generator last run?
- What is the validity date interval for this rule? If the rule is no longer valid (or not yet valid) this explains why.
- Has the rule been set up to run a fixed number of times? If so, are there any runs left? If not, this could explain why.
- Is the rule active? If not, this may explain why.

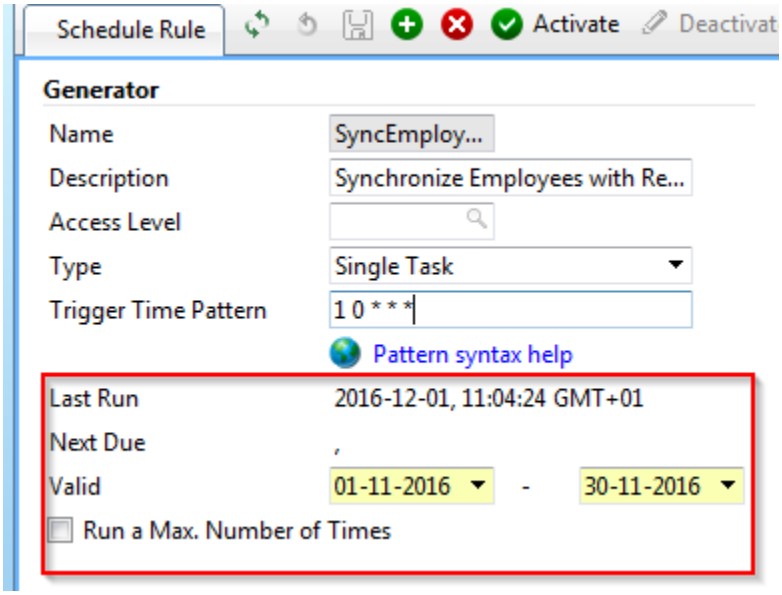
Notice:

You can investigate these things by going to: Set Up / Background Tasks, Setup » List of Schedule Rules » Schedule Rule.

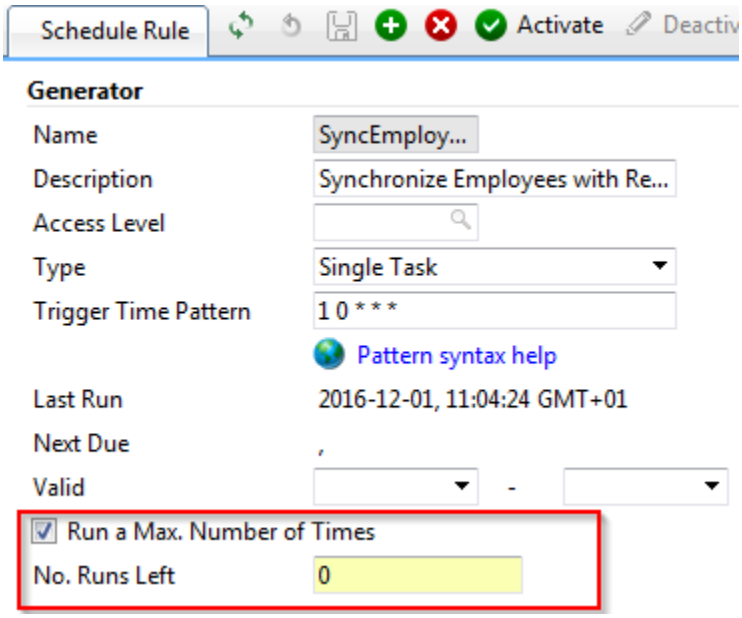
If you can activate the rule, it is currently inactive.



Other settings can be seen here:



If the rule is set to be run a maximum number of times, check if there are no runs left:



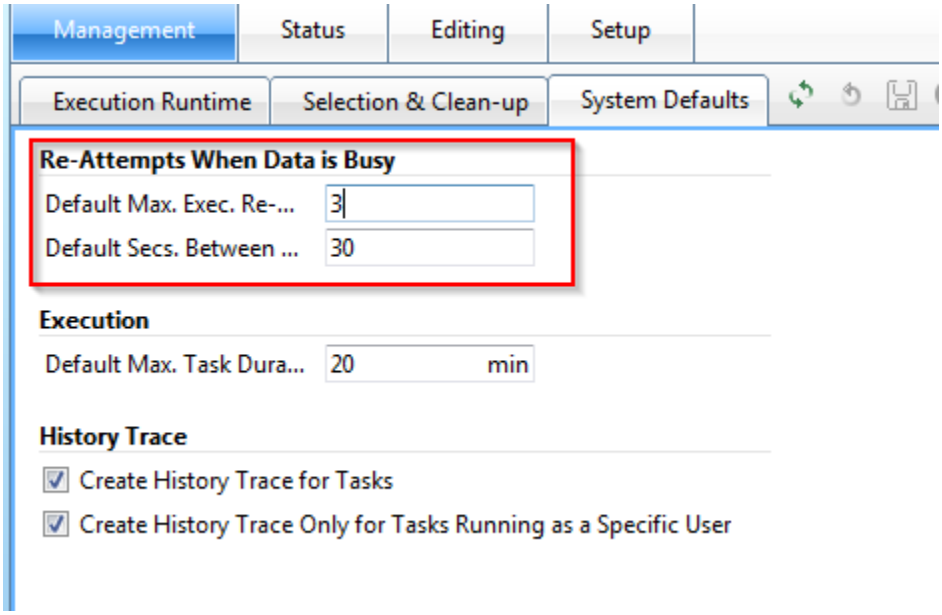
What Happens If Other Users are Changing the Data of a Background Task?

Just as “normal” users can experience that someone else changes that the user is working with, this can happen for background tasks. In case of a “Data has been changed by another user”, the execution engine will automatically detect this situation and will re-schedule the background task to be attempted again “soon”. The number of times this “retry” will occur can be configured, as can the interval between retries. If the task keeps failing for this reason, it will eventually be marked as failed with a “Data Busy” error indication.

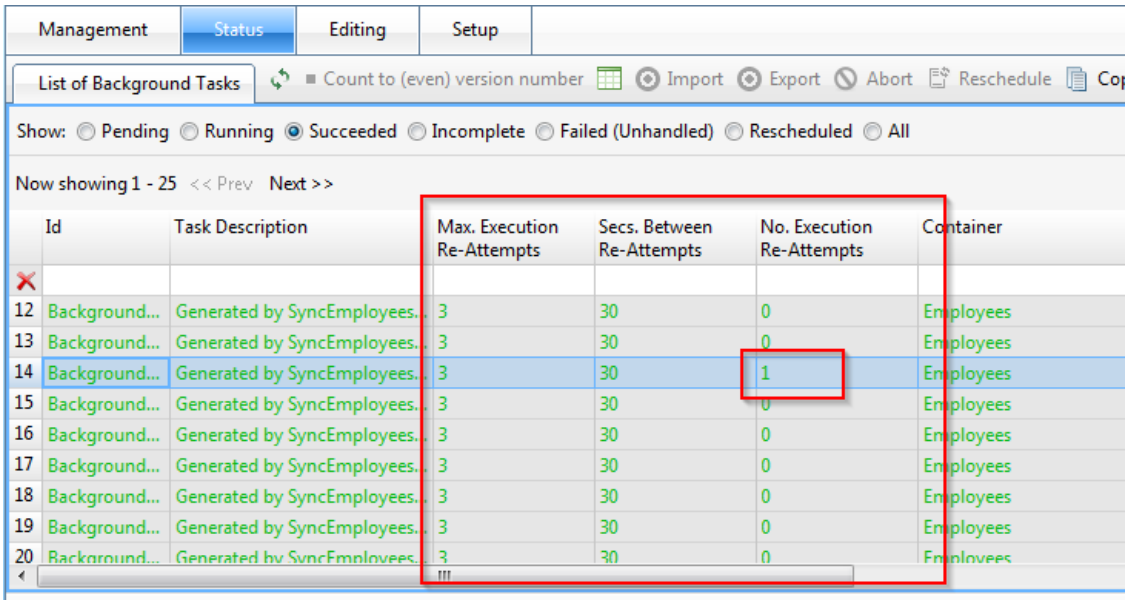
You can track how many times a given task has been re-attempted.

To set-up the default number of re-tries and duration between re-tries, go to Set Up / Background Tasks, Management » System Defaults.

Here you can specify the default number of re-tries and the time between re-tries.



It is possible to look-up what the settings for a given task is, including the number re-tries that has occurred.



Where Did the Output of a Background Task End Up?

When a background task executes, sometimes output documents need to be taken care of. This could be, for example, printed journals, invoices or log files (e.g., from import programs).

If a background task produces output, it needs to be handled. If it is not specified how to handle output files (but the “core logic” otherwise succeeded) the task ends up having status “Incomplete”.

The background task log will show what output documents were produced and how/if they were handled.

Basically, output files can be handled in two ways:

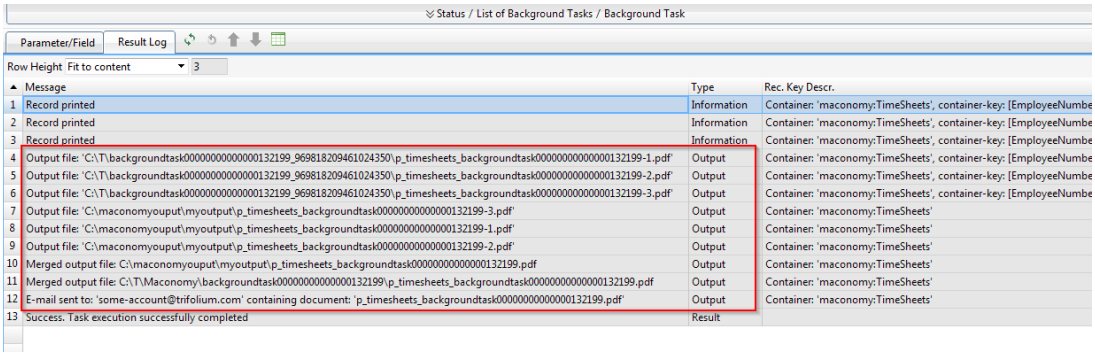
- Sent by e-mail to a specific recipient
- Stored on the file system somewhere

You can associate a file handler to a “pattern”. If you do, only output files having a name that matches will be handled by that handler. In this way, you can have PDF-files stored in one location, and TXT-files stored in another location.

You can also choose to have some files both sent by e-mail and stored on a file system.

To see what output documents were produced and how they are handled, go to: Set Up / Background Tasks, Status » List of Background Tasks » Background Task » Result Log.

In the example below, files where stored on a file system and sent by e-mail.



	Message	Type	Rec. Key Descr.
1	Record printed	Information	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
2	Record printed	Information	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
3	Record printed	Information	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
4	Output file: 'C:\T\backgroundtask0000000000000132199_969818209461024350\p_timesheets_backgroundtask0000000000000132199-1.pdf'	Output	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
5	Output file: 'C:\T\backgroundtask0000000000000132199_969818209461024350\p_timesheets_backgroundtask0000000000000132199-2.pdf'	Output	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
6	Output file: 'C:\T\backgroundtask0000000000000132199_969818209461024350\p_timesheets_backgroundtask0000000000000132199-3.pdf'	Output	Container: 'maconomy:TimeSheets', container-key: [EmployeeNumbe
7	Output file: 'C:\maconomyouput\myoutput\p_timesheets_backgroundtask0000000000000132199-3.pdf'	Output	Container: 'maconomy:TimeSheets'
8	Output file: 'C:\maconomyouput\myoutput\p_timesheets_backgroundtask0000000000000132199-1.pdf'	Output	Container: 'maconomy:TimeSheets'
9	Output file: 'C:\maconomyouput\myoutput\p_timesheets_backgroundtask0000000000000132199-2.pdf'	Output	Container: 'maconomy:TimeSheets'
10	Merged output file: C:\maconomyouput\myoutput\p_timesheets_backgroundtask0000000000000132199.pdf	Output	Container: 'maconomy:TimeSheets'
11	Merged output file: C:\T\Maconomy\backgroundtask0000000000000132199\p_timesheets_backgroundtask0000000000000132199.pdf	Output	Container: 'maconomy:TimeSheets'
12	E-mail sent to: 'some-account@trifolium.com' containing document: 'p_timesheets_backgroundtask0000000000000132199.pdf'	Output	Container: 'maconomy:TimeSheets'
13	Success. Task execution successfully completed	Result	

Admin Field Descriptions

Note: The information in this section is in DRAFT form and is not yet updated to reflect 2.5.2.

Setup » Background Tasks workspace » Management section » Execution Runtime tab

This tab displays information that was current at the time you loaded or refreshed the workspace. The execution of background tasks is highly dynamic, but the data in this tab only changes when you click **Refresh**.

Status island

Field	Description
Execution of Background Tasks	This field displays the current overall status of the background tasks system. It contains one of the following values: ▪ **Running** – New scheduled tasks may be picked up for execution. The field can have this value even if no coupling services are set up for executing background tasks. However, it is strongly advised that there always be at least one coupling service set up; otherwise, Maconomy will not carry out background tasks. ▪ **Stopping** – No new scheduled tasks may be picked up for execution, but currently running tasks exist. Click the **Refresh** action to update the status. ▪ **Stopped** – No new scheduled tasks may be picked up for execution, and no running tasks exist. If you plan to shut down coupling services, make sure that the status is “Stopped”. Otherwise, you might continue to see running tasks for that coupling service (even if these tasks are no longer running). For more detailed status and control (especially when shutting down a single coupling service in a setting where several coupling services run concurrently), refer to the information in the Execution Nodes sub-tab.
No. Running Tasks	This field displays the number of currently running background tasks. Click the **Refresh** action to monitor how the number changes over time.

Setup » Background Tasks workspace » Management section » Execution Runtime tab » Execution Nodes sub-tab

This sub-tab displays information that was current at the time you loaded or refreshed the workspace. The execution of background tasks is highly dynamic, but the data in this sub-tab only changes when you click **Refresh**.

Field	Description
Node	An execution node carries out background tasks. This field displays identification for each node in the system. ▪ The top node is always called “Maconomy” and is of the “System” type. It represents the entire collection of task execution nodes for the specific Maconomy installation. ▪ Second-level nodes are of the “Host” type. Each represents a computer/machine that runs one or more coupling services which carry out background tasks. The title of the node identifies the name of the computer. ▪ Third-level nodes are of the “PID” type, and represent actual coupling service instances. The title of the node represents the process ID of a coupling service that runs on the “Host” parent computer. This allows you to differentiate between two coupling services that run on the same computer. ▪ Fourth-level nodes are of the “Thread” type, and represent actual task execution threads. The title of the node is a reference number identifying each execution thread on the parent “PID” coupling service. Each “Thread”-type node can carry out the actual background tasks. The parent “PID” nodes merely group the execution threads together, and allow you to start/stop processing multiple threads at once.
Latest Activity	This field only displays information for “Thread”-type nodes. It shows the thread’s most recent query to the system for the execution of a background task. Even stopped threads exhibit activity by querying. This activity causes Maconomy to update thread timeout information.
Timeout	This field only displays information for “Thread”-type nodes. It shows the time the execution thread timed out. If the thread does

	not report back before this time, the system may opt to remove the thread from the overview and abort any task that the thread is seemingly running.
Running Action	This field only displays information for “Thread”-type nodes. It displays the name of any action that the execution thread is currently performing.
Running Container	This field only displays information for “Thread”-type nodes. If the execution thread is currently running a background task, this field displays the name of the container in which the current action resides.
Duration	This field only displays information for “Thread”-type nodes. If the execution thread is currently running a background task, this field displays the running duration of the task execution. If a task is long-running, it could mean one of the following: ▪ The task is simply taking longer than expected. ▪ The coupling service running the task was shut down while running that task. The task seems to be running (even if it is not), and the duration of the task execution continues to increase until the thread times out. At any time after that, the system may choose to abort the task and remove the execution thread from the overview.
Status	This field displays the status of the node, and can have one of the following values: ▪ **Running** – The node is allowed to pick up tasks for execution. If the specific node has sub-nodes, these may be more restrictive. For example, a “PID” node with the **Running** status could have one or more “Thread” nodes with the **Stopped** status. The stopped “Thread” nodes do not pick up tasks for execution. ▪ **Stopped** – The node and its sub-nodes (if any) are not allowed to pick up tasks for execution. ▪ **Stopped (when done)** – This status is similar to “Stopped”, but for a node that is

	currently executing a task. The execution will continue until done. ▪ **Suspended** – The node is not allowed to pick up tasks for execution, but only because one of its parent nodes is "Stopped". If the parent node is restarted, this status changes to "Running". ▪ **Suspended (when done)** – This status is similar to "Suspended", but for a node that is currently executing a task. The execution will continue until done.
Start/Stop	This is an action link to the **Enable Execution** or **Disable Execution** action (not included in the default layout). The available action link depends on the status of the specific node. If you click the **Stop** link, Maconomy disables further task execution for the node and its sub-nodes. For example, if you click **Stop** for a PID coupling service node, that coupling service will no longer receive tasks for execution even if it sends out queries for them. Currently running tasks are allowed to continue until finished. If you click the **Start** link, Maconomy enables further task execution for the node. Each sub-node's status remains unchanged. **Warning**: Before you shut down a coupling service, make sure it is not carrying out any tasks. You can do this by stopping any PID sub-nodes, waiting for all threads to finish currently running tasks, and then shutting down the coupling service. If you fail to do this, you risk one or more background tasks timing out and being aborted by the system. Rescheduling these aborted tasks will require manual intervention, which defeats the purpose of scheduling background tasks in the first place.
Running Task Id	This field only displays information for "Thread"-type nodes. If the execution thread is currently running a background task, this field displays the ID of that task.

Action	Description

Remove Timed-Out Threads	Click this action to remove all timed-out execution threads. Remember that this sub-tab displays data that was current at the time you loaded or refreshed the workspace, and that timeout information may have changed since then. To make sure you are looking at the latest information, click **Refresh** before using this action. **Note**: By default, the system invokes this action every 10 minutes. However, you can customize the interval on the coupling service.

Setup » Background Tasks workspace » Management section » Execution Runtime tab » Execution Nodes sub-tab » Background Task panel

If you select an execution thread from the Execution Nodes sub-tab, this panel displays the properties of the background task that the thread is currently running. These properties were originally entered manually or generated (by a background task generator) in the Editing section of the Background Tasks workspace.

Setup » Background Tasks workspace » Management section » Execution Runtime tab » Execution Nodes sub-tab » Background Task panel » Parameter/Field sliding panel

If you select an execution thread from the Execution Nodes sub-tab, this sliding panel displays further details about the background task the thread is currently running. Again, this information was originally entered manually or generated (by a background task generator) in the Editing section of the Background Tasks workspace.

Setup » Background Tasks workspace » Management section » Selection & Clean-up tab

Selection Criteria island

Field	Description
Container Name	If you specify a container in this field, Maconomy displays completed background tasks that are related to the specified container. Otherwise, no restrictions are made based on this field.
Action Name	If you specify an action in this field, Maconomy displays completed background tasks that are related to the specified action. Otherwise, no restrictions are made based on this field.
Completed Before	If you specify a date in this field, only background tasks completed prior to that date are listed in the Background Task sub-tab.

Status	Select a value from the drop-down list to filter the lines shown in the sub-tab. Only completed background tasks with the specified status are displayed. If you leave this field blank, no restrictions are made based on the status.
From Access Level-To Access Level	In these fields, you can specify a range of access levels. Only completed background tasks associated with an access level that falls within the specified range are displayed in the sub-tab.
Show Lines	Select this check box to display the completed background tasks that fall within the selection criteria you specify. Maconomy displays the lines in the Background Task sub-tab.
[Delete Shown]	If you select the **Show Lines** check box, you can click this action link to delete all the background tasks listed in the sub-tab. Otherwise, this action remains grayed out.

Action	Description
Delete Tasks Completed Before…	Click this action to delete any tasks completed (that is, successful, incomplete, or failed) before the date specified (typically in a wizard). You can also specify the date relative to the current date, but this is not possible in the default wizard layout. This is done using the background task generator for automatically deleting old background tasks.

Setup » Background Tasks workspace » Management section » Selection & Clean-up tab » Background Task sub-tab

The Background Task sub-tab displays the completed background tasks that match the selection criteria specified in the Selection & Clean-up tab.

Field	Description
Id	This field displays the unique ID of the background task.
Task Description	This field displays a description of the task.

Container	This field displays the container name associated with the task.
Action Name	This field displays the action name associated with the task.
Date Started	This field displays the date when the task started.
Date Ended	This field displays the date when the task ended.

Setup » Background Tasks workspace » Management section » System Defaults tab

These fields determine the default settings for new background tasks.

Re-Attempts When Data is Busy

Field	Description
Default Max. Exec. Re-Attempts	In this field, specify the default number of times the system is allowed to re-attempt a task if execution fails because data is locked to another user. You can override this setting if you fill out the **Max. Execution Re-Attempts** field on a specific task. For example, a task attempts to approve a job budget, but the corresponding budget is being edited by another user. The task execution system could automatically reschedule the task to run again at a later time. This setting determines the maximum number of times the system does this before giving up and marking the task as failed.
Default Secs. Between Re-Attempts	In this field, specify the default number of seconds between system re-attempts to execute a task. The value must fall within the range of 0 – 3600 seconds (or one hour).

Execution island

Field	Description
Default Max. Task Duration	In this field, specify the default maximum task duration. The value is measured in minutes. When the system picks up a task for execution, the execution thread will be considered timed-out after the period of time you specify in this field. If the thread has not reported back before the time is up,

	Maconomy may automatically abort the task. For example, this could occur if the coupling service is shut down during execution of background tasks. You can override this value on individual tasks, as well as on tasks generated by schedule rules. If nothing is specified on the individual task, the task assumes the value specified in this field.

History Trace island

Field	Description
Create History Trace for Tasks	Select this check box if you want Maconomy to maintain a history trace of successfully executed background tasks. Each time this check box is selected/cleared, the database logs the change and the user who made the change. Maconomy keeps the history trace even after the tasks are deleted. In this way,you can investigate and find out whether a certain operation has been performed by a background task, on what container key, and at what date and time. Since the size of this database table may become too large over time, do not select this check box if it is not relevant or important for your installation and legislation that you keep track of successful tasks.
Create History Trace Only for Tasks Running as a Specific User	Select this check box if you want Maconomy to maintain a history trace only for background tasks set to run on behalf of specific users. This keeps the size of the history trace down. If you select this check box, you must also select the **Create History Trace for Tasks** check box. Each time this check box is selected/cleared, the database logs the change and the user who made the change.

Setup » Background Tasks workspace » Status section » Background Task tab

If you select a background task from the filter list, this tab displays the properties of that task. These properties were originally entered manually or generated (by a background task generator) in the Editing section of the Background Tasks workspace.

Setup » Background Tasks workspace » Status section » Background Task tab » Parameter/Field sub-tab

If you select a background task from the filter list, this sub-tab displays further details about that task. Again, this information was originally entered manually or generated (by a background task generator) in the Editing section of the Background Tasks workspace.

Setup » Background Tasks workspace » Status section » Background Task tab » Result Log sub-tab

This sub-tab shows detailed result logs for tasks. In the case of failed tasks, the error encountered is explained in more detail. In the case of completed tasks, a detailed log output is provided for notifications, warnings, and so on.

Field	Description
Message	This field displays any message that Maconomy displayed while executing the background task shown in the tab.
Type	This field displays the type of message, and can have one of the following values: ▪ **Notification** – This type covers any notification that Maconomy issues during execution. For example: “Job No. jobnumber has been created.” ▪ **Warning** - This type covers any warning that Maconomy issues during execution. For example: “The Time Sheet has been temporarily submitted. Continue?” ▪ Background tasks always accept warnings issued by Maconomy. ▪ **Error** – This type covers any error message that Maconomy displays during execution. These could be user errors (such as “No registrations can be made before or after the employment period.”) or internal program errors. ▪ **Trace** – This type covers stack trace information relayed as part of error messages. A stack trace is useful for debugging and error identification. ▪ **Input** – This type covers messages displayed which are related to input documents that Maconomy picked up during task execution. ▪ **Output** – This type covers messages displayed which are related to output documents produced during task execution. For example, an output

	message could be about where an output document was stored. ▪ **Information** – This type covers other informative messages related to the task execution. ▪ **Result** – This type covers messages that state the final result of the task execution.
Rec. Key Descr.	This field displays a textual representation of the record on which an action was performed.

About Deltek

Better software means better projects. Deltek is the leading global provider of enterprise software and information solutions for project-based businesses. More than 23,000 organizations and millions of users in over 80 countries around the world rely on Deltek for superior levels of project intelligence, management and collaboration. Our industry-focused expertise powers project success by helping firms achieve performance that maximizes productivity and revenue. www.deltek.com