Job Execution Framework for SSIS

Documentation

Version 1.0

Author: Christoph Seck

Supported by:





# Table of Contents

[1. Table of Contents 2](#_Toc452467956)

[2. Introduction 3](#_Toc452467957)

[2.1 In a nutshell 3](#_Toc452467958)

[2.2 History and Thanks 3](#_Toc452467959)

[3. Master Package 4](#_Toc452467960)

[3.1 Execution 4](#_Toc452467961)

[3.2 Logic 5](#_Toc452467962)

[3.2.1 Start 6](#_Toc452467963)

[3.2.2 LoopOverPackages 7](#_Toc452467964)

[3.2.3 Finish 7](#_Toc452467965)

[9. Configuration Tables in the configuration database 8](#_Toc452467966)

[9.1 Package and JobStep 8](#_Toc452467967)

[9.2 JobStep and Groupings 9](#_Toc452467968)

[9.3 JobStep and Constrains 11](#_Toc452467969)

[9.4 JobStepCluster 11](#_Toc452467970)

[9.5 Parameter and ParameterValue 12](#_Toc452467971)

[10. Getting the Metadata with “GetMetadata.dtsx” 15](#_Toc452467972)

[11. Important Stored Procedures & Functions 15](#_Toc452467973)

[11.1 Package execution 15](#_Toc452467974)

[11.2 prc\_GetNextPackage 15](#_Toc452467975)

[11.3 prc\_InitJobExecution 15](#_Toc452467976)

[12. Planed Downtime 17](#_Toc452467977)

[12.1 Tables 17](#_Toc452467978)

[12.2 Stop and Recovery 18](#_Toc452467979)

[13. How To: Executing a package 19](#_Toc452467980)

[13.1 Basics 19](#_Toc452467981)

[13.2 Execution prerequisites in the Database 19](#_Toc452467982)

[13.3 Executing the Master Package 20](#_Toc452467983)

[13.4 “Stolpersteine” 23](#_Toc452467984)

[14. Installation 23](#_Toc452467985)

# Introduction

## In a nutshell

The main task of the Execution Framework is the management of ETL Processes implemented with SQL Server Integrations Services. The central component is a single SSIS package, the “Master Package”, which executes all the other packages (“work packages”), governed by configuration data stored in the configuration database.

Instances of the Master Package may run in parallel on the same server or on several servers, executions and problems are logged

## History and Thanks

The first version was developed for a company with several production plants that needed a central DWH for production data. Because of security constrains and the questions of load balancing it was necessary to distribute the ETL processes over different servers with the possibility of automatic or ad hoc reconfiguration.

The second version was developed with Cluster Reply and utilized the new possibilities which came for SSIS with SQL Server 2012.

The third and current version was a complete reimplementation done during my time at Ceteris AG (www.ceteris.ag). The discussions in the open atmosphere at Ceteris greatly enhanced the original ideas. Without the feedback of guys like Thorsten Huss, Chris Jacob, Ben Kanter and Markus Schechner the Framework would be much buggier and still lack some important features. An optional interface for the Advanced Logging Components for SSIS from Ceteris is still included.

Further input came from our main customers where we used and enhanced the Framework. Special thanks here to the DPD IT team.

The deal with the customers always was: You do not have to pay for the usage of the framework, but all enhancements done at your site go back to the product and can be used by all other users of the product.

Now Ceteris took this principle a step further with allowing me to publish the Framework to GitHub under the MIT license. Thanks!

# Master Package

## Execution

The Master Package “Master.dtsx” normally is executed by a SQL Server Agent Job (it is also possible to execute it via a stored procedure etc.). As a first step in this job normally the SSIS package “GetMetadata.dtsx” should be executed. This package grabs some metadata from the SSISDB like newly deployed SSIS packages etc.

When the Master Package is executed certain parameters govern its configuration. These parameters have to be set in the agent job (or the executing stored procedure etc.).

On the one hand, there are some technical parameters which e.g. determine the maximal number of work packages executed in parallel etc.

On the other hand, there are those parameters which specify the work packages the Master Package tries to execute. Currently there are:

1. ApplicationID
2. JobGroupID
3. JobID
4. LayerID
5. MetaGroupID
6. StepNo = n (Start from StepNo n)
7. StepNoEqual (execute all Steps versus execute just the step specified by StepNo)

A special case are the reserved parameters: Int\_ClientID, Bool\_IsInitialLoad, int\_ParentAuditID, guid\_ParentSourceGUID. These parameters are passed through to the work packages. Means, if the child package supports these parameters they will be set by the master package according to the values the master package itself was fed with. It is not possible to use the standard mechanism of setting parameters for a job (cf. 7.5). Thus they should not be used as regular Package Parameters but only according to their special role for the advanced logging e.g. The names of these reserved parameters are maintained in the table etljob.ReservedParameter

## Logic

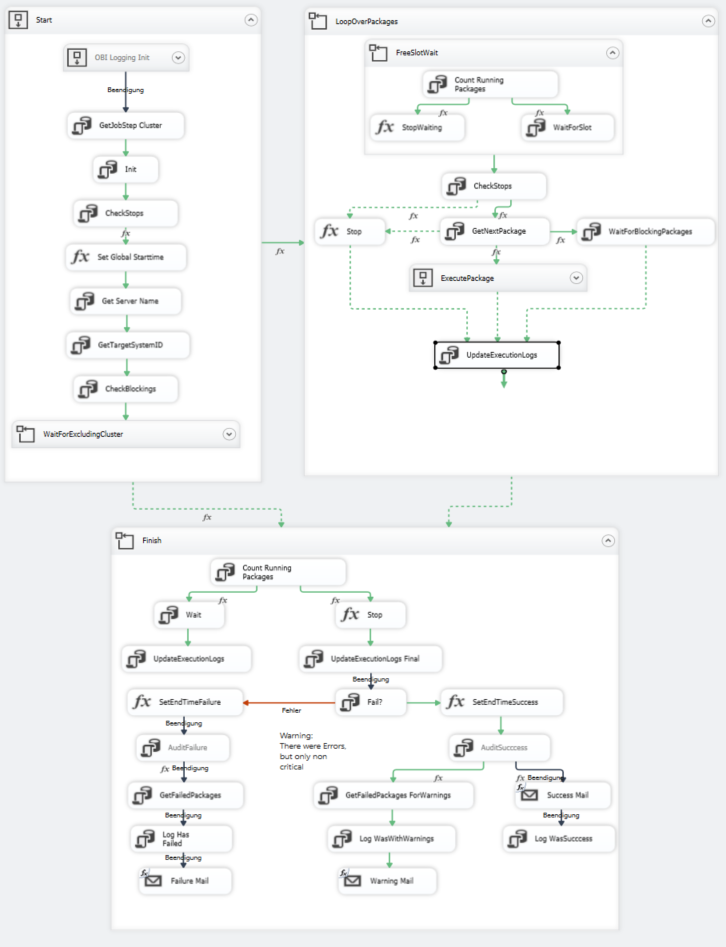


Figure 1: Master Package

## Start

1. OBI Logging Init: If you are using the “OBI” Advanced Logging components from Ceteris AG, you should activate this container and its subtasks. You than should further point the OBI connection manager to the correct logging database used by the OBI components and activate the tasks “AuditFailure” and “AuditSucccess” in the third main container “finish”.  
   If you do not use these components, just ignore or delete these tasks and the OBI connection manager. If you do not delete the connection manager, make sure it points to a valid database.
2. GetJobStep Cluster: Is there a name for the current configuration (entry in the view etljob.vw\_JobStepCluster)? If yes get ID, if no create new in Table etljob.JobStepCluster.
3. Init: Which work packages are identified by the current parameters? Copy them to the queue (table “JobStepExecution”), store the current JobExecutionID and the parameters with them.  
   Special case: when the package parameter Restart is set to 1, the steps of the last execution of the current configuration (identified by its JobStepCluster as read in the previous step) are copied. Only “unfinished business” (JobSteps failed, not run etc.) is given a status to include it in the current execution.
4. Remember start time for timeout detection (not yet implemented)
5. Get server name (for status mails)
6. Get ID of System (Development, QA, Production etc.) “All” has ID 0  
   Helpful if the configuration of those system is maintained in one configuration Database.
7. Check Blockings: Checks if there are executions of the MasterPackage which faileded on which the current executions of the MasterPackage depends (-> etljob. vw\_JobStepClusterConstrain) or if it failed in the previous run. If Configuration of the JobStepCluster says so (do not retry etc.) all steps are canceled. An executions of the MasterPackage is classified by the JobStepCluster (-> etljob. vw\_JobStepCluster) it belongs to.
8. Wait for excluding Clusters: Via entries in the view etljob.vw\_JobStepClusterExclusion it can be defined that certain configurations are not allowed to run in parallel. These constrains are checked here. The current Master package execution is waiting until no excluding Cluster (= Master package with configuration specified by that excluding Cluster) is running

## LoopOverPackages

1. FreeSlotWait: Based on the restrictions to parallelity: Wait until there is a free slot.
2. Execute Package: Based on the Column StepNo (table JobStepExecution) and possible dependencies on other packages (defined in the view vw\_JobStepConstrain): Get the next package from the queue (table JobStepExecution, current JobExecutionID) and execute:
   1. Prepare package for execution (ceck if it exists etc.)
   2. Get list of parameters
   3. Set the values of those parameters
   4. Start excecution
3. If there is none or a critical work package failed, check:
   1. Critical work package failed or nothing left to to? => Stop
   2. Blockings because of dependencies? => Wait
4. Update tables JobExecution and JobStepExecution

## Finish

1. Wait for Packages to finish.
2. Send mail with failure or success message

# Configuration Tables in the configuration database

The Execution Framework uses the namespace etljob. The Master Package and its related stored procedures never access those tables directly; they always use the corresponding views. The views use the naming: vw\_NameOfTable. Background: In a future version the configuration data will be placed in MDS, using the views will make this or similar migrations simpler.

## Package and JobStep

From the SSISDB a list of all deployed SSIS packages is acquired and stored in the table “Package” via the SSIS package “GetMetadata.dtsx”.The table “JobStep” builds the repository for all work packages the Master Package may access. JobStep references Packages, thus a SSIS Package may be executed by different Jobsteps.

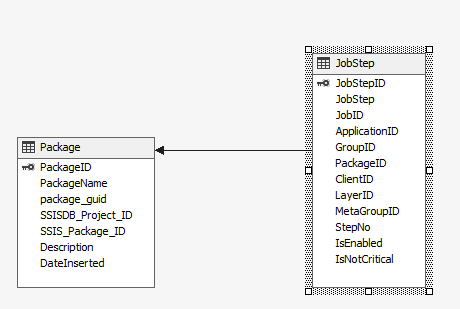


Figure 2: Tables Package and JobStep

## JobStep and Groupings

JobSteps may be classified by different categories. Example:

|  |  |  |
| --- | --- | --- |
| Name | Possible Values | Comment |
| Application |  | Big Grouping |
| Job | Standard | “Main Container” (Is **not** the same as a SQL Agent Job). Standard Job: Job where GetMetadata.dtsx automatically adds new packages |
| Group | FullLoad |  |
| Layer | Stage, Core, Star, Cube | When naming conventions are used they can be used to automatically assign new packages to one of them by GetMetadata.dtsx |
| MetaGroup |  | Further option for Job partitioning |

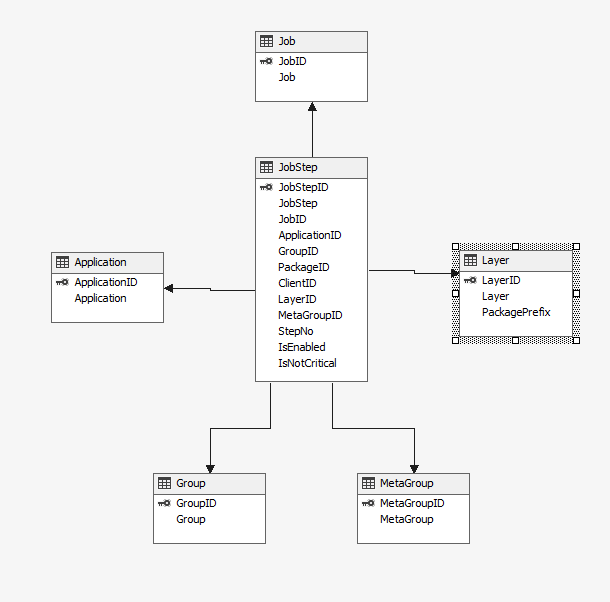
In the database this classification currently is implemented by five tables, referenced by JobStep:

Figure 3: JobStep and Classifications

“Layer” is somewhat special. The PackagePrefix is used to identify the Layer a package belongs to. If the package info is read from SSISDB via the SSIS package “GetMetadata.dtsx” and its name starts with a PackagePrefix which is specified in “Layer”, the corresponding entry in JobStep will be matched to this Layer. **Currently there is no constrain checking for duplicates here, take care**! (or write trigger etc.…).

Entries in the view vw\_Folder2Job may override this.

## JobStep and Constrains

If no other information is given, the Master Package assumes that there are no dependencies and thus allows all packages to run in parallel or arbitrary sequence. It orders the start of the packages by the field “StepNo”.

Dependencies are read from the view “vw\_JobStepConstrain”. It implements a m:n self join on JobStep:

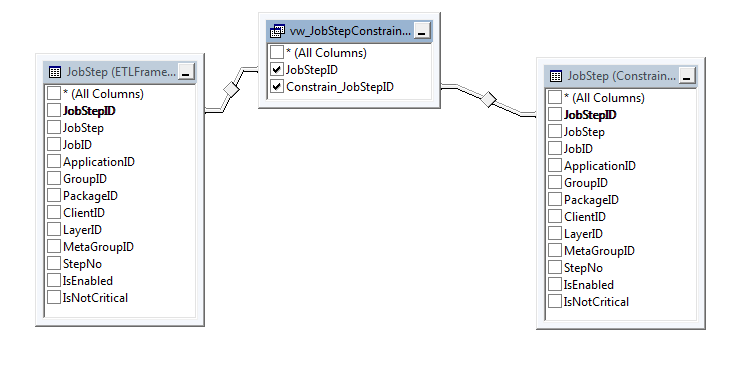


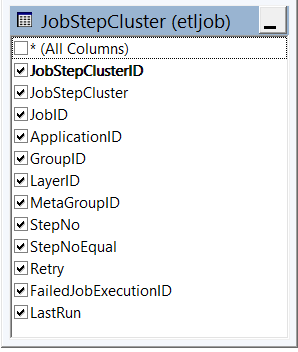
Figure 4: JobStep Dependencies via JobStepConstrain

The Master Package will execute a package only when there is no constraining package which is not executed successfully in the actual job. Constraining packages not part of the current job will be ignored.

## JobStepCluster

The table etljob.JobStepCluster/view etljob.vw\_JobStepCluster contains paramter combinations (used for selecting JobSteps) which were already used for the Masterpackage or added manually. Covered are:

* JobID
* ApplicationID
* GroupID
* LayerID
* MetaGroupID
* StepNo
* StepNoEqual



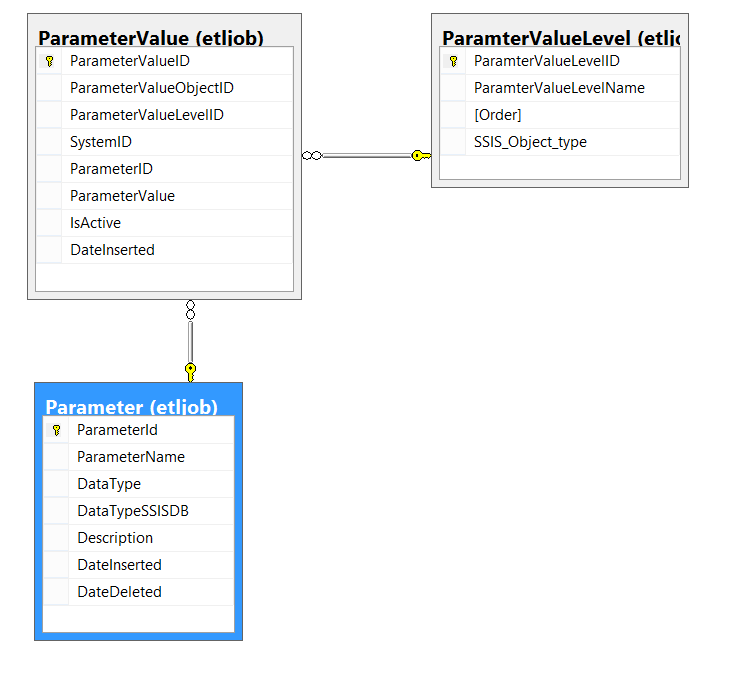
When the Master Package starts it first looks if the combination of the values of these parameters which are used in this current run already has an entry in this table. If yes it just retrieves the corresponding ID, if no it creates a new row with a standard value for the name “JobStepCluster”.

If a run fails it is logged in the field “FailedJobExecutionID”. A run is aborted if the corresponding row has a non Null value in FailedJobExecutionID (i.e. the last run failed) unless Retry is <>0.

## Parameter and ParameterValue

The table “Parameter” contains a list of all parameters imported from SSISDB. The parameters are grouped by name and data type.

The table “ParameterValue” stores the parameter values that are assigned to a package at runtime. Parameter Values can be specified at the level of JobStep, Package, Project, Folder or the Application. This level is specified by the field ParamtereLevelID. The field ParameterValueObjectID points to the actual object (JobStep, Package, Project, Folder or Application) the parameter value applies to.



Use the view vw\_ParameterValuesForJobStep to check which parameter values are assigned to a JobStep (field “Val”). The fields ParameterValue\_JST], [ParameterValue\_PKG], [ParameterValue\_PRJ], [ParameterValue\_FLD] and [ParameterValue\_APP] show on which levels there are possible values for the specific parameter and JobStep. During excecution the parameter with the lowest level (that’s the one from “Val”) is used.

The MetaData Package grabs the standard design values for new entries in the table JobStep, if there are paramters which are not covered by the existing elements of ParamterValue.

**Caution:** Reserved Parameters are handled differently (cf. 3.1).

# Getting the Metadata with “GetMetadata.dtsx”

The second SSIS package in the project is GetMetadata.dtsx. It looks if new packages, new folders etc. are deployed to SSISDB. If yes it updates the infos in the configuration database.

GetMetadata.dtsx executes 6 stored procedures:

* prc\_GetSSISDBFolders imports new folders
* prc\_GetSSISDBProjects imports new projects
* prc\_GetSSISDBPackages imports new packages
* prc\_GetSSISDBParameter imports new parameter
* prc\_SetStandardJobStep creates entries in JobStep for new packages if their project or folder is configured in the table Folder2Job
* prc\_GetSSISDBDefaultParameterValues creates entries in the table ParameterValue, based on the standard design value for this parameter. If the correct value differs, it can be corrected manually.

# Important Stored Procedures & Functions

The Execution Framework uses the namespace Etljob. The central stored procedures for the master package are:

## Package execution

* prc\_InitPackage
* prc\_SetPackageParameter
* fn\_ParameterValuesForJobStep
* prc\_StartPackage

handle the execution of a package

## prc\_GetNextPackage

Gets next package from queue ([Etljob].[vw\_JobStepExecution])

## prc\_InitJobExecution

Puts Packages from [Etljob].[vw\_JobStep] into the queue based on:

* JobID
* ApplicationID
* LayerID
* GroupID
* MetaGroupID
* StepNo
* StepNoEqual

And initializes [Etljob]. [JobExecution]

# Planed Downtime

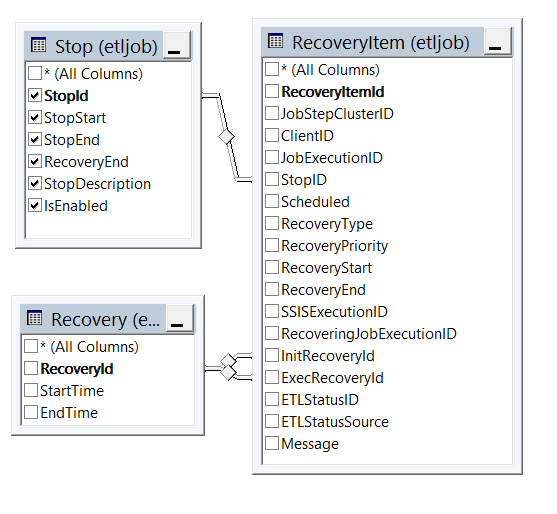
There is the possibility to schedule downtimes during which the master package does not execute packages. The system can be in three states:

* Normal
* Stop
* Recovery

“Stop” means, no packages are executed. Recovery means only those Master Packages explicitly started as a recovery (since they were omitted in a previous stop) are executing child packages.

There further is a package ”RecoverStops.dtsx” which when run checks if there are interrupted or omitted MasterPackage executions and tries to rerun them.

## Tables



There are three tables relevant here (used in fact are again only the accompanying views):

* Stop
* Recovery
* RecoveryItem

## Stop and Recovery

A **Stop** has a Start, an End and a RecoveryEnd. When there is a stop without end or with the current time between its Start and End the system is in Stop Mode. No child packages are executed. The run of the master package only logs its stop into JobExecution by setting StoppedBy to the relevant StopID and StopMode to 1. If the Stop is detected while the Master Package is already executing, the Master Package is “interrupted”. Its current Step will continue but the rest of the steps won’t be. In JobExecution the StopMode is set to 2.

If there is a Stop without RecoveryEnd and the system is not in Stop Mode, the system is in Recovery Mode. Only MasterPackages executed with the parameter StopRecoveryID <> 0 are executing child packages then. Only RecoverStops.dtsx should start the Master Package with a value for StopRecoveryID <> 0. In this case StopRecoveryID is the ID of the Stop responsible for interrupting or omitting the originally run.

Every run of RecoverStops.dtsx creates a new **Recovery**

The table RecoverStops.dtsx checks if there are runs of the Master Packages which were interrupted or omitted (StopMode 2 or 1). Those cases are written to **RecoveryItem**. Interrupted entries get RecoveryPriority 20, omitted 10. The recovery process starts with the entries with the highest priority. Entries with the same priority are ordered by the time they were originally “Scheduled”.

When all entries in RecoveryItem are handled, RecoverStops.dtsx executes AgentJobs of category “Framework” which were scheduled in the time of the stop. Carefull, if the Agent was running during the stop, those Agent Jobs already were handled by the previous steps of the package.

# How To: Executing a package

## Basics

First the basic tables Job, Group, Metagroup, Layer and Application have to be filled.

## Execution prerequisites in the Database

Executed are JobSteps. To be executed, a package has to be referenced by a JobStep. A JobStep can belong to a Job, a Group, a Metagroup, a Layer and an Application.

Specifying values (via paramters) for those when starting the Master Package filters the overall pool of possible JobSteps and specifies which JobSteps should be executed in this run of the master Package.

A package can be referenced by more than one JobStep. Only the Combination of Job, Application, Group, Layer, Package has to be unique.

The parameter values for a package can be configured on the levels:

1. JobStep
2. Package
3. Project
4. Folder
5. Application

The first level found wins

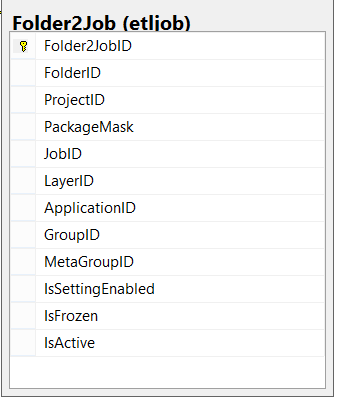
To run a package thus the following data have to be in the database (besides the basic tables Job, Group, Metagroup, Layer and Application):

1. Package (Table etljob.package)
2. JobStep (Table etljob.JobStep) pointing to the package
3. Job, Application, Group, Layer etc. which are used (referenced) in the JobStep (all tables in name space etljob)
4. Values for the packages parameters for this JobStep (Table etljob. ParameterValue)

Step 1 is done by “GetMetadata.dtsx” automatically.

Step 3 has to be done by hand.

Steps 2 & 4 can be done by hand or automatically by “GetMetadata.dtsx” too. For the second case the table etljob. Folder2Job is important:

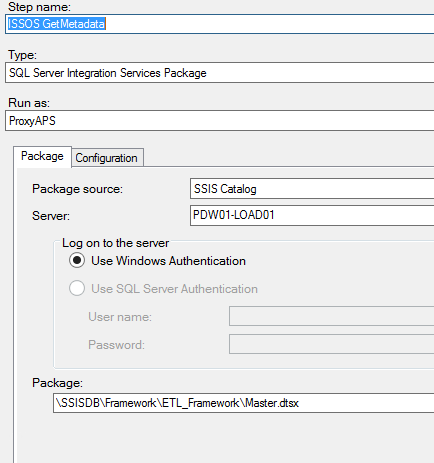


In this table you specify for which packages are imported from SSISDB a JobStep will be created automatically. Either a Folder or a project can be specified. Optionally PackageMask can be used for further filtering (standard string compare with “%” etc.) With IsSettingEnabled you can speciyfy if the JobStep is enabled, IsActive concerns the rule itself.

There can be more than one entry relevant for a package. In this case one row in JobStep per relevant entry will be created.

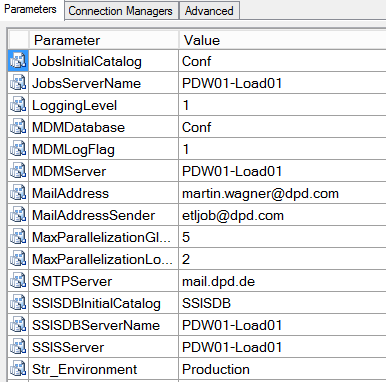
## Executing the Master Package

The Master Package is normally executed via a step of an SQL Server Agent Job.

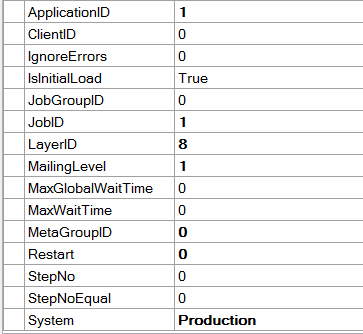


Creating this step the relevant parameter have to be filled. If one of the values for JobID, ApplicationID, GroupID, LayerID, MetaGroupI, StepNo is zero, it is ignored.

There are paramters which are more or less stable like the connections to the SSISDB or the Conf DB with the data of the Framework



And there are specific parameters which specify what JobSteps are to be executed in this special case:



## “Stolpersteine”

* A JobStep has to be enabled
* A parameter value has to be enabled
* If a run failed once it will not run the next time if in JobStepCluster the retry flag is not set to 1 or the field “FailedJobExecutionID” is NULL.
* The collation of the ConfigDB has to be equal to the collation of the SQL Server internal SSISDB.

# Installation

There is a Solution "Job\_Execution\_Framework" with two database projects ("ConfigDB" and "SSISDB") and one SSIS project ("SSIS")

* Deploy "ConfigDB" to a Database (there should be a SSIS Catalog at the server)
* Do NOT deploy SSISDB. It is only needed as reference for the ConfigDB project
* Adjust the connection strings in the SSIS Project and deploy