Dartec Systems

Data Quality Engine (DQE)

Installation Guide

Table of Contents

1 Overview

This document describes the steps required to setup the Data Quality Engine (DQE).

This document does not describe the design or technical aspects of the DQE; this information is available within …\<*VersionNo*>\DQEReleaseDocumentation\DQE_TechnicalOverview.pdf

This document does not describe the use or operating instructions for DQE; this information is available within …\<*VersionNo*>\DQEReleaseDocumentation\DQE_UserGuide.pdf

A short presentation that provides an overview of DQE can be found within …\<*VersionNo*>\DQEReleaseDocumentation\DQE Overview.pdf

1.1 Understanding the Deployment Folder

The deployment folder consists of two folders and this file.

- DQE Code Deployment: Contains all code and file that will be needed to complete this installation
- DQEReleaseDocumentation: Contains three documents
    - Technical Overview that describes the structure of the code and system
    - User Guide that provides useful operating information
    - DQE Overview presentation that provides an easy to digest overview of the DQE

1.2 Installation summary

To carry out the installation of DQE you will need permissions to restore databases, configure scheduled jobs, submit packages to the SSIS Catalog and deploy models to your MDS server.

There are four components to DQE installation

- Database Project
    - SQL Agent Job
- SSIS Project
    - SSIS Catalog

        - Variables

- MDS Model
- SSRS Reports

2 Database Project

A SQL Server bak file is included within the deployment folder. The database can either be restored directly from this file or deployed using the source scripts contained in 1 Database Deployment.

Restore the database DataQualityDB.bak file to the server where you intend to host DQE.

The bak file can be found within the DQEDeployment folder at the below path.

…\<*VersionNo*>\DQE Initial Deployment\9 Database BAK file \DataQualityDB.bak

The database restoration you run will be specific to your environment (i.e. where you store the Deployment folder and where your mdf and ldf files reside- highlighted in yellow), the below is included only for illustration purposes.

```
USE [master]
RESTORE DATABASE [DataQualityDB]
FROM  DISK = N'C:\Downloads\DQEDeployment\<VersionNo>\DQE Code Deployment\9
Database BAK File\DataQualityDB.bak'
WITH  FILE = 1,
MOVE N'DataQualityDB' TO N'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\DataQualityDB.mdf',
MOVE N'DataQualityDB_log' TO N'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\DataQualityDB_log.ldf',
NOUNLOAD,  REPLACE,  STATS = 5
GO
```

3 SSIS Project

The following steps will deploy the DQE SSIS project to the SSIS catalog.

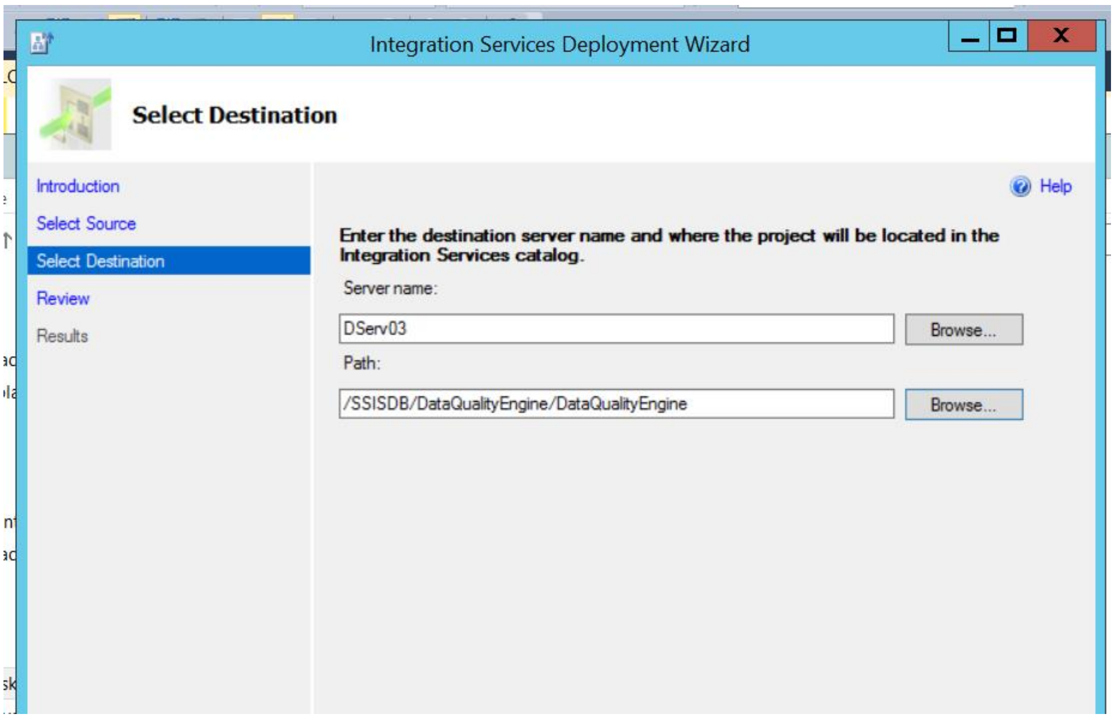
Prerequisite: Before proceeding the SSIS catalog must exist on your server before running the below steps.

1. Run the below script to create the SSIS Folder

```
USE [SSISDB];
GO

DECLARE @FolderName SYSNAME = N'DataQualityEngine';

IF NOT EXISTS (SELECT * FROM catalog.folders where folder_id = (SELECT
F.folder_id FROM [catalog].folders AS F WHERE F.name = @FolderName))
begin
       EXEC [SSISDB].[catalog].[create_folder] @FolderName
end
```

2. Run the ISPAC by double clicking it. The ISPAC can be found within …
   \<VersionNo>\DQE Code Deployment\1 SSIS Deployment\ISPAC\
   DataQualityEngine.ispac

   The solution should be stored within the SSIS catalog folder created above.



Enter the destination server name and where the project will be located in the Integration Services catalog.

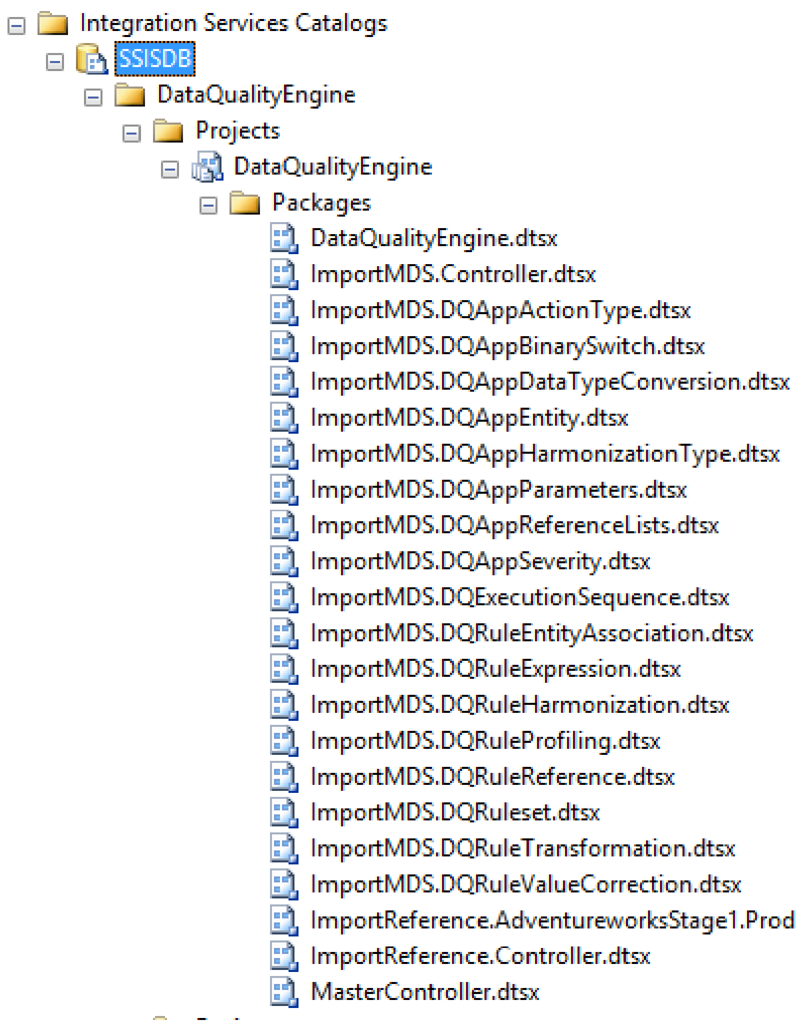
Server name:

DServ03

Path:

/SSISDB/DataQualityEngine/DataQualityEngine

Once done the SSIS project should appear within the DataQualityEngine folder (see below)



Integration Services Catalogs
SSISDB
DataQualityEngine
Projects
DataQualityEngine
Packages
DataQualityEngine.dtsx
ImportMDS.Controller.dtsx
ImportMDS.DQAppActionType.dtsx
ImportMDS.DQAppBinarySwitch.dtsx
ImportMDS.DQAppDataTypeConversion.dtsx
ImportMDS.DQAppEntity.dtsx
ImportMDS.DQAppHarmonizationType.dtsx
ImportMDS.DQAppParameters.dtsx
ImportMDS.DQAppReferenceLists.dtsx
ImportMDS.DQAppSeverity.dtsx
ImportMDS.DQExecutionSequence.dtsx
ImportMDS.DQRuleEntityAssociation.dtsx
ImportMDS.DQRuleExpression.dtsx
ImportMDS.DQRuleHarmonization.dtsx
ImportMDS.DQRuleProfiling.dtsx
ImportMDS.DQRuleReference.dtsx
ImportMDS.DQRuleset.dtsx
ImportMDS.DQRuleTransformation.dtsx
ImportMDS.DQRuleValueCorrection.dtsx
ImportReference.AdventureworksStage1.Prod
ImportReference.Controller.dtsx
MasterController.dtsx

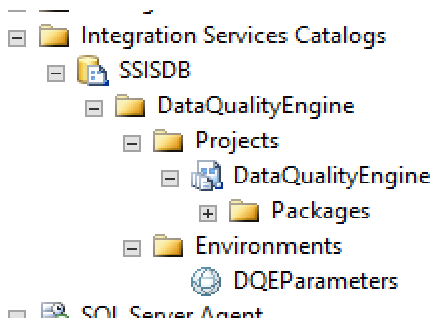
3. The below script will create the environment file and parameters within the DataQualityEngine project.

   NOTE: You will need to edit the parameters at the top of the script to define **your** Server and Master Data Services database names (highlighted in YELLOW).

```
USE [SSISDB];
GO

-- CONFIGURATION - GLOBAL DEFINITIONS - set the environment
configurations here
DECLARE @EnvironmentName SYSNAME = N'DQEParameters';
DECLARE @FolderName SYSNAME = N'DataQualityEngine';
DECLARE @ProjectName SYSNAME = N'DataQualityEngine';
DECLARE @MDSSERVER SYSNAME = N'DSERV03'; -- EDIT: Your MDS Server
Instance
DECLARE @MDSDB SYSNAME = N'MDSDB'; -- EDIT: Your MDS Database name
DECLARE @DataQualityServer SYSNAME = N'DServ03'; -- EDIT: The Server
you have installed your DataQualityDB
DECLARE @DataQualityDB SYSNAME = N'DataQualityDB';
DECLARE @AuditServer SYSNAME = N'DServ03';-- EDIT: The Server you have
installed your DataQualityDB
DECLARE @AuditDB SYSNAME = N'DataQualityDB';

-- get the ids for the configure names
-- no need to touch this code
DECLARE @FolderID BIGINT = (SELECT F.folder_id FROM [catalog].folders
AS F WHERE F.name = @FolderName);
DECLARE @ProjectID BIGINT = (SELECT P.project_id FROM
[catalog].projects AS P WHERE P.name = @ProjectName AND P.folder_id =
@FolderID);

-- CONFIGURATION - PARAMETERS
-- this table will hold all environment variables to be defined and
mapped
-- there are two ways to go about defining what needs to be mapped, so
check out the options below
DECLARE @Variables TABLE
(
    MyKey INT IDENTITY (1, 1) NOT NULL PRIMARY KEY CLUSTERED, /*
Iteration Index, don't worry about this. */

    EnvironmentVariableName     SYSNAME         NOT NULL,       /*
Name of the new environment variable. */
    EnvironmentVariableValue    SQL_VARIANT     NOT NULL,       /* Use
N' (unicode) notation or CAST for strings to ensure you get an
NVARCHAR. Use CAST with numeric values to ensure you get the correct
type. */
    EnvironmentVariableType     SYSNAME         NOT NULL,       /* The
SSIS Variable Type (String, Int32, DateTime...) */
    TargetObjectType            SYSNAME         NOT NULL,       /*
What to map this variable to. Use "Project" or "Package" */
    TargetObjectName            SYSNAME         NOT NULL,       /*
When mapping to a project use the Project Name. When mapping to a
package use the Package Name. */
    TargetParameterName         SYSNAME         NOT NULL,       /*
Name of the parameter to map the environment variable to. */

    /* Duplicate environment variable names are not allowed. */
    UNIQUE (EnvironmentVariableName)
)

BEGIN

    INSERT INTO @Variables (EnvironmentVariableName,
EnvironmentVariableValue, EnvironmentVariableType, TargetObjectType,
TargetObjectName, TargetParameterName)
    VALUES
       ('DataQualityDB_ConnectionString' ,'Data Source=' +
@DataQualityServer + ';Initial Catalog='+ @DataQualityDB +';Integrated
Security=SSPI;'
,'String','Project',@ProjectName,'DataQualityDB_ConnectionString'),
       ('MDSDB_ConnectionString' ,'Data Source=' + @MDSSERVER +
';Initial Catalog=' + @MDSDB + ';Provider=SQLNCLI11.1;Integrated
Security=SSPI;Auto Translate=False'
,'String','Project',@ProjectName,'MDSDB_ConnectionString'),
       ('AuditDB_ConnectionString' ,'Data Source=' + @AuditServer +
';Initial Catalog='+ @AuditDB +';Provider=SQLNCLI11.1;Integrated
Security=SSPI;Auto Translate=False'
,'String','Project',@ProjectName,'AuditDB_ConnectionString'),
       ('DomainName','DQEDomain','String','Project',@ProjectName,'Doma
inName'),
       ('ParentLoadId', '0'
,'Int32','Project',@ProjectName,'ParentLoadId'),
       ('RuleEntityAssociationCode','0','String','Project',@ProjectNam
e,'RuleEntityAssociationCode')
    ;
END

-- END OF CONFIGURATION - everything below should work on its own

-- create the environment now
IF NOT EXISTS (SELECT * FROM catalog.environments WHERE name =
@EnvironmentName AND folder_id =@FolderID)
BEGIN
    EXECUTE [catalog].[create_environment]
        @environment_name = @EnvironmentName,
        @environment_description= N'',
        @folder_name = @FolderName;
END
DECLARE @EnvironmentID INT = (SELECT environment_id FROM
[catalog].environments WHERE name = @EnvironmentName AND folder_id
=@FolderID);


-- loop variables
DECLARE @VariableKey INT, @LastVariableKey INT;
DECLARE @VariableName SYSNAME, @VariableValue SQL_VARIANT,
@VariableType SYSNAME, @TargetObjectType SYSNAME, @TargetObjectName
SYSNAME, @TargetParameterName SYSNAME;

-- create all the variables in the environment now
SET @VariableKey = 1;
SET @LastVariableKey = (SELECT MAX(MyKey) FROM @Variables);
WHILE (@VariableKey <= @LastVariableKey)
BEGIN

    SELECT
        @VariableName = EnvironmentVariableName,
        @VariableValue = EnvironmentVariableValue,
        @VariableType = EnvironmentVariableType
    FROM
        @Variables
    WHERE
        MyKey = @VariableKey;

       if @VariableType <> 'int32'
       begin
    IF NOT EXISTS (SELECT * FROM [catalog].environment_variables AS V
WHERE V.name = @VariableName AND V.environment_id = @EnvironmentID)
    BEGIN
        EXEC [catalog].[create_environment_variable]
            @variable_name = @VariableName,
            @sensitive = False,
            @description = N'',
            @environment_name = @EnvironmentName,
            @folder_name = @FolderName,
            @value = @VariableValue,
            @data_type = @VariableType
    END
       end

       if @VariableType = 'int32'
       begin
    IF NOT EXISTS (SELECT * FROM [catalog].environment_variables AS V
WHERE V.name = @VariableName AND V.environment_id = @EnvironmentID)
    BEGIN
              declare @intVariableValue INT
              SET @intVariableValue = CAST (@VariableValue AS INT)
        EXEC [catalog].[create_environment_variable]
            @variable_name = @VariableName,
            @sensitive = False,
            @description = '',
            @environment_name = @EnvironmentName,
            @folder_name = @FolderName,
            @value = @intVariableValue,
            @data_type = @VariableType
       END
       end


    SET @VariableKey += 1;

END

-- associate the project with the environment
-- this will be done with a relative reference as the Environment is
in the same folder as the Project
DECLARE @ReferenceID BIGINT = NULL;
IF NOT EXISTS (SELECT R.reference_id FROM
[catalog].environment_references AS R WHERE R.environment_folder_name
IS NULL AND R.environment_name = @EnvironmentName AND R.project_id =
@ProjectID)
BEGIN
    EXECUTE [catalog].[create_environment_reference]
        @environment_name = @EnvironmentName,
        @reference_id = @ReferenceID OUTPUT,
        @project_name = @ProjectName,
        @folder_name = @FolderName,
        @reference_type = R
END

-- associate every Environment Variable with the proper parameter in
the Project
SET @VariableKey = 1;
SET @LastVariableKey = (SELECT MAX(MyKey) FROM @Variables);

WHILE @VariableKey <= @LastVariableKey
BEGIN

    SELECT
        @VariableName = V.EnvironmentVariableName,
        @TargetObjectType = V.TargetObjectType,
        @TargetObjectName = V.TargetObjectName,
        @TargetParameterName = V.TargetParameterName
    FROM
```

```
        @Variables AS V
    WHERE
        MyKey = @VariableKey;

    DECLARE @ObjectTypeCode SMALLINT = (CASE WHEN @TargetObjectType =
'Project' THEN 20 ELSE 30 END); /* 20 = "Project Mapping", 30 =
"Package Mapping" */

    IF NOT EXISTS (SELECT * FROM [catalog].object_parameters AS P
WHERE
        P.project_id = @ProjectID AND
        P.object_type = @ObjectTypeCode AND
        P.object_name = @TargetObjectName AND
        P.parameter_name = @TargetParameterName AND
        P.value_type = 'R' AND /* R = Referenced, V = Value */
        P.referenced_variable_name = @VariableName)
    BEGIN

        EXECUTE [catalog].[set_object_parameter_value]
            @object_type = @ObjectTypeCode,
            @parameter_name = @TargetParameterName,
            @object_name = @TargetObjectName,
            @folder_name = @FolderName,
            @project_name = @ProjectName,
            @value_type = R,
            @parameter_value = @VariableName
    END


    SET @VariableKey += 1;
END
```

Once this script has completed you should be able to view an environments file within the DataQualityEngine of your SSIS catalog



The environment file should be configured with six separate variables.

4. Run the below script to create the scheduled job that will be used to invoke the DataQualityEngine code.

NOTE: Before progressing, please consider Section 4 and evaluate the context under which you will execute the DataQualityEngine scheduled job. If you decide to implement the Proxy account please ensure the line within the sp_add_jobstep is uncommented.

NOTE: The use of a scheduled job is not ideal but is needed to overcome issues of handing through credential pass-through found with some of our clients.

The below code will create the scheduled job and link the scheduled job to the project parameters defined in the previous step.

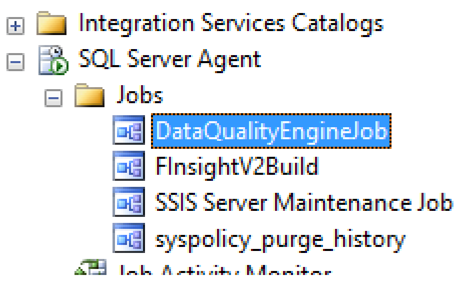
NOTE: You will need to edit the parameters at the top of the script to define **your** SSIS Server name (highlighted in YELLOW).

NOTE: The account used to run this scheduled job will need to be able to access the database hosting your MDS database, the DataQualityDB, any locations where you decide to output cleansed data, the SSIS Catalog and your source databases. Given this you may decide to run the job under some form of proxy account.

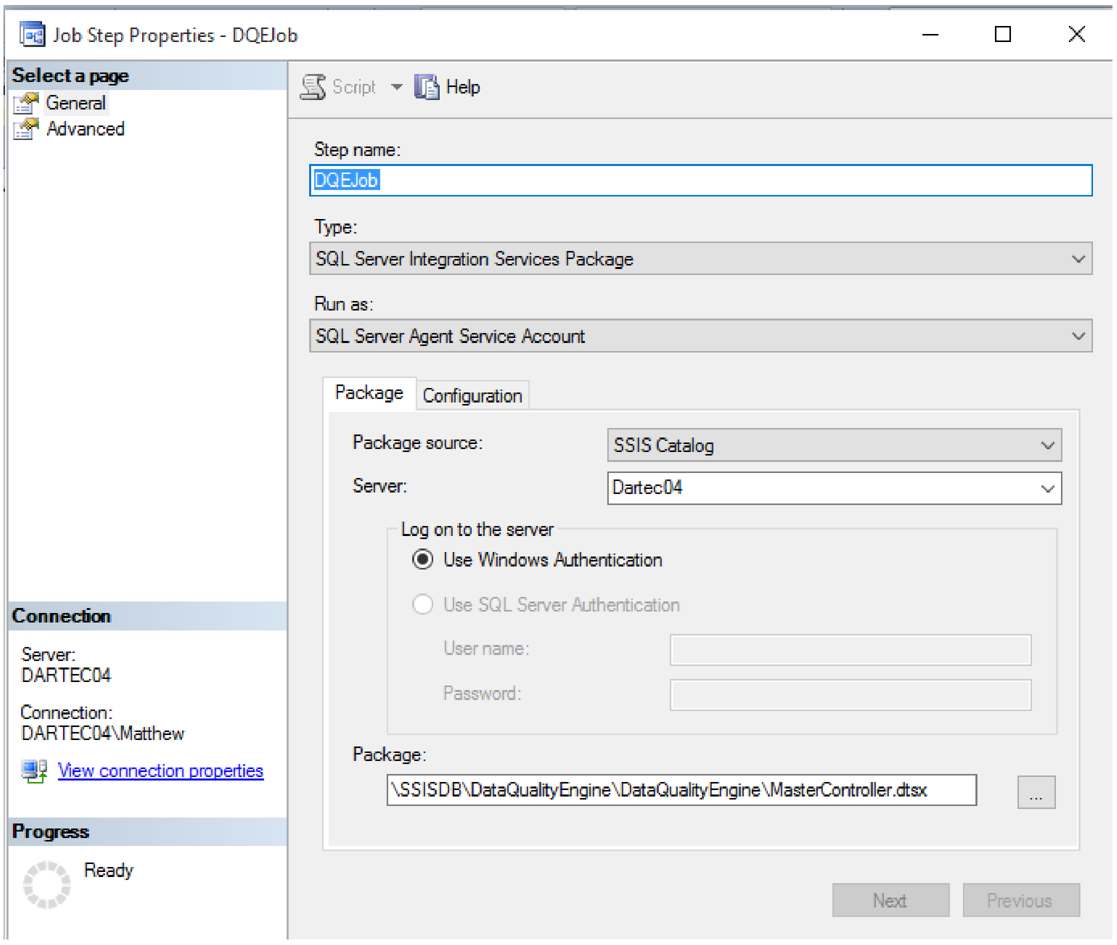
```
USE [msdb]
GO

DECLARE @command VARCHAR (4000)
, @SSISServerName VARCHAR (255)
, @SSISEnvironmentId VARCHAR (10)

/* Set parameres*/
-- Name of the SSIS server that hosts the SSIS project
SET @SSISServerName = 'DServ03'
--Get the environment ID
SELECT @SSISEnvironmentId = E.reference_id FROM SSISDB.[catalog].folders F
JOIN SSISDB.[catalog].projects P
ON P.folder_id = F.folder_id JOIN SSISDB.catalog.environment_references E ON
P.project_id = E.project_id
WHERE E.environment_name = 'DQEParameters' AND F.Name = 'DataQualityEngine'
AND e.environment_folder_name IS NULL

SET @command = '/ISSERVER
"\"\SSISDB\DataQualityEngine\DataQualityEngine\MasterController.dtsx\""
/SERVER "' + @SSISServerName + '" /ENVREFERENCE '+@SSISEnvironmentId+' /Par
"\"$ServerOption::LOGGING_LEVEL(Int16)\"";1 /Par
"\"$ServerOption::SYNCHRONIZED(Boolean)\"";True /CALLERINFO SQLAGENT
/REPORTING E'
PRINT @command


IF EXISTS (SELECT 1 FROM msdb..sysjobs WHERE name = 'DataQualityEngineJob')
BEGIN
EXEC msdb..sp_delete_job
    @job_name = N'DataQualityEngineJob' ;
END

/****** Object:  Job [DataQualityEngineJob]    Script Date: 23/02/2016
14:28:19 ******/
BEGIN TRANSACTION
DECLARE @ReturnCode INT
SELECT @ReturnCode = 0
```

```
/****** Object:  JobCategory [[Uncategorized (Local)]]    Script Date:
23/02/2016 14:28:19 ******/
IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE
name=N'[Uncategorized (Local)]' AND category_class=1)
BEGIN
EXEC @ReturnCode = msdb.dbo.sp_add_category @class=N'JOB', @type=N'LOCAL',
@name=N'[Uncategorized (Local)]'
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)
EXEC @ReturnCode =  msdb.dbo.sp_add_job @job_name=N'DataQualityEngineJob',
              @enabled=1,
              @notify_level_eventlog=0,
              @notify_level_email=0,
              @notify_level_netsend=0,
              @notify_level_page=0,
              @delete_level=0,
              @description=N'No description available.',
              @category_name=N'[Uncategorized (Local)]',
              @owner_login_name=N'sa', @job_id = @jobId OUTPUT
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
/****** Object:  Step [DQEJob]    Script Date: 23/02/2016 14:28:20 ******/
EXEC @ReturnCode = msdb.dbo.sp_add_jobstep @job_id=@jobId,
@step_name=N'DQEJob',
              @step_id=1,
              @cmdexec_success_code=0,
              @on_success_action=1,
              @on_success_step_id=0,
              @on_fail_action=2,
              @on_fail_step_id=0,
              @retry_attempts=0,
              @retry_interval=0,
              @os_run_priority=0, @subsystem=N'SSIS',
              @command=@command,
              @database_name=N'DataQualityDB',
              @flags=0
              --@proxy_name=N'DQEExecuterSSISProxy' -- SEE SECTION 4 TO
DETERMINE WHETHER TO USE A PROXY AND HOW TO CONFIGURE IT
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
EXEC @ReturnCode = msdb.dbo.sp_update_job @job_id = @jobId, @start_step_id =
1
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
EXEC @ReturnCode = msdb.dbo.sp_add_jobserver @job_id = @jobId, @server_name =
N'(local)'
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
COMMIT TRANSACTION
GOTO EndSave
QuitWithRollback:
    IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION
EndSave:

GO
```

Once done a new job named ‘DataQualityEngineJob’ will have been created within the SQL Server Agent.


Integration Services Catalogs
SQL Server Agent
Jobs
DataQualityEngineJob
FInsightV2Build
SSIS Server Maintenance Job
syspolicy_purge_history

This job will be configured to call the DQE SSIS Master package.


Job Step Properties - DQEJob

Select a page
General
Advanced

Connection
Server:
DARTEC04
Connection:
DARTEC04\Matthew
View connection properties

Progress
Ready

Script Help

Step name:
DQEJob

Type:
SQL Server Integration Services Package

Run as:
SQL Server Agent Service Account

Package | Configuration

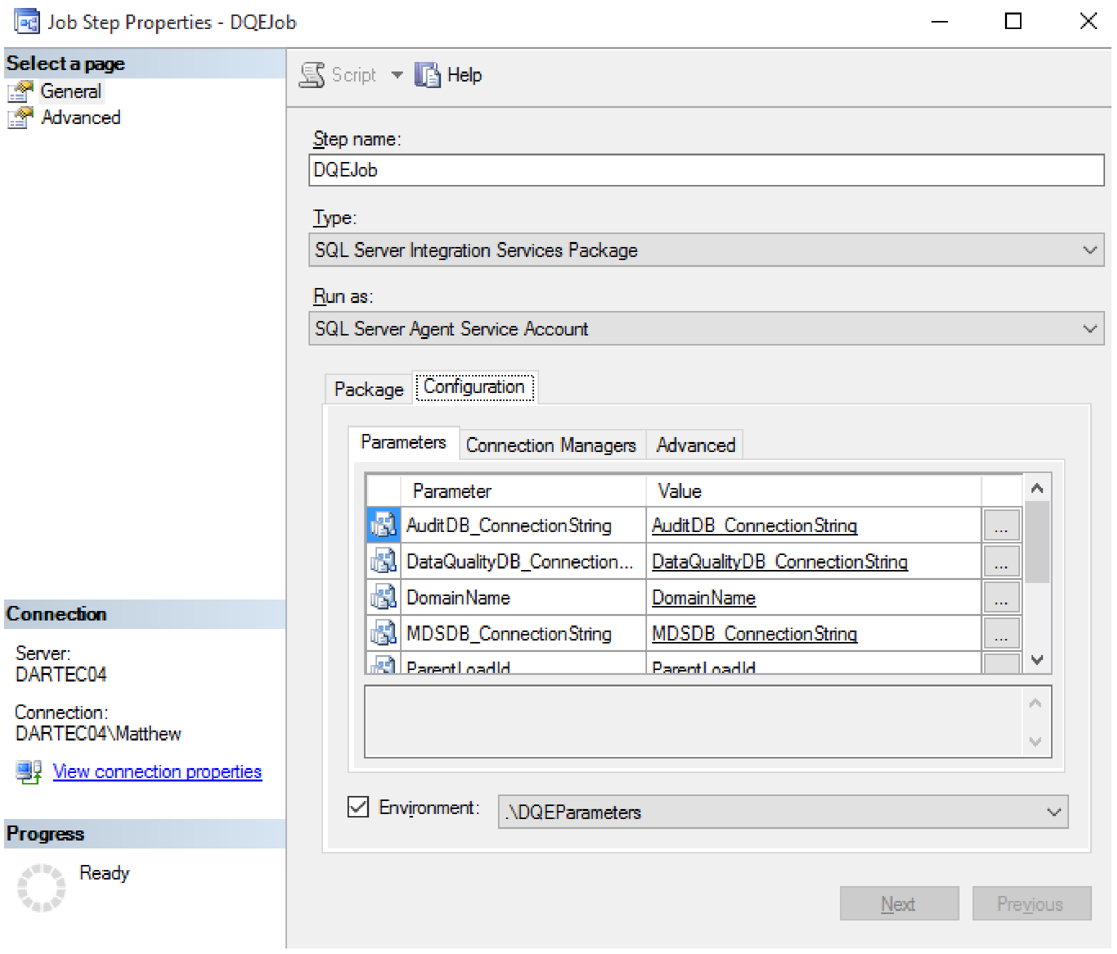
Package source: SSIS Catalog
Server: Dartec04

Log on to the server
Use Windows Authentication [✓]
Use SQL Server Authentication
User name:
Password:

Package:
\SSISDB\DataQualityEngine\DataQualityEngine\MasterController.dtsx

Next Previous

The job will also be configured to make use of the environment variables defined in the previous steps.



Server: DARTEC04

Connection: DARTEC04\Matthew

Step name: DQEJob

Type: SQL Server Integration Services Package

Run as: SQL Server Agent Service Account

Package | Configuration

Parameters | Connection Managers | Advanced

Parameter	Value
AuditDB_ConnectionString	AuditDB_ConnectionString
DataQualityDB_Connection...	DataQualityDB_ConnectionString
DomainName	DomainName
MDSDB_ConnectionString	MDSDB_ConnectionString
ParentLoadId	ParentLoadId

☑ Environment: .\DQEParameters

At this point the database, SSIS project and scheduled job components will be available for use.

The next step creates the MDS model that is the ‘front-end’ for the DQE.

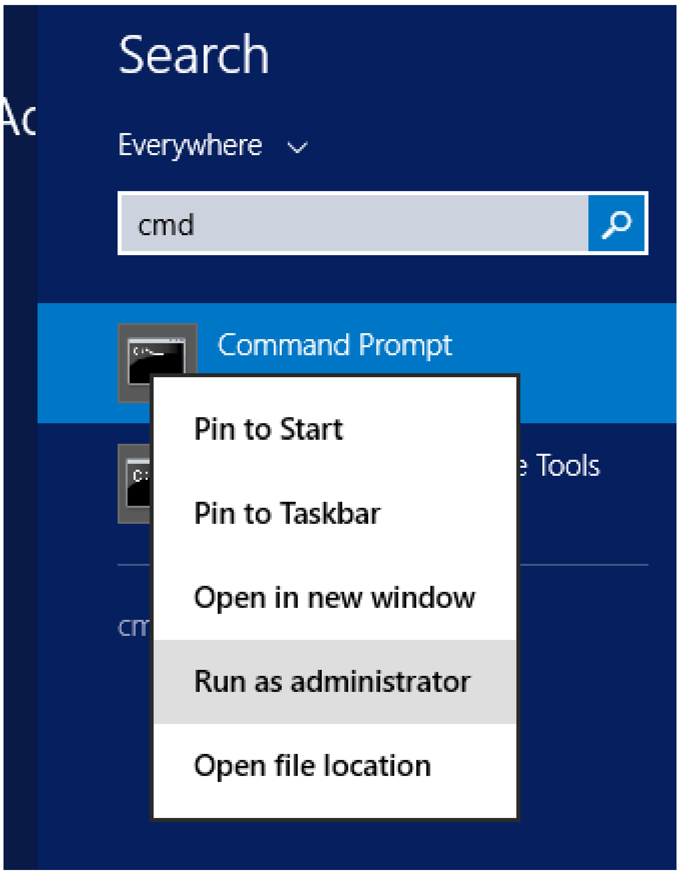
3.1 MDS Model

To install this file you will need to copy the file to and run the scripts from your MDS server.

The MDS model can be found in …\<VersionNo>\DQE Code Deployment\2 MDS Deployment\DataQuality.pkg

Copy the DataQuality.pkg file to your MDS server. Once copied run the below command to ‘install’ the MDS model.

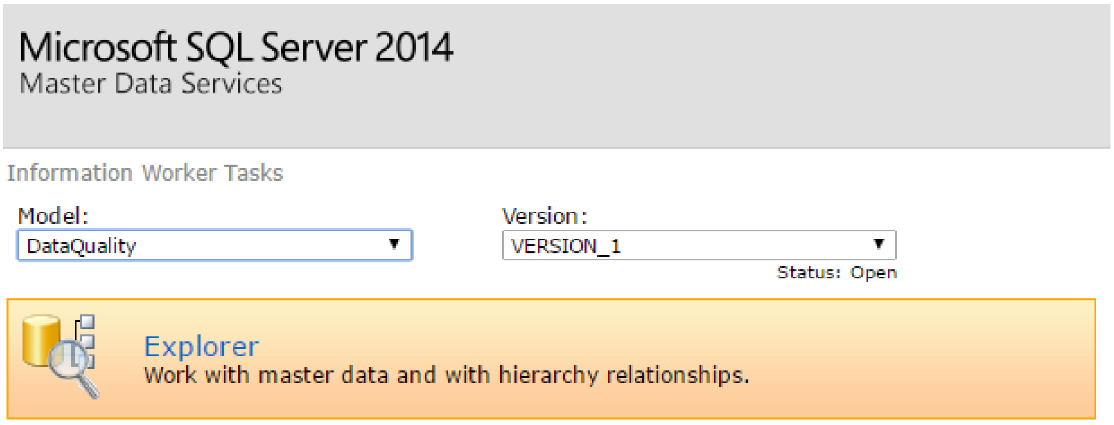
Open the Command line on your MDS server (remember, open as Administrator)



NOTE: The version of SQL Server will affect what folder the MDSDeployDeploy.exe is found within. Based on your environment you may need to edit the text heighted in yellow.

```
REM Navigate to the folder that contains
MDSModelDeploy.exe

CD C:\Program Files\Microsoft SQL Server\110\Master
Data Services\Configuration

REM Deploy the Data Quality Model

MDSModelDeploy deployclone -service MDS1 -package
"C:\Downloads\<VersionNo>\DQE Code Deployment\2 MDS
Deployment\DataQuality.pkg"
```

Once the model is deployed you should see the DataQuality model appear within the MDS GUI,


Microsoft SQL Server 2014
Master Data Services

Information Worker Tasks

Model: DataQuality

Version: VERSION_1

Status: Open

Explorer
Work with master data and with hierarchy relationships.

Before finishing you will need to validate the DataQuality model within SQL Management studio. To do this you will need to run the below script on the server that hosts your MDS database. This is done using the below script.

Note: You will need to ensure the below script is running within the context of your MDS database (highlighted in yellow)

```
USE MDSDB
GO


DECLARE @ModelName NVARCHAR(50) = 'DataQuality'
DECLARE @Model_Id INT
--DECLARE @UserName NVARCHAR(50) = 'DSERV03\Administrator'
DECLARE @UserName NVARCHAR(50) = 'Domain\User'
DECLARE @User_Id INT
DECLARE @Version_ID INT

SET @User_Id = (SELECT ID FROM mdm.tbluser u WHERE u.UserName = @UserName)
SET @Model_Id = (SELECT TOP 1 model_id FROM mdm.viw_system_schema_version
WHERE Model_Name = @modelname)
SET @Version_ID = (SELECT MAX(ID) FROM mdm.viw_system_schema_version WHERE
Model_ID = @Model_Id)

EXEC mdm.udpValidateModel @user_id, @Model_ID, @Version_ID, 1
```

3.2 SSRS Project

A basic set of reports have been included as an SSRS solution to illustrate the reporting possibility.

The SSRS reports project can be found within

…\VersionNo\DQE Code Deployment\3. SSRS Source\DQEReports\DQEReports

4 A note on access and permissions

The DQE scheduled job (DataQualityEngineJob) will need to run under a context that has a number of permissions. There are a wide variety of permissions that you may choose to assign to a specific SQL Login or Windows Service account.

If you opt to run the DQE under the context of an account other than your standard SQL Agent service account, you will need to create a SQL Server credential and proxy account. The DataQualityEngineJob scheduled job can then be configured to operate using this proxy (see the create SQL Agent job script in section 3).

As well as requiring access to any source databases or tables you would like DQE to get its data from, there are a number of other permissions that need to be granted

- DataQualityDB: It is assumed that the account will have full access to this database since the job will be reading and writing data as well as creating and dropping database objects.
- MSDB
- Master
- SSISDB

A set of sample permission scripts are included below. These are the minimum permissions that the account used to create the SQL Credential will require. The [Account] will need to be replaced by the name of your local account.

```
/************************************************/
use msdb;
CREATE USER [Account] FOR LOGIN [Account] WITH DEFAULT_SCHEMA=[dbo]
GRANT SELECT ON DBO.sysjobs TO [Account]
GRANT SELECT ON DBO.sysjobhistory TO [Account]
GRANT EXECUTE ON dbo.sp_start_job TO [Account]
EXEC sp_addrolemember @rolename = 'SQLAgentOperatorRole', @membername =
'Account'

use master;
CREATE USER [Account] FOR LOGIN [Account] WITH DEFAULT_SCHEMA=[dbo]
GRANT EXECUTE ON master.dbo.xp_sqlagent_enum_jobs TO [Account]
GO

Use ssisdb;
CREATE USER [Account] FOR LOGIN [Account] WITH DEFAULT_SCHEMA=[dbo]
GRANT SELECT ON SSISDB.[catalog].folders TO [Account]
GRANT SELECT ON SSISDB.[catalog].environments TO [Account]
GRANT SELECT ON SSISDB.[internal].[environment_variables]  TO [Account]
GRANT UPDATE ON SSISDB.[internal].[environment_variables]  TO [Account]

-- Grant Access to SSIS folders,Projects and Environments
```

```sql
/* Create SSIS permission Parameters */
DECLARE @principleId INT
DECLARE @folderId INT
DECLARE @projectId INT
DECLARE @environmentId INT

/* Set SSIS permission Parameters */
SELECT @principleId = principal_id FROM SSISDB.sys.database_principals WHERE
[name] = '[Account]'
SELECT @folderId = folder_id FROM SSISDB.[catalog].[folders] WHERE [name] =
'DataQualityEngine'
SELECT @projectId = [project_id] FROM ssisdb.[catalog].[projects] WHERE
folder_id = @folderId
SELECT @environmentId = [environment_id] FROM ssisdb.[catalog].[environments]
WHERE folder_id = @folderId

-- Read and Execute on Folder
EXEC [SSISDB].[catalog].[grant_permission] @object_type=1,
@object_id=@folderId, @principal_id=@principleId, @permission_type=103
EXEC [SSISDB].[catalog].[grant_permission] @object_type=1,
@object_id=@folderId, @principal_id=@principleId, @permission_type=1
-- Read and Execute on Project
EXEC [SSISDB].[catalog].[grant_permission] @object_type=2,
@object_id=@projectId, @principal_id=@principleId, @permission_type=1
-- Read on Enviroment
EXEC [SSISDB].[catalog].[grant_permission] @object_type=3,
@object_id=@environmentId, @principal_id=@principleId, @permission_type=1
/* Background information
-- Object_type: Securable objects types include folder (1), project (2),
environment (3), and operation (4).
-- OBJECT_ID: Depends on the type of object
              -- Type 1: select * from ssisdb.[catalog].[folders]
              -- Type 2: SELECT [project_id] FROM ssisdb.[catalog].[projects]
              -- Type 3: SELECT [environment_id] FROM
ssisdb.[catalog].[environments]
              -- Type 4: SELECT *, Operation_id FROM
ssisdb.[catalog].[operations]
-- principal_id: select * from SSISDB.sys.database_principals
-- Permission Types : https://msdn.microsoft.com/en-us/library/ff878149.aspx

*/
```

Once permissions have been granted to an account, that account can be used to create a SQL credential and proxy account.

```sql
/* Create credential and proxy accounts*/
USE msdb
GO

CREATE CREDENTIAL DQEExecuter WITH IDENTITY = '[Account]',
  SECRET = '<EnterStrongPasswordHere>';
GO

-- creates proxy "DQEExecuterSSISProxy" and assigns the credential
'DQExecuter' to it.
```

```
EXEC dbo.sp_add_proxy
    @proxy_name = 'DQEExecuterSSISProxy',
    @enabled = 1,
    @description = 'Executes the DQE.',
    @credential_name = 'DQEExecuter' ;
GO
-- grants the proxy " DQEExecuterSSISProxy " access to the SSIS subsystem.
EXEC dbo.sp_grant_proxy_to_subsystem
    @proxy_name = N'DQEExecuterSSISProxy',
    @subsystem_id = 11 ;
GO

/* Configure the DataQualityEngine Scheduled Job to use
DQEExecuterSSISProxy*/
```

5 Testing that it all works

Run the below stored procedure to run a single simple rule

```
USE DataQualityDB
GO

EXEC [DQ].[sExecuteJobDataQualityEngine]
@JobName = 'DataQualityEngineJob'
, @DQDomainName = 'AdventureWorksStage1'
, @ProjectName = 'DataQualityEngine'
, @FolderName = 'DataQualityEngine'
, @EnvironmentName = 'DQEParameters'
, @RuleEntityAssociationCode = 21 -- Optional, the stand-alone code of the
rule you want to run
```

If everything has been installed correctly (and the scheduled job account used has permissions to access to all of the resources) you should get an output like the one below

```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
Job 'DataQualityEngineJob' started successfully.
JobDataQualityEngineJob completed successfully.

(1 row(s) affected)
```

To monitor the progress of the DQE job or review is success statuses on completion, use the below queries

```
/* The base auditing table*/
SELECT *
FROM [Audit].[RoutineLoad]
ORDER BY LoadId DESC

/* The enriched auditing table */
SELECT *
FROM [Audit].[RoutineLoadHistory]
ORDER BY LoadId DESC
```

A full description of auditing questions and mechanisms can be found in …*<VersionNo>*\DQEReleaseDocumentation\DQE_UserGuide_V3.pdf