BISSANTZ

Datenladen auf Anforderung – eine Übersicht

Autor	Bernd Werther
Datum der Veröffentlichung	17.08.2018

Bissantz & Company GmbH
Nordring 98 90409 Nürnberg
T +49 911 935536 - 0
F +49 911 935536 - 10
service@bissantz.de

Geschäftsführer
Dr. Nicolas Bissantz
Michael Westphal

USt-IdNr. DE181064258
Steuer-Nr. 241/118/25048
Amtsgericht Nürnberg
HRB 15590

Sparkasse Nürnberg
IBAN DE88 7605 0101 0570 1792 91
BIC SSKNDE77XXX

www.bissantz.de

Abstract

Dieser Blog beschäftigt sich mit der Frage, wie Anwender in DeltaMaster Daten aktualisieren können, ohne auf nächtlich laufende Aktualisierungs-Jobs warten zu müssen.

Dabei werden zunächst zwei klassische Ansätze über SQL Server-Agent-Berechtigungen gezeigt. Anschließend wird ein trickreicher Ansatz beschrieben, bei dem die Lade-Jobs durch benutzerdefinierte Warnmeldungen getriggert werden können und als letzter Lösungsvorschlag wird die Umsetzung mittels Integration-Services-Katalog beschrieben.

Bei allen vier Ansätzen wird zudem auf die jeweiligen Vor- und Nachteile eingegangen.

Datenladen auf Anforderung – eine Übersicht

In Kundenprojekten fragen Anwender häufig danach, wie sie die Daten selbstständig aktualisieren können, wenn sie nicht bis zum nächsten automatisierten Datenladevorgang warten möchten.

Die Gründe dafür können vielschichtig sein, in der Regel gehen die Anwender aber wohl davon aus, dass sich die Daten, seien es Stamm- oder Bewegungsdaten, in irgendeiner Form geändert haben. Solche Veränderungen sollen natürlich auch möglichst schnell in einer Anwendung sichtbar werden.

Für diesen Blog gehen wir davon aus, dass der Anwender die Aktualisierung direkt über die DeltaMaster CustomApp aus einer DeltaMaster-Anwendung heraus anstoßen soll und somit die Nutzung von weiteren Programmen, z. B. SQL Server Management Studio, nicht notwendig ist. Es werden mehrere Ansätze gezeigt und bewertet.

1 Pauschale Jobberechtigungen – der Holzhammer

In praktisch jedem Kundenprojekt ist ein Job im SQL Server-Agent des SQL-Servers eingerichtet, der zeitgesteuert die Daten aktualisiert. Da ist es doch naheliegend dem Anwender einfach die Berechtigung für die Job-Ausführung zu erteilen und diese Job-Ausführung in einer Prozedur zu hinterlegen, die aus der CustomApp heraus aufgerufen werden kann.

Leider lassen sich Jobs im SQL Server-Agents nur sehr grob berechtigen. Die Berechtigungen müssen auf der Systemdatenbank msdb gesetzt werden.

Für den SQL Server-Agent sind folgende Rollen vorgesehen:

- SQLAgentOperatorRole
- SQLAgentReaderRole
- SQLAgentUserRole

Der einfachste Weg ist es, dem Anwender für die Rolle SQLAgentOperatorRole zu berechtigen (implizit werden dabei auch die beiden anderen Rollen gesetzt). Der Anwender kann dann den Aktualisierungs-Job ausführen. Der Jobaufruf muss lediglich noch in die Prozedur die aus der CustomApp heraus ausgerufen wird hinterlegt werden und wir sind fertig:

```
ALTER PROCEDURE [dbo].[P_StartJob] AS

      EXEC msdb.dbo.sp_start_job @job_name = N'DeltaMaster_Full_Process'
```

Diese Variante hat jedoch einen entscheidenden Nachteil:

User in der Rolle SQLAgentOperatorRole haben die Berechtigung sämtliche Jobs des SQL Server-Agents zu sehen und auch auszuführen, d. h. es entsteht eine Sicherheitslücke, weil unter Umständen auch Jobs vorhanden sind, die der Anwender keinesfalls ausführen darf bzw. von denen er eventuell noch nicht einmal wissen darf.

In der Praxis wird dieser Punkt häufig in Kauf genommen, weil entweder keine weiteren Jobs vorhanden sind, diese nicht kritisch sind, oder es dem Endanwender schlicht und ergreifend

technisch nicht zugetraut wird die Jobs anzusehen und auszuführen. Dennoch sollte man sich der Auswirkung dieser Rolle stets bewusst sein.

2 Berechtigung über die Rollen Reader oder User

Wie bereits geschrieben existieren noch die beiden Rollen SQLAgentReaderRole und SQLAgentUserRole. Sind diese Rollen gesetzt gibt es folgende Auswirkungen:

- SQLAgentReaderRole

  Der Anwender sieht sämtliche Jobs im SQL Server-Agent. Er kann jedoch nur die Jobs starten, bei denen er als Job-Besitzer eingetragen ist. Er wird damit automatisch Mitglied der Rolle SQLAgentUserRole. Zusätzlich kann er die Jobs bei denen er als Job-Besitzer hinterlegt ist auch bearbeiten.

- SQLAgentUserRole
  Der Anwender sieht nur die Jobs im SQL Server-Agent, bei denen er als Job-Besitzer eingetragen ist und kann diese ausführen und bearbeiten.

Egal welche der beiden Berechtigungen man setzt, es entstehen diverse Folgeprobleme:

- Es kann nur ein Job-Besitzer pro Job hinterlegt werden. Soll mehr als ein Anwender die Möglichkeit zum Ausführen von Jobs erhalten, müssen parallele Jobs (einer pro Anwender) angelegt werden, bei denen jeweils ein Anwender als Besitzer eingetragen wird. AD-Gruppen können nicht als Job-Besitzer hinterlegt werden. Diese Parallelität müsste zudem in den CustomApp-Menüs bzw. der Aufrufprozedur berücksichtigt werden, damit stets der zum Anwender passende Job aufgerufen wird.
- Werden im Job auch Schritte ausgeführt, die nicht vom Typ „Transact-SQL-Skript (T-SQL)“ sind (z. B. verarbeiten der OLAP-Datenbank, Ausführen von SSIS-Paketen etc.), dann wird spätestens jetzt ein Proxy-User benötigt. Der Anwender muss dem Proxy als Prinzipal hinzugefügt werden. Es ist dann nicht mehr ausreichend das „Konto des SQL Server-Agent-Diensts“ zu nutzen. Dieses Konto kann nur genutzt werden, wenn der Job-Besitzer auch gleichzeitig die Serverrolle „sysadmin“ hat.
- Als Job-Besitzer können die Anwender den Job beliebig bearbeiten wodurch unter Umständen wieder ein Sicherheitsproblem entsteht.

3 Vorsicht – Job Startet (über benutzerdefinierte Warnmeldungen)!

Dieser eher exotische Ansatz macht sich die Tatsache zu Nutze, dass Jobs nicht nur zeitgesteuert, sondern auch über Warnungen gestartet werden können.

Zunächst wird eine benutzerdefinierte Warnmeldung im SQL-Server angelegt:

```
sp_addmessage
@msgnum = 50010
,@lang = 'us_english'
,@with_log = 'True'
,@severity = 1
,@msgtext = 'Start DeltaMaster data load.'
GO
```

```
sp_addmessage
@msgnum = 50010
,@lang = 'Deutsch'
,@severity = 1
,@msgtext = 'DeltaMaster Datenaufbereitung starten.'
```

Bei der ID (@msgnum) ist darauf zu achten, dass Werte größer 50000 gewählt werden, weil dieser Wertebereich für benutzerdefinierte Warnmeldungen reserviert ist. Zudem muss immer zuerst die englische Version der Warnmeldung angelegt werden, bevor weitere Sprachen folgen können. @severity = 1 ist das niedrigste Warnlevel (eher wie ein Hinweis zu interpretieren).

In die von der CustomApp aufgerufenen Prozedur wird nun ein Aufruf der definierten Warnmeldung eingebaut, damit sie bei jedem Aufruf der Prozedur ausgelöst wird:

```
ALTER PROCEDURE [dbo].[P_StartJob] AS

      RAISERROR(50010, 1, 1) return
```

Im Bereich „Warnungen“ des SQL Server-Agents muss nun eine neue Warnung angelegt werden:

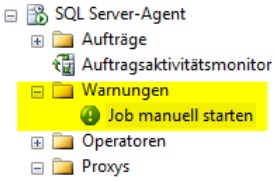

Abbildung 1: Warnung im SQL Server-Agent

Bei der Einrichtung der Warnung muss auf der Seite „Allgemein“ ein Name vergeben, die Warnung aktiviert, im Idealfall die Projektdatenbank ausgewählt (es können auch alle Datenbanken gewählt werden) und die ID der benutzerdefinierten Warnmeldung hinterlegt werden.

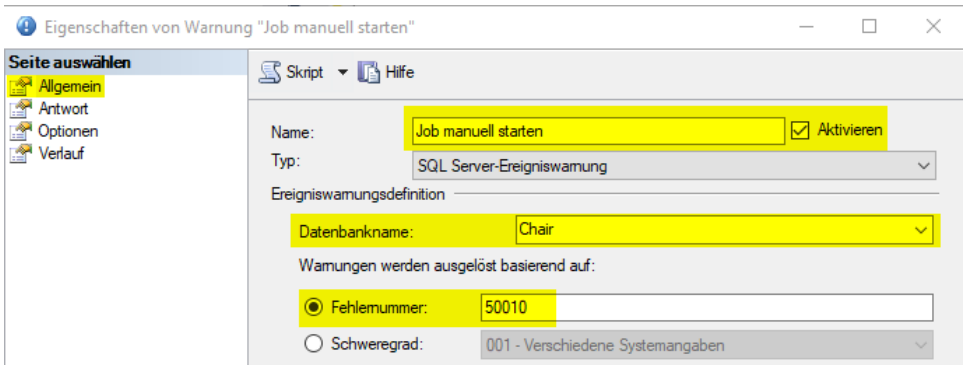

Abbildung 2: Eigenschaften einer Warnung - Seite „Allgemein“

Auf der Seite „Antwort“ muss der entsprechende Job eingestellt und der Haken bei „Auftrag ausführen“ gesetzt werden.

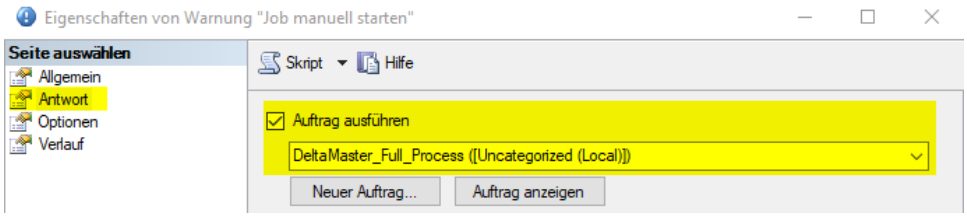

Abbildung 3: Eigenschaften einer Warnung - Seite „Antwort"

Diese trickreiche Variante wurde bereits vor längerer Zeit ausführlich in einem Blog vorgestellt (https://crew.bissantz.de/sql-server-agent-auftrage-ereignisgesteuert-ausfuhren).

Der Nachteil dieser Lösung ist, dass der Prozess nur extrem schwer nachvollziehbar ist, wenn keine Dokumentation vorliegt. Auch kann nicht ausgeschlossen werden, dass die Warnmeldung auch noch an anderen Stellen ausgelöst wird und der Job somit versehentlich gestartet wird. Die Variante wirkt eher unsauber und nicht so, als würde sie in dieser Form von Microsoft beabsichtigt sein.

Für das Anlegen von benutzerdefinierten Warnmeldungen sind zudem mindestens serveradmin-Rechte nötig, was sicherlich auch nicht in jedem Kundenprojekt gegeben ist.

4 Der Integration-Services-Katalog – die eierlegende Wollmilchsau?

Die bisher vorgestellten Ansätze hatten allesamt damit zu kämpfen, dass sie an irgendeiner Stelle unsauber wirkten. Abhilfe verspricht der Integration-Services-Katalog.

An dieser Stelle verabschieden wir uns von der Vorstellung, das für das Nachladen von Daten immer zwingend ein Job ausgeführt werden muss. Eigentlich ist es ausreichend, wenn der Anwender in die Lage versetzt wird SSIS-Pakete auszuführen, die die nötigen Schritte ausführen.

Voraussetzung ist zunächst, dass der „Integration-Services-Katalog" eingerichtet ist und die nötigen SSIS-Pakete im Katalog gespeichert wurden. Wie das funktioniert wurde bereits an anderer Stelle beschrieben (https://crew.bissantz.de/ssis-katalog-teil-1).

Sollen mehrere Pakete gemeinsam ausgeführt werden (parallel oder sequentiell), empfiehlt es sich ein weiteres dtsx-Paket zu definieren, das die Pakete dann in der gewünschten Reihenfolge ausführt.

Um nun neue Nutzer auf diese Pakete zu berechtigen, muss folgendes getan werden:

- Der Anwender (oder eine geeignete AD-Gruppe) muss auf der SSIS-Datenbank (SSISDB) bekannt gemacht werden, indem die Datenbank-Rolle public zugewiesen wird, d. h. der Anwender kann die Datenbank sehen, jedoch sonst nichts weiter mit ihr tun. Nur wenn das geschehen ist, kann der Anwender (oder die Gruppe) auch bei den im Folgenden beschriebenen Berechtigungen „gefunden" werden.

- Berechtigungen im Integration-Services-Katalog können auf Ordner- und Projektebene definiert werden (jeweils Rechtsklick auf das Objekt → Eigenschaften → Berechtigungen).

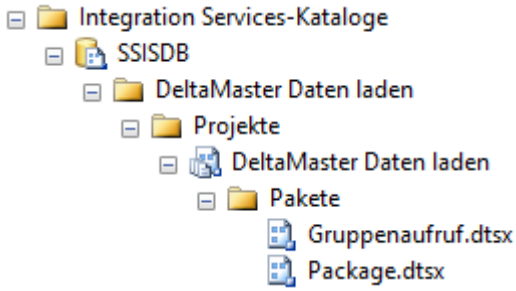

Abbildung 4: Struktur des Integration-Services-Katalogs

Um einem Anwender das Ausführen bestimmter Pakete zu ermöglichen sind mindestens folgende Berechtigungen zu setzen:

Ordner-Ebene: lesen

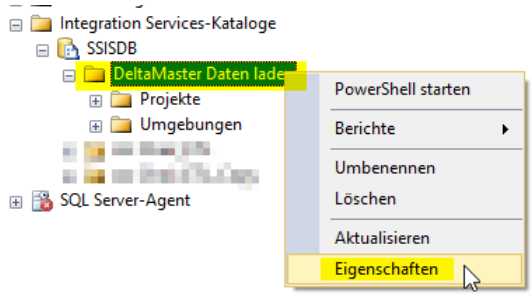

Abbildung 5: Zugriff auf Ordnereigenschaften

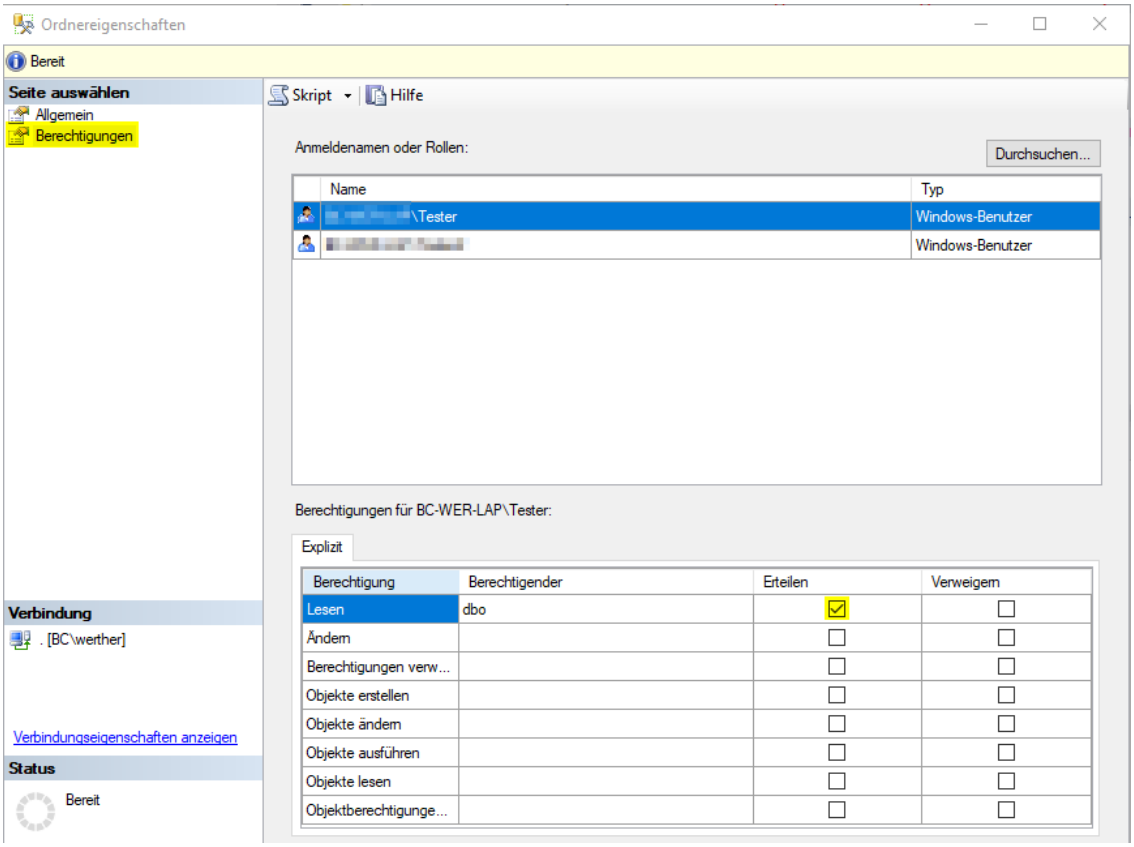

Abbildung 6: Lese-Berechtigung auf Ordner-Ebene gesetzt

Projekt-Ebene: lesen und ausführen

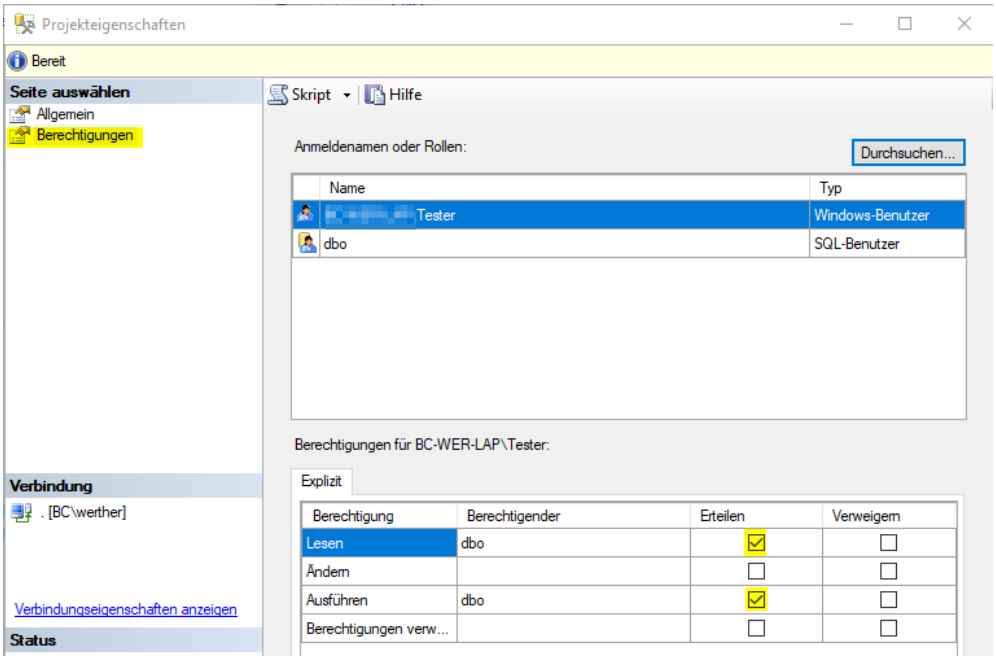

Abbildung 7: Lese- und Ausführen-Berechtigung auf Projekt-Ebene gesetzt

- Die Pakete können nun im Kontext des ausführenden Benutzers ausgeführt werden. Das bedeutet, dass der Nutzer gegebenenfalls auch Berechtigungen benötigt, die für die Ausführung der einzelnen SSIS-Pakete notwendig sind, z. B. Zugriff auf Dateien im Dateisystem, die importiert werden sollen, Verarbeiten den OLAP-Datenbank etc.

Abbildung 6 und Abbildung 7 geben einen Eindruck, welche weiteren Berechtigungen existieren, um beispielsweise einzelnen Anwendern oder Gruppen das Bearbeiten oder Berechtigen der Objekte zu ermöglichen.

Zusätzlich gibt es die Möglichkeit sogenannte Umgebungen zu definieren. Hier besteht die Möglichkeit die Parameter für Connection-Strings zu setzen. Damit lassen sich z. B. sehr einfach die Produktivumgebung und das Testsystem parametrisieren, ohne dass die Pakete doppelt gespeichert werden müssen. Auch lassen sich Passwörter für andere Datenquellen an dieser Stelle sicher hinterlegen (z. B. Zugriff auf ein Vorsystem mittels fixer Nutzer-Passwort-Kombination).

Als letztes muss das Paket natürlich noch ausgeführt werden. Der Standardweg ist der Rechtsklick auf das Paket und der Menüeintrag „Ausführen…".

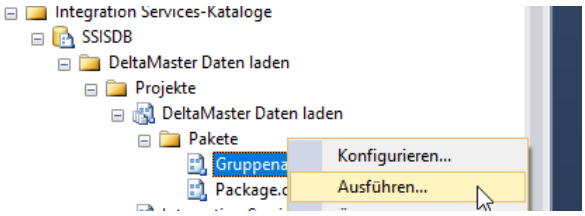

Abbildung 8: Aufrufen des Ausführen-Dialogs

Im sich öffnenden Menü besteht nochmals die Möglichkeit Parameter- und Verbindungsmanager zu definieren. Auf der Registerkarte „Erweitert" versteckt sich die häufig benötigte Eigenschaft „32-Bit-Laufzeit". Sind die Einstellungen abgeschlossen kann das Paket mit Klick auf OK gestartet werden.

Da wir das Paket künftig jedoch über eine SQL-Prozedur starten wollen, lassen wir uns den Paketaufruf skripten (Schaltfläche „Skript“).

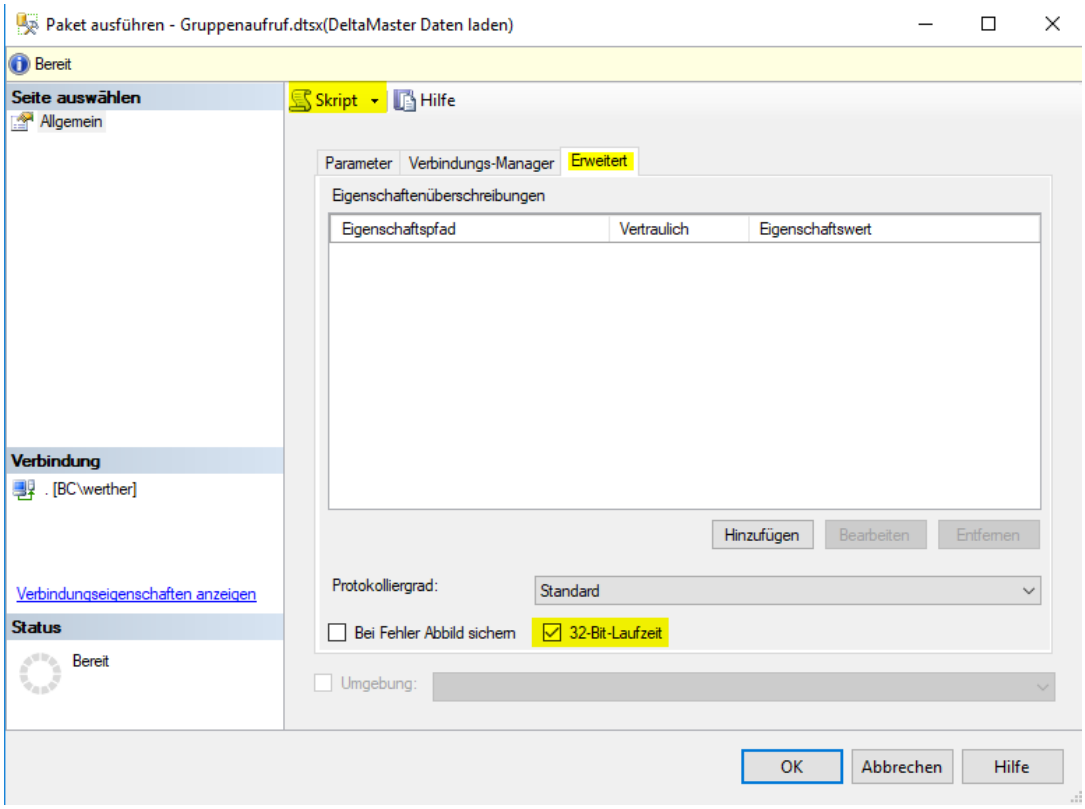

Abbildung 9: Ausführungs-Einstellungen und Skript-Erstellung

```
Declare @execution_id bigint

EXEC [SSISDB].[catalog].[create_execution] @package_name=N'Gruppenaufruf.dtsx', @execution_id=@execution_id OUTPUT, @folder_name=N'DeltaMaster Daten laden', @project_name=N'DeltaMaster Daten laden', @use32bitruntime=True, @reference_id=Null

Select @execution_id

DECLARE @var0 smallint = 1

EXEC [SSISDB].[catalog].[set_execution_parameter_value] @execution_id, @object_type=50, @parameter_name=N'LOGGING_LEVEL', @parameter_value=@var0

EXEC [SSISDB].[catalog].[start_execution] @execution_id

GO
```

Es empfiehlt sich diesen Code wie folgt zu erweitern:

```
IF
      (select count(*) from SSISDB.catalog.executions
      where object_id = <Objekt_ID>     -- object_id über "select * from SSISDB.catalog.object_versions"  herausfinden!
      and status in (1, 2, 5, 8)) <> 0 -- Status 1 = "erstellt", Status 2 = "wird ausgeführt", Status 5 = "ausstehend", Status 8 = "wird beendet"
      BEGIN
```

```
            raiserror('Die Aktualisierung kann nicht gestartet werden,
weil aktuell bereits eine Aktualisierung läuft!',16,1) return
      END
Declare @execution_id bigint
EXEC [SSISDB].[catalog].[create_execution] @package_name=N'Gruppenau-
fruf.dtsx', @execution_id=@execution_id OUTPUT, @folder_name=N'DeltaMaster
Daten laden', @project_name=N'DeltaMaster Daten laden',
@use32bitruntime=True, @reference_id=Null
--Select @execution_id
DECLARE @var0 smallint = 1
EXEC [SSISDB].[catalog].[set_execution_parameter_value] @execution_id,
@object_type=50, @parameter_name=N'LOGGING_LEVEL', @parameter_value=@var0
EXEC [SSISDB].[catalog].[start_execution] @execution_id
```

Dabei wird vor jeder Ausführung geprüft, ob das Paket bereits ausgeführt wird. Somit können Parallelausführungen (z. B. durch ungeduldige Anwender ausgelöst) direkt abgefangen werden.

5 Fazit

Zusammengefasst spricht fast alles dafür, den Ansatz über den Integration-Services-Katalog zu wählen, insbesondere wenn dieser ohnehin schon eingerichtet ist. Bei allen anderen Varianten gilt es die beschriebenen Nachteile sorgfältig abzuwägen.

Der Integration-Services-Katalog bietet zudem noch viele weitere Vorteile, wie etwa eine Versionierung, Parametrisierbarkeit etc., die für die Problemstellung in diesem Blog zwar nicht relevant sind, jedoch ebenfalls große Vorteile im Projektvorgehen bringen können.