# Badcock Report Subscription Service

The Report Subscription Service is a system that supports ***data driven*** subscriptions. This allows a single query to create a batch of reports, each with different options, and either save them as files or email them.

This service has features to automatically re-queue failed reports or abandon the subscription if a pre-selected number of reports fail. In practice, it’s easy to queue up hundreds of reports which can cause timeouts.

The Reporting Services web site has a Subscription service that will create a single report to a single destination on a fixed schedule. If you don’t the flexibility of a data driven subscription, it might be a better choice.

# Overview

The best way to understand how the Report Subscription Service works is to start in the middle of the process. As mentioned above, a single query is used to create reports with different options. The options for each report are packaged as an XML fragment. The following is a sample that creates a SAP report as a PDF and emails it.

<xml>

<cycle>0</cycle>

<store>120</store>

<store>127</store>

<currperiod>201109</currperiod>

<cycletot>Yes</cycletot>

<filename>

SAP\_DealerStores\_AllCycles\_Totals\_AsOf\_20110913.pdf

</filename>

<dealercorp>D</dealercorp>

<deliverymode>email</deliverymode>

<emailto>robert.hatton@badcock.com</emailto>

<emailcc>kathy.richards@badcock.com</emailcc>

<emailsubject>Report Subscription: SAP/DealerStores/AllCycles/Totals - created: 09-13-2011</emailsubject>

<emailbody>Please do not reply to this email. It is from the Badcock automated Report Subscription system.If you have questions or concerns about the report or subscription, please contact the Help desk and reference: "SAP\_DealerStores\_AllCycles\_Totals\_PDF\_eMail"</emailbody>

<exportformat>pdf</exportformat>

</xml>

There are a few things to note about the above example:

The value of the ‘cycle’ element is 0. The SAP report interprets this as ‘all cycles’. The point here is that it is important to understand what values the report expects.

The second point to note is that the ‘store’ element occurs twice. This is the way the Reporting Services expects multi-value parameters to be passed. This applies to any multi-value parameters, not just the store. A SAP report that is for all stores would expect over 300 <store> elements.

The XML fragments defining report options are housed in the ReportQueue table. They are created by a SQL Agent job called the Splitter (which splits a single Report Request into multiple ReportQueue records), and are processed into individual reports by a job called the Runner.

An illustration of the process is as follows:

Your request process

Insert Subscription Request

Subscription Request

Subscription

Report

Report

filesystem

email

Runner

ReportQueue

Splitter

The Subscription is a definition of what the subscription is supposed to do. This is something you will need to create as part of the process of setting up a new subscription. A detailed discussion of how to do this is provided later. Subscriptions are housed in the Subscriptions table in the ReportSubscriptions database. This table has columns for the subscriptionName, a URL for the report, the name of the report, a query that will create the XML fragments, a connection string for the query the email address of the person to be notified if the job fails, and a place to specify how many reports can fail before the entire subscription batch is abandoned.

The box labeled ‘Your request process’ represents the second thing that you will need to create to set up a subscription. A more detail discussion will follow, but this will most commonly be a SQL statement in a SQL Agent job that will execute a stored procedure that will insert a record into the SubscriptionRequest table.

The box labeled ‘Insert Subscription Request’ represents a stored procedure that will accept the name of a subscription with an optional BatchID. If you don’t provide a BatchID, it will create one. It will then insert a record into SubscriptionRequests.

The Subscription Request is a record in the SubscriptionsRequet table that serves as a signal for the Splitter that it should run the query from the appropriate Subscription and create ReportQueue a record for each desired report.

The Splitter is a SSIS package that is started by a SQL Agent job. The SQL Agent job (named Report Subscriptions - Splitter) is scheduled to run every 5 minutes. Once it’s done splitting the request into report parameters, it will update the ProcessingStatusID with a 3. The complete list of Processing Status values is as follows:

|  |  |  |
| --- | --- | --- |
| ProcessingStatusID | Description | IsComplete |
| 0 | Requested, Pending Split Into Queue | 0 |
| 1 | Split Successful, Reports Queued | 0 |
| 2 | Split Failed | 0 |
| 3 | Report Generation in Progress | 0 |
| 4 | Report Generation - Succeeded | 1 |
| 5 | Report Generation - Failed | 1 |

The Report Queue is a table with one record for each report to be generated. It is populated by the Splitter, and used by the Runner (which also updates it’s ResultStatusID). It contains columns to show when it was created, which batch it belongs to, the ReportParameters XML fragment, a ResultStatusID, and ResultMessage. The list of result status values is as follows:

|  |  |  |
| --- | --- | --- |
| ResultStatusID | Description | IsComplete |
| 0 | Pending | 0 |
| 1 | In Progress | 0 |
| 2 | Succeeded | 1 |
| 3 | Failed | 1 |

The Runner is the part of the system that actually creates reports. It’s an SSIS package that runs in a SQL Agent (named Report Subscriptions - Runner) job that is scheduled to start every 5 minutes. If it finds a record in SubscriptionRequests that does not already have an owner, it will fill in the Owner field with the host name & PID (this is to support multiple Runner jobs). It will then loop through all of the records in the ReportQueue table that are associated with the current BatchID, and create reports. The runner uses several elements in the XML fragment that aren’t options for the report. These elements and their valid values are as follows:

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Notes |
| deliverymode | email or filesystem | required |
| exportformat | pdf or csv or mhtml | required |
| filename | Full pathname if deliverymode is filesystem. Basic file name if deliverymode is email. | Required (for either filesystem and email) |
| emailto | Valid email address | Required if deliverymode is email |
| emailcc | Valid email address | optional |
| emailbcc | Valid email address | optional |
| emailsubject | Text (80 characters or less) | optional |
| Emailbody | Text(varchar(max)) | optional |

Note that emailto, emailcc, and emailbcc may have multiple elements, the deliverymode, exportformat and filename can only occur once.

# Creating a new Subscription

A new Subscription can be created by inserting a record into the Subscription table. Note that this will not cause any reports to be created, you’ll need a SubscriptionRequest for that (more on that later).

The following is a sample insert statement to create a subscription request:

INSERT INTO [ReportSubscriptions].[dbo].[Subscriptions]

([SubscriptionName]

,[ReportingServicesURL]

,[ReportName]

,[Query]

,[ConnectionStringForQuery]

,[MaxAllowedErrors]

,[NotifyEmailAddress]

,[MaxReprocessFailedRequest])

VALUES

('SAP\_DealerStores\_AllCycles\_Totals\_PDF\_eMail'

,'http://tsql02/ReportServer\_V2008'

,'/sap reports/SAPReportMerge'

,'ReportSubscriptions..SAP\_DealerStores\_AllCycles\_Totals\_PDF\_eMail'

,'Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=STOREnetDW;Data Source=PSQL03'

,0

,'robert.hatton@badcock.com'

,1)

GO

The SubscriptionName should be descriptive. It **must** be unique.

The ReportingServicesURL points to the server that you want the subscription to run on. Note that this must be manually changed between test and production.

The ReportName is the fully qualified path to the report on the ReportingServices Server.

The Query can either be the name of a stored procedure or the text of a SQL query. A more detailed discussion of how to create the query is provided below.

ConnectionStringForQuery should point to the database that the associated query needs to run against. Not that this is often different that he connection that the report uses.

MaxAllowedErrors specifies how many reports can fail before the entire batch is abandoned. If this number of errors is exceeded, the Runner will update the ProcessingStatusID of the SubscriptionRequest and terminate with a failed result code.

## Creating the query

The query that creates the XML fragments used to create reports can either be saved directly in the Query column of the Subscription or packaged in a stored procedure. In actual practice, saving the query text directly in the Query field is annoying as it’s difficult to save carriage returns in a text field. Should you create a stored procedure, save it in the ReportSubscriptions database and put the fully qualified name in the Query field. It’s advisable to use the SubscriptionName as the stored procedure name (or at least come as close as possible).

The Splitter will actually execute the query. It expects the query to insert XML fragments into a temporary table named #report\_queue The Splitter will have created the table beforehand.

The following is a sample of a stored procedure that houses a query for report subscriptions:

USE [ReportSubscriptions]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SAP\_AllStores\_AllCycles\_Totals\_PDF\_eMail] Script Date: 09/14/2011 11:30:54 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE proc [dbo].[SAP\_AllStores\_AllCycles\_Totals\_PDF\_eMail] as

--Set the cycle number

declare @cycleNum as int = 0

--Create the 'store' elements for the stores in the current cycle before we need them....

declare @stores as varchar(max) = ''

declare @storeID as varchar(50) = ''

declare stores cursor for

select StoreID from

(select distinct StoreID

from SDW.STOREnetDW.dbo.CycleEndSAP SAP

inner join MDMS.dbo.lm\_store\_master store

on store.store\_number = SAP.StoreID

inner join MDMS.dbo.lm\_cycle cycle

on store.cycle\_id = cycle.id

where CycleYearMonth = convert(char(6), getdate(), 112)

and StoreID between 1 and 899) as tmp

order by StoreID

open stores

fetch next from stores into @StoreID

while @@FETCH\_STATUS = 0

begin

set @stores = @stores + '<store>' + rtrim(@StoreID) + '</store>'

fetch next from stores into @StoreID

end

close stores

deallocate stores

--Assemble the email subject

declare @subject as varchar(100)

set @subject = 'Report Subscription: SAP/'

set @subject = @subject + 'AllStores/AllCycles/Totals - created: ' + convert(char(10), getdate(), 110)

--Assemble the text of the message body

declare @body as varchar(max)

set @body = 'Please do not reply to this email. It is from the Badcock automated Report Subscription system.'

set @body = @body + 'If you have questions or concerns about the report or subscription, please contact the Help desk '

set @body = @body + 'and reference: "SAP\_AllStores\_AllCycles\_Totals\_PDF\_eMail"'

--Assemble the file name

declare @fileName as varchar(80)

set @fileName = 'SAP\_AllStores\_AllCycles\_Totals\_AsOf\_'

set @fileName = @fileName + convert(char(8), getdate(), 112) + '.pdf'

--Now create the xml fragment into the temp table

insert into #report\_queue

select top 1

'<xml>'

+ '<cycle>'

+ cast(@cycleNum as varchar)

+'</cycle>'

+ @stores

+ '<currperiod>'

+ rtrim(convert(char(6), dateadd(m, 1, dateadd(d, -1, getdate())), 112))

+ '</currperiod>'

+ '<cycletot>'

+'Yes'

+ '</cycletot>'

+ '<filename>'

+ @fileName

+ '</filename>'

+ '<dealercorp>'

+ 'A'

+ '</dealercorp>'

+ '<deliverymode>'

+ 'email'

+ '</deliverymode>'

+ '<emailto>'

+ 'robert.hatton@badcock.com'

+ '</emailto>'

+ '<emailcc>'

+ 'kathy.richards@badcock.com'

+ '</emailcc>'

+ '<emailsubject>'

+ @Subject

+ '</emailsubject>'

+ '<emailbody>'

+ @body

+ '</emailbody>'

+ '<exportformat>'

+'pdf'

+ '</exportformat>'

+ '</xml>'

GO

This query starts out by creating and populating some variables to house XML chunks that would be awkward to build inside of the insert statement. The @Stores variable will actually have 300+ <store> elements (all stores are included in this report).

As mentioned earlier, the insert statement will put records in the #report\_queue table. There is only one column in the table, so a field list is not needed. Note that this implies that your query can only return one column.

If you do want to create a subscription with an embedded query instead of creating a stored procedure, you will need to embed the query in the insert statement that creates the subscription. If, for example, you wanted to create a batch of SAP reports for a past cycle, you could create a subscription that has the desired cycle and date portion of the filename hard coded. This could be done by changing the underlined portions of the following example:

INSERT INTO [ReportSubscriptions].[dbo].[Subscriptions]

([SubscriptionName]

,[ReportingServicesURL]

,[ReportName]

,[Query]

,[ConnectionStringForQuery]

,[MaxAllowedErrors]

,[NotifyEmailAddress]

,[MaxReprocessFailedRequest])

VALUES

('SAP\_DealerStores\_AllCycles\_Totals\_PDF\_eMail\_sample'

,'http://tsql02/ReportServer\_V2008'

,'/sap reports/SAPReportMerge'

, 'insert into #report\_queue select distinct ''<xml>'' + ''<cycle>'' + **rtrim(ST\_CYCLE\_NUM)** +''</cycle>'' + ''<store>'' + rtrim(ST\_CONV\_NBRS) + ''</store>'' + ''<currperiod>'' + rtrim(case when ST\_CYCLE\_NUM < datepart(d, dateadd(d, -1, getdate())) then convert(char(6), dateadd(m, 1, getdate()), 112) else convert(char(6), getdate(), 112) end) + ''</currperiod>'' + ''<cycletot>No</cycletot>'' + ''<filename>'' + ''\\corp\shares\department\_shares\Retail\_Operations\SAP Reports\Cycle ''+ rtrim(ST\_CYCLE\_NUM)+ ''\'' + rtrim(ST\_CYCLE\_NUM) + ''\_'' + replace(rtrim(ST\_CONV\_NAME), '','', ''\_'') + ''\_'' + rtrim(ST\_CONV\_NBRS) + ''\_'' + **convert(char(8), dateadd(d, -1, getdate()), 112)** + ''.PDF</filename>'' + ''<deliverymode>0</deliverymode>'' + ''<exportformat>PDF</exportformat>'' + ''<dealercorp>A</dealercorp>'' + ''</xml>'' from SDW.STOREnetDW.dbo.stconv s inner join SDW.STOREnetDW.dbo.CycleEndSAP SAP on s.ST\_CONV\_NBRS = SAP.StoreID where CycleYearMonth = convert(char(6), getdate(), 112) and ST\_CONV\_NBRS between 1 and 899 and ST\_CYCLE\_NUM = datepart(d,dateadd(d,-1,getdate()))'

,'Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=STOREnetDW;Data Source=PSQL03'

,0

,'robert.hatton@badcock.com'

,1)

GO

Remember that you can’t embed carriage returns in the query, and all embedded single quotes must be escaped with a second single quote.

# Starting a Subscription

Starting a Subscription is simply a matter of executing the InsertSubscriptionRequest stored procedure, and providing the name of the Subscription as an argument. Here is an example:

exec ReportSubscriptions.dbo.InsertSubscriptionRequest 'SAP\_DealerStores\_AllCycles\_Totals\_PDF\_eMail'

The most common place to call InsertSubscriptionRequest is in a SQL Agent job. It could also be called from an SSIS package or even done manually in Management Studio.

Bear in mind that once the SubscriptionRequest has been inserted, the calling process will continue immediately and it may be more than 10 minutes before the report starts generating (depending on the schedule that the Splitter and Runner use). If you want to have the calling process wait for the report generation to finish, InsertSubscriptionRequest can be called with both a SubscriptionName and a BatchID. The BatchID can then be used to monitor the status of the SubscriptionRequest as illustrated in the following example:

declare @MyBatchID uniqueidentifier

declare @MyStatus tinyint

declare @OkToExit bit = 0

declare @StartTime datetime = getdate()

declare @WaitForMinutes int = 60

declare @FailCount int = 0

--queue the request

set @MyBatchID = NEWID()

**exec SQL02.ReportSubscriptions.dbo.InsertSubscriptionRequest 'SAP\_ByDay\_ByStore\_DealerCorp\_Details\_PDF\_portal', @MyBatchID**

--start polling every minute until the request is complete or we hit the

--expiration

while @OkToExit = 0 and DATEDIFF(mi, @StartTime, GETDATE()) < @WaitForMinutes

begin

--retrieve current status

exec SQL02.reportsubscriptions.dbo.CheckSubscriptionRequestStatus

@MyBatchID, @MyStatus output, @FailCount output

--best case, finished with no issues

if @MyStatus = 4 or @MyStatus = 5

set @OkToExit = 1

--wait 1 minute and check again

waitfor delay '00:01:00'

end

--notify user that we're ending without results

if @OkToExit = 0

RAISERROR('Report Subscriptions - Wait for report completion exceeded specified wait interval.', 16, 1)

--notify user if we're in a fail status and the fail count was over our threshold - its better to proceed if we're mostly successful than fail due to 1 error

if @MyStatus = 5 and @FailCount > 5

RAISERROR('Report Subscriptions - One or more reports failed for this request failed. Details can be found in the ReportQueue table.', 16, 1)

# Troubleshooting

Failures due to anything other than timeouts on production reports are ***very*** rare. For most failures where fewer than MaxAllowedErrors are encountered, the auto-retry will be successful. Be aware that the auto-retry won’t happen until the next time the Runner is triggered, so it may be a few minutes before this happens.

In the event the MaxAllowedErrors is exceeded, an email is sent to the recipient listed on the Subscription with instructions on how to manually re-start the batch. Simply cut and paste from the email into a query window and run the command to restart the batch.

If the problem wasn’t a timeout and the automatic and manual retries didn’t work, the first place to look is the ReportQueue table. Check the ResultMessage column to see why individual reports failed.

If you are creating a new subscription and it is failing, the best approach is to check the ReportQueue to see if the ResultMessage gives a clue to the problem. If that doesn’t help, you may need to open one of the XML fragments and manually select all of the options in the report that are specified in the XML.