GETTING MORE THAN 8000 ROWS IN Cisco Unified Intelligence Center

Version: 1.0

BENEK OZER

Cisco AS CoE – Network Consulting Engineer

PREFACE

This study has been conducted to overcome the 8000 row limit on CUIC for report result sets that exceed the solution hard limit. Readers of this study may deploy same or similar approaches to go beyond CUIC's hard limit of 8000 rows.

The approach used in this document is developed as a sample. It has not been tested or deployed for extensibility or concurrent usage by Cisco or the document publisher.

INTRODUCTION

CUIC is the official reporting interface for Cisco Contact Center solution. It provides a customizable interface for end users to access and analyze contact center data via web portal and even other data sources (MS SQL and Informix) if CUIC is deployed with a premium license.

CUIC is not a data store, it only stores report configurations, report entities and connection information for report data sources. It does contain access points to contact center databases and representation layers for data fetched from these data sources. It allows a query passage into these databases to bring requested chunk of data and present this data in user readable and recognizable form. Because it is a web portal - a kind of web application solution - its pure goal is to bring an interaction between raw data and reporting user both in tabular and visual forms.

Once a report is run, it brings the requested data to be viewed and observed by the reporting user. The data for this report is not recorded on CUIC itself, CUIC will only record changes if the report's presentation related configurations are modified and user wants to keep them.

If the reporting user wants to keep the tabular data that is displayed on CUIC web interface, he/she can export it to Microsoft Excel in csv format. This extraction presents the tabular form user is seeing on the screen and it is used for referential purposes. This option provides user an offline copy which can exist for personal archival purposes or be used in other different ways.

WHY 8000 ROWS LIMITATION?

CUIC is the visible and recognizable face of a Contact Center for the business stakeholders. It will be accessed and used by a number of end users concurrently. These users require an acceptable response time and minimum latency from CUIC for optimum experience.

A CUIC cluster is designed based on number of concurrent users and other sizing factors. However reports rich in data and content will impose utilization issues both on network and system resources as the data is not local; it is all pulled from the external data source.

For this reason, CUIC uses (concurrent) *report row* processing as an identifier for overall CUIC resource consumption and management. Using this identifier, it deploys a throttling mechanism to prevent servers from freezing or memory related issues as a result of overload. This method does not ensure a complete quality of service for end users; however it provides memory usage and control for the CUIC system itself.

As an extension of this preservation, one historical report is not allowed to have more than 8000 rows. This rule extends to both run time reports and scheduled reports.

RAW DATA VERSUS PROCESSED DATA

Note that data is only useful when analyzed and understood; not seen in raw format. For example, a contact center may receive hundreds of thousands of calls per day but just extracting the details of that quantity into a report or tabular display without any analytical logic will not have any real meaning. Data analytics come to play when building logic into these raw rows of data. For a user, a report or a graphic will not really benefit from those hundreds of thousands of rows generated by hundreds of thousands of calls. Nevertheless, they will make excellent input for useful summary data such as 'How many calls received in a day?', 'How many calls abandoned?' or 'Which hour received the highest number of calls?' Meaningful summaries and calculations from the original raw data will make better sense to report on. They will have meaning and better results and conclusion.

A database is storage of structured data. The data it contains can be (and should be) archived and kept for as long as needed. From Contact Center's perspective, if data from 5 years ago regarding an agent's performance is required; CUIC will not be the resource for this, nor the UCCE databases. Data that old must have already been purged from UCCE databases. However it should be available in a data mart somewhere within the enterprise. In a well-managed environment, such data will never be lost but can be accessed as and when needed.

If in theory data is never lost in an enterprise and can be accessed through databases, data marts and data warehouses, what is the real need to extract raw data of thousands of rows and keep in an excel sheet? Why would anyone keep and maintain such redundant and enormous sized data?

One business user might say they keep the data in Excel as is and untouched. However, data is not 'useful' data if not understood and analyzed.

Another business user might say 'I use my own methods, macros and calculations to analyze data' Analysis of data is a process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. (Reference: Wikipedia) This is perhaps a better and more realistic requirement compared to extracting large sum of raw data. However, as advanced as MS Excel or similar software is, same calculations, summarizations, filtering and business logic can be applied on SQL query logic and from that into CUIC's report templates. If data analysis is completed on database or CUIC report template layer, it also can be used by wider CUIC audience and increase efficiency within the enterprise.

WHY A USER WOULD NEED 8000+ ROWS ON CUIC?

On the other side of the coin, a CUIC reporting user may be trying to get a summarized data of Skill Group statistics for a day. However, if the UCCE system is set to generate summaries at 15 minute intervals and this user requires tracking statistics of 84 Skill Groups or more, he/she will definitely hit the CUIC 8000 row limitation even for single day:

1 Skill Group generates 4 rows per hour, 96 rows per day

84 Skill Groups will have -> 84 x 96 = 8064 rows/day

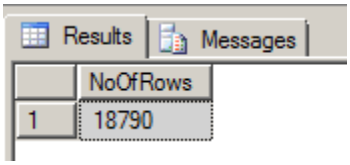
This user cannot get 84 Skill Groups' summary data per day without decreasing the number of Skill Groups selected for the filter screen.

How to get 8000+ rows from one report template?

When a report to be developed is suspected to bring more than 8000 rows per request, the report template and its report definition needs to be considered to address this potential requirement.

In the example, UCCE Skill Group table to be used has a total of 18790 rows. Note that the data source is from a lab deployment.

```
SELECT COUNT(1) AS NoOfRows FROM Skill_Group_Interval sgi (nolock)
```



If the report user wants to display all rows from table Skill_Group_Interval, he/she will face the 8000 row limitation and cannot achieve the goal.

If this person is a user with DB and SQL expertise, he/she can login to SQL Management Console to easily query the Skill_Group_Interval table and list all and every Skill Group row of 18790. The query may take a short while and consume system resources but it will still return back the result set.

In SQL Language, using 'TOP' statement in SELECT query will limit the number of rows returned. The below statement will bring the first 8000 rows from the Skill Group Interval table.

```
SELECT TOP 8000 * FROM Skill_Group_Interval sgi (nolock)
```

While the first 8000 records are viewed like this, the remaining rows cannot be easily fetched. A logic needs to be formulated to bring the data in chunks of 8000.

Approach In TSQL

To display all rows when the quantity is over 8000 is not possible at one go on CUIC interface. However, total number of rows can be grouped into pages and user will call 2nd, 3rd, 4th (and so on) page when the total row number overlaps 8000 rows. Based on this page assignment to rows approach:

1st Page:	Row 1	-	Row 8000
2nd Page:	Row 8001	-	Row 16000
3rd Page:	Row 16001	-	Row 24000
4th Page:	Row 24001	-	Row 32000

…

So on and so forth.

CONSTRUCTING QUERY IN TSQL

Not all but a couple of columns from Skill_Group_Interval table will be queried for this exercise. These columns are:

- SkillTargetID
- DateTime
- RecoveryKey
- LoggedOnTime
- TalkTime
- NotReadyTime
- BusyOtherTime
- WorkReadyTime
- WorkNotReadyTime

Normally, the TSQL will be as simple as this:

```
SELECT
      SkillTargetID,
      DateTime,
      RecoveryKey,
      LoggedOnTime,
      TalkTime,
      NotReadyTime,
      BusyOtherTime,
      WorkReadyTime,
      WorkNotReadyTime
FROM
      Skill_Group_Interval SGI (nolock)
WHERE
      DateTime BETWEEN @start AND @end
ORDER BY
      RecoveryKey
```

…where start date and end date can be set as parameters. However, the TSQL should be constructed with logic to allow querying all records; only 8000 rows at a time.

The complete TSQL is as follows:

```
IF OBJECT_ID('tempdb..#SGI') IS NOT NULL
BEGIN
DROP TABLE #SGI
END

DECLARE @i INT
DECLARE @pageMax INT
DECLARE @pageMin INT
DECLARE @start DATETIME
DECLARE @end DATETIME

SET @i = 1
SET @start = '2013-01-01'
SET @end = '2015-01-01'
```

```
IF @i IS NULL OR @i < 1
	SET @i = 1

SET @pageMax = @i*8000
SET @pageMin = (@i-1)*8000 + 1

CREATE TABLE #SGI
(
	SeqID INT IDENTITY(1,1) NOT NULL,
	SkillTargetID INT,
	DateTime DATETIME,
	RecKey FLOAT,
	LoggedOnTime INT,
	TalkTime INT,
	NotReadyTime INT,
	BusyOtherTime INT,
	WorkReadyTime INT,
	WorkNotReadyTime INT
)

INSERT INTO #SGI
(
	SkillTargetID,
	DateTime,
	RecKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
)
SELECT
	SkillTargetID,
	DateTime,
	RecoveryKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
FROM
	Skill_Group_Interval SGI (nolock)
WHERE
	DateTime BETWEEN @start AND @end
ORDER BY
	RecoveryKey

SELECT
	SeqID,
	SkillTargetID,
	DateTime,
	RecKey,
	LoggedOnTime,
	TalkTime,
```

```
      NotReadyTime,
      BusyOtherTime,
      WorkReadyTime,
      WorkNotReadyTime
FROM
      #SGI (nolock)
WHERE
      SeqID BETWEEN @pageMin AND @pageMax

DROP TABLE #SGI
```

In this paging method, the query is designed to run multiple times (as many as required). Each time it runs, it splits returned rows into groups/pages where each group/page only returns 8000 rows. The query will supply the page number, so if it is 1, it will fetch records 1 to 8000; if it is 2, records from 8001 to 16000, etc.

The TSQL logic is explained in detail below

Check tempdb if a table named #SGI exists. If so, drop it:

```
IF OBJECT_ID('tempdb..#SGI') IS NOT NULL
BEGIN
DROP TABLE #SGI
END
```

@i will represent the number of the page which should be returned in the result set.
@pageMax and @pageMin will be used to identify the starting and ending row ID for the page/group requested.
@start and @end parameters are used to input start date and end date for the query.

```
DECLARE @i INT
DECLARE @pageMax INT
DECLARE @pageMin INT
DECLARE @start DATETIME
DECLARE @end DATETIME
```

@i, @start and @end must be supplied as inputs. @pageMax and @pageMin do not need to be fed to the query since query has the logic to calculate them in the next portion.

```
SET @i = 1
SET @start = '2013-01-01'
SET @end = '2015-01-01'
```

Check to see if @i parameter is an acceptable integer. If not, set it to 1:

```
IF @i IS NULL OR @i < 1
      SET @i = 1
```

The calculation logic for starting row number and ending row number:

```
SET @pageMax = @i*8000
SET @pageMin = (@i-1)*8000 + 1
```

A temp table is required to store returned rows. It has all the columns required and also an Identity column to mark the row number.

```
CREATE TABLE #SGI
(
	SeqID INT IDENTITY(1,1) NOT NULL,
	SkillTargetID INT,
	DateTime DATETIME,
	RecKey FLOAT,
	LoggedOnTime INT,
	TalkTime INT,
	NotReadyTime INT,
	BusyOtherTime INT,
	WorkReadyTime INT,
	WorkNotReadyTime INT
)
```

Temp table is populated with required data. Note that 'ORDER BY' statement is critical to have a solid ordering mechanism every time this query is run for different pages.

```
INSERT INTO #SGI
(
	SkillTargetID,
	DateTime,
	RecKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
)
SELECT
	SkillTargetID,
	DateTime,
	RecoveryKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
FROM
	Skill_Group_Interval SGI (nolock)
WHERE
	DateTime BETWEEN @start AND @end
ORDER BY
	RecoveryKey
```

'SELECT' from the final table. This SELECT will only bring 8000 rows (unless it is the last page)

```
SELECT
	SeqID,
	SkillTargetID,
	DateTime,
	RecKey,
```

```
        LoggedOnTime,
        TalkTime,
        NotReadyTime,
        BusyOtherTime,
        WorkReadyTime,
        WorkNotReadyTime
FROM
        #SGI (nolock)
WHERE
        SeqID BETWEEN @pageMin AND @pageMax
```

Drop the temporary table:

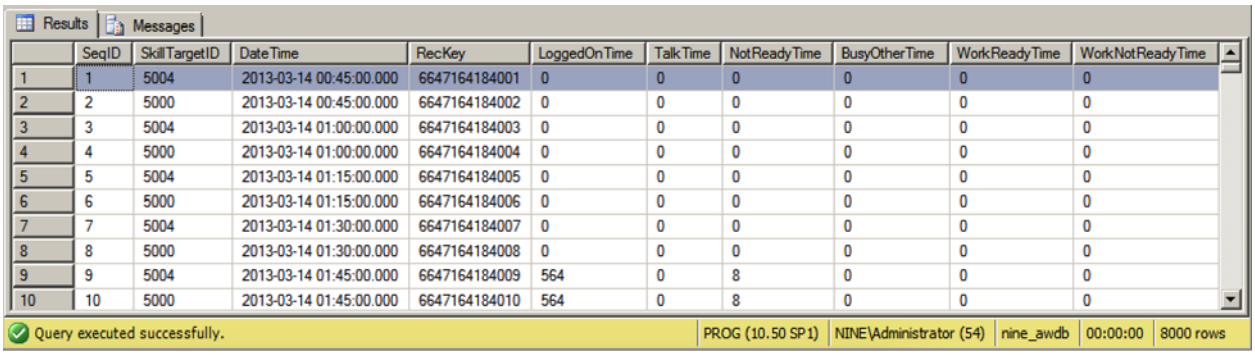
```
DROP TABLE #SGI
```

RUNNING TSQL

When the TSQL is run with the following parameters:

```
SET @i = 1
SET @start = '2013-01-01'
SET @end = '2015-01-01'
```

First returned rows:

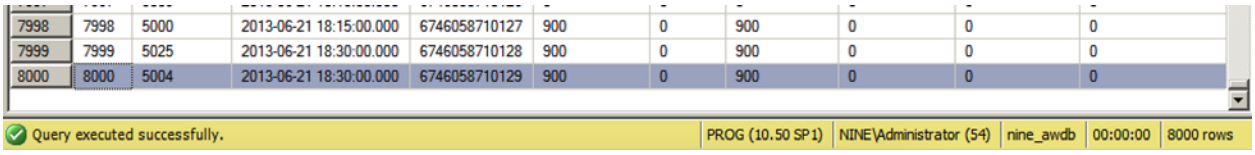


Results | Messages

	SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime	WorkReadyTime	WorkNotReadyTime
1	1	5004	2013-03-14 00:45:00.000	6647164184001	0	0	0	0	0	0
2	2	5000	2013-03-14 00:45:00.000	6647164184002	0	0	0	0	0	0
3	3	5004	2013-03-14 01:00:00.000	6647164184003	0	0	0	0	0	0
4	4	5000	2013-03-14 01:00:00.000	6647164184004	0	0	0	0	0	0
5	5	5004	2013-03-14 01:15:00.000	6647164184005	0	0	0	0	0	0
6	6	5000	2013-03-14 01:15:00.000	6647164184006	0	0	0	0	0	0
7	7	5004	2013-03-14 01:30:00.000	6647164184007	0	0	0	0	0	0
8	8	5000	2013-03-14 01:30:00.000	6647164184008	0	0	0	0	0	0
9	9	5004	2013-03-14 01:45:00.000	6647164184009	564	0	8	0	0	0
10	10	5000	2013-03-14 01:45:00.000	6647164184010	564	0	8	0	0	0

Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 8000 rows

Last returned rows:



7998	7998	5000	2013-06-21 18:15:00.000	6746058710127	900	0	900	0	0	0
7999	7999	5025	2013-06-21 18:30:00.000	6746058710128	900	0	900	0	0	0
8000	8000	5004	2013-06-21 18:30:00.000	6746058710129	900	0	900	0	0	0

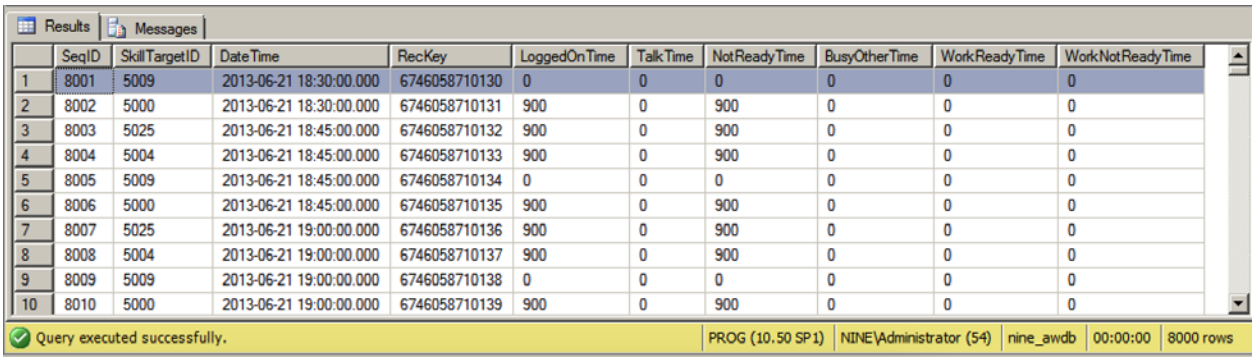
Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 8000 rows

---

When the TSQL is run with the following parameters:

```
SET @i = 2
SET @start = '2013-01-01'
SET @end = '2015-01-01'
```

First returned rows:

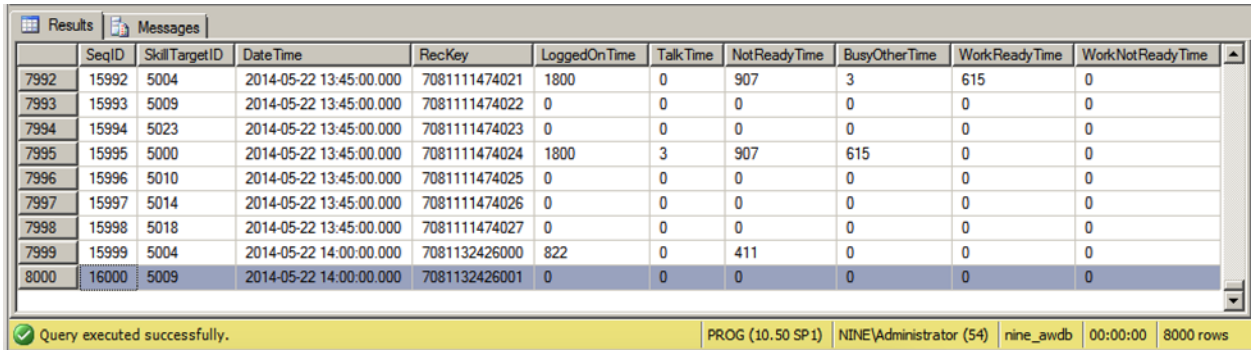


Results | Messages

	SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime	WorkReadyTime	WorkNotReadyTime
1	8001	5009	2013-06-21 18:30:00.000	6746058710130	0	0	0	0	0	0
2	8002	5000	2013-06-21 18:30:00.000	6746058710131	900	0	900	0	0	0
3	8003	5025	2013-06-21 18:45:00.000	6746058710132	900	0	900	0	0	0
4	8004	5004	2013-06-21 18:45:00.000	6746058710133	900	0	900	0	0	0
5	8005	5009	2013-06-21 18:45:00.000	6746058710134	0	0	0	0	0	0
6	8006	5000	2013-06-21 18:45:00.000	6746058710135	900	0	900	0	0	0
7	8007	5025	2013-06-21 19:00:00.000	6746058710136	900	0	900	0	0	0
8	8008	5004	2013-06-21 19:00:00.000	6746058710137	900	0	900	0	0	0
9	8009	5009	2013-06-21 19:00:00.000	6746058710138	0	0	0	0	0	0
10	8010	5000	2013-06-21 19:00:00.000	6746058710139	900	0	900	0	0	0

Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 8000 rows

Last returned rows:



Results | Messages

	SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime	WorkReadyTime	WorkNotReadyTime
7992	15992	5004	2014-05-22 13:45:00.000	7081111474021	1800	0	907	3	615	0
7993	15993	5009	2014-05-22 13:45:00.000	7081111474022	0	0	0	0	0	0
7994	15994	5023	2014-05-22 13:45:00.000	7081111474023	0	0	0	0	0	0
7995	15995	5000	2014-05-22 13:45:00.000	7081111474024	1800	3	907	615	0	0
7996	15996	5010	2014-05-22 13:45:00.000	7081111474025	0	0	0	0	0	0
7997	15997	5014	2014-05-22 13:45:00.000	7081111474026	0	0	0	0	0	0
7998	15998	5018	2014-05-22 13:45:00.000	7081111474027	0	0	0	0	0	0
7999	15999	5004	2014-05-22 14:00:00.000	7081132426000	822	0	411	0	0	0
8000	16000	5009	2014-05-22 14:00:00.000	7081132426001	0	0	0	0	0	0

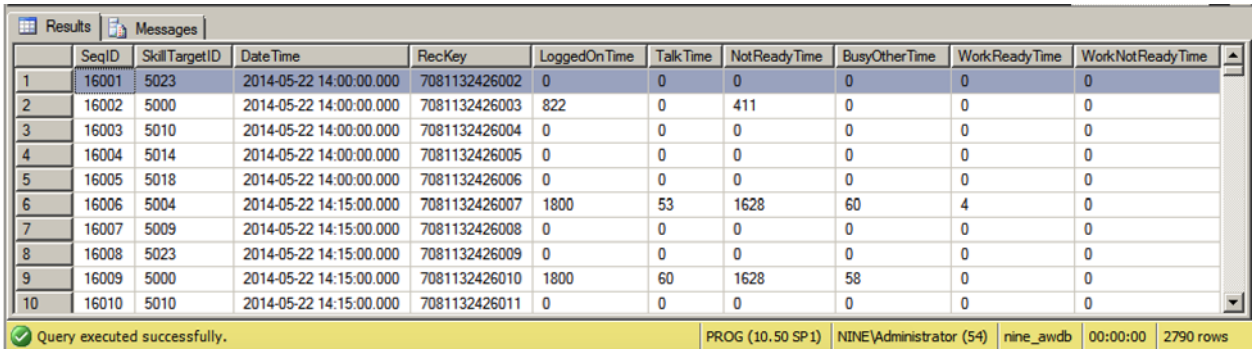
Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 8000 rows

---

When the TSQL is run with the following parameters:

```
SET @i = 3
SET @start='2013-01-01'
SET @end='2015-01-01'
```

First returned rows:

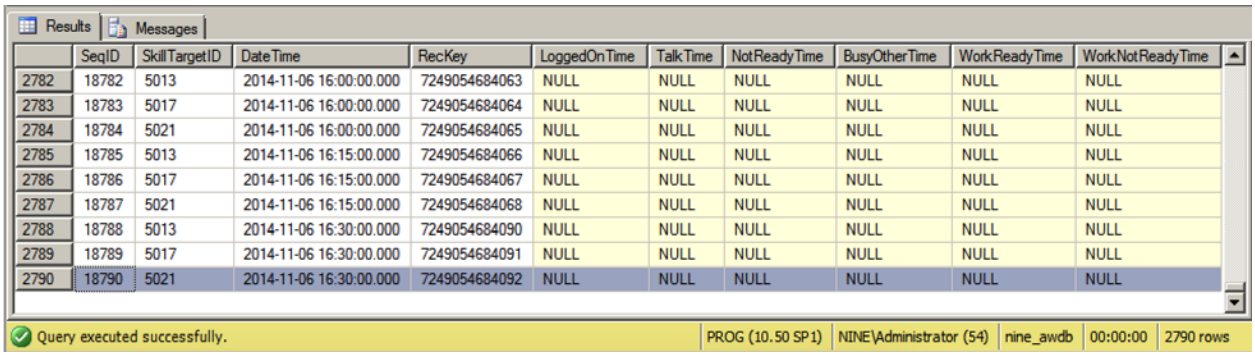


Results | Messages

	SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime	WorkReadyTime	WorkNotReadyTime
1	16001	5023	2014-05-22 14:00:00.000	7081132426002	0	0	0	0	0	0
2	16002	5000	2014-05-22 14:00:00.000	7081132426003	822	0	411	0	0	0
3	16003	5010	2014-05-22 14:00:00.000	7081132426004	0	0	0	0	0	0
4	16004	5014	2014-05-22 14:00:00.000	7081132426005	0	0	0	0	0	0
5	16005	5018	2014-05-22 14:00:00.000	7081132426006	0	0	0	0	0	0
6	16006	5004	2014-05-22 14:15:00.000	7081132426007	1800	53	1628	60	4	0
7	16007	5009	2014-05-22 14:15:00.000	7081132426008	0	0	0	0	0	0
8	16008	5023	2014-05-22 14:15:00.000	7081132426009	0	0	0	0	0	0
9	16009	5000	2014-05-22 14:15:00.000	7081132426010	1800	60	1628	58	0	0
10	16010	5010	2014-05-22 14:15:00.000	7081132426011	0	0	0	0	0	0

Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 2790 rows

Last returned rows:



Results | Messages

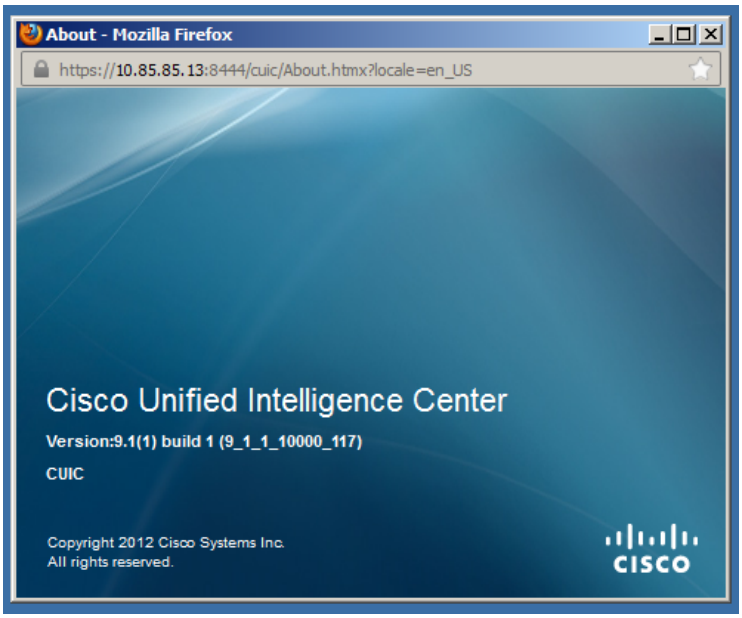
	SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime	WorkReadyTime	WorkNotReadyTime
2782	18782	5013	2014-11-06 16:00:00.000	7249054684063	NULL	NULL	NULL	NULL	NULL	NULL
2783	18783	5017	2014-11-06 16:00:00.000	7249054684064	NULL	NULL	NULL	NULL	NULL	NULL
2784	18784	5021	2014-11-06 16:00:00.000	7249054684065	NULL	NULL	NULL	NULL	NULL	NULL
2785	18785	5013	2014-11-06 16:15:00.000	7249054684066	NULL	NULL	NULL	NULL	NULL	NULL
2786	18786	5017	2014-11-06 16:15:00.000	7249054684067	NULL	NULL	NULL	NULL	NULL	NULL
2787	18787	5021	2014-11-06 16:15:00.000	7249054684068	NULL	NULL	NULL	NULL	NULL	NULL
2788	18788	5013	2014-11-06 16:30:00.000	7249054684090	NULL	NULL	NULL	NULL	NULL	NULL
2789	18789	5017	2014-11-06 16:30:00.000	7249054684091	NULL	NULL	NULL	NULL	NULL	NULL
2790	18790	5021	2014-11-06 16:30:00.000	7249054684092	NULL	NULL	NULL	NULL	NULL	NULL

Query executed successfully. | PROG (10.50 SP1) | NINE\Administrator (54) | nine_awdb | 00:00:00 | 2790 rows

Note that page 3 only returned 2790 rows because it is the last page.

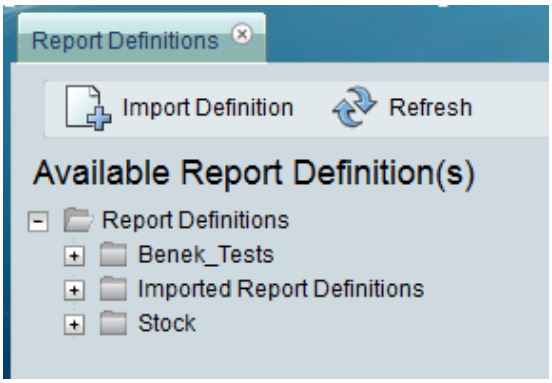
Moving the Logic to CUIC

The CUIC Version used in this example is 9.1(1)

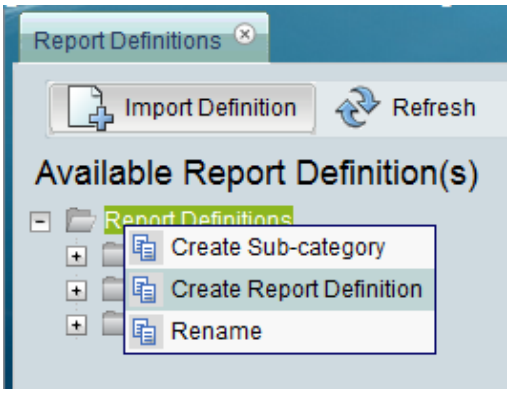


Creating Report Definitions

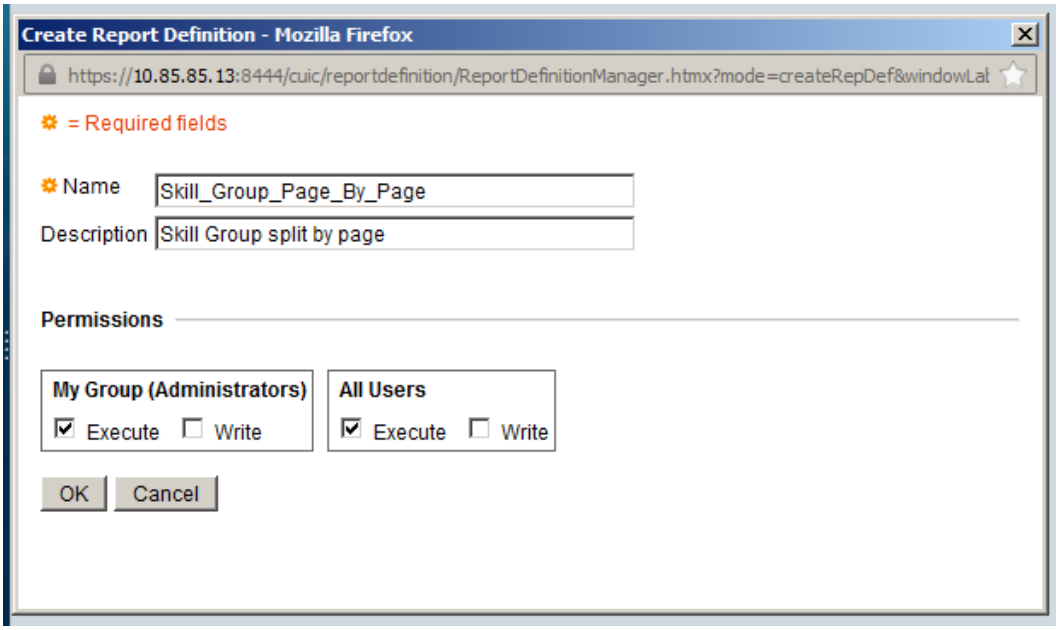
Go to Report Definitions:



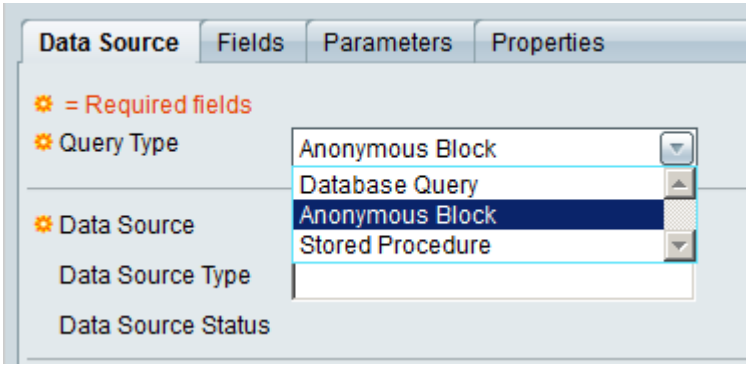
Create Report Definition:



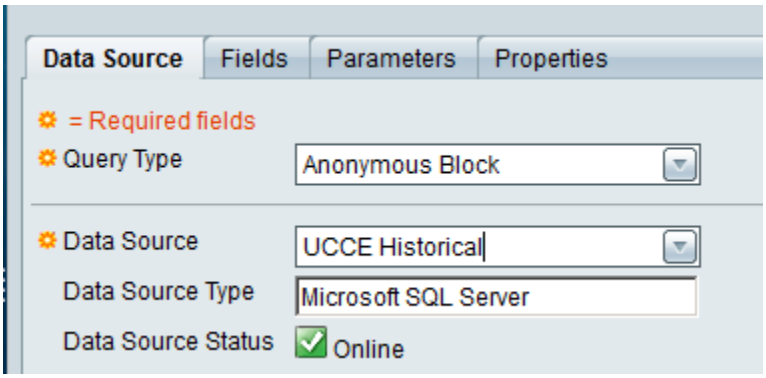
Complete the textboxes for Name and Description:



On the Data Source Tab, The Data Source should be selected as 'Anonymous Block'



Data Source is ‘UCCE Historical’

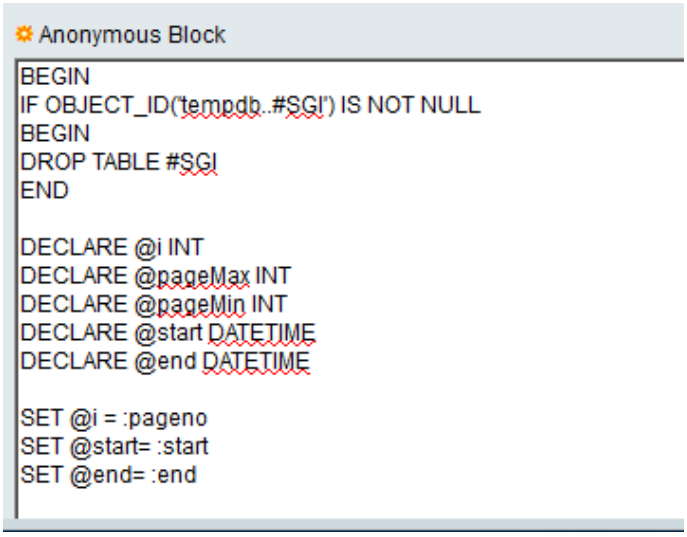


Modify TSQL slightly to fit CUIC for input parameters:

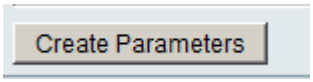
```
BEGIN
IF OBJECT_ID('tempdb..#SGI') IS NOT NULL
BEGIN
DROP TABLE #SGI
END

DECLARE @i INT
DECLARE @pageMax INT
DECLARE @pageMin INT
DECLARE @start DATETIME
DECLARE @end DATETIME

SET @i = :pageno
SET @start = :start
SET @end = :end

IF @i IS NULL OR @i < 1
      SET @i = 1

SET @pageMax = @i*8000
SET @pageMin = (@i-1)*8000 + 1

CREATE TABLE #SGI
(
      SeqID INT IDENTITY(1,1) NOT NULL,
      SkillTargetID INT,
      DateTime DATETIME,
      RecKey FLOAT,
      LoggedOnTime INT,
      TalkTime INT,
      NotReadyTime INT,
      BusyOtherTime INT,
      WorkReadyTime INT,
      WorkNotReadyTime INT
)

INSERT INTO #SGI
(
```

```
	SkillTargetID,
	DateTime,
	RecKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
)
SELECT
	SkillTargetID,
	DateTime,
	RecoveryKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
FROM
	Skill_Group_Interval SGI (nolock)
WHERE
	DateTime BETWEEN @start AND @end
ORDER BY
	RecoveryKey

SELECT
	SeqID,
	SkillTargetID,
	DateTime,
	RecKey,
	LoggedOnTime,
	TalkTime,
	NotReadyTime,
	BusyOtherTime,
	WorkReadyTime,
	WorkNotReadyTime
FROM
	#SGI (nolock)
WHERE
	SeqID BETWEEN @pageMin AND @pageMax

DROP TABLE #SGI
END
```

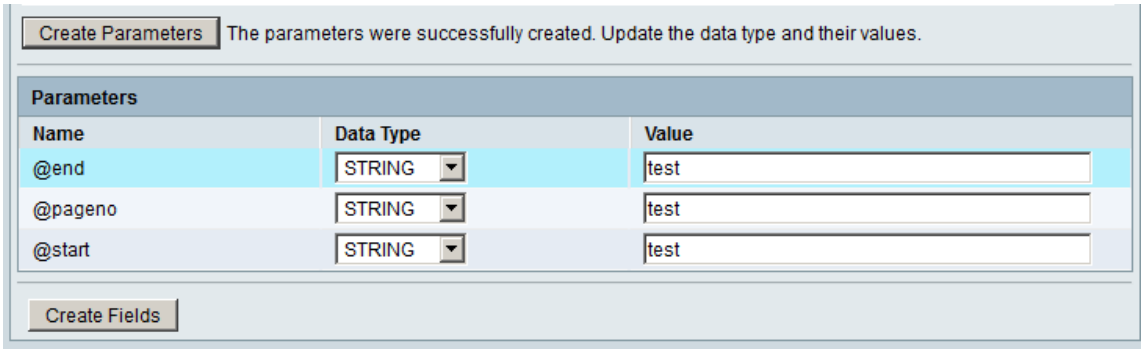
Paste it in the ‘Anonymous Block’ space:



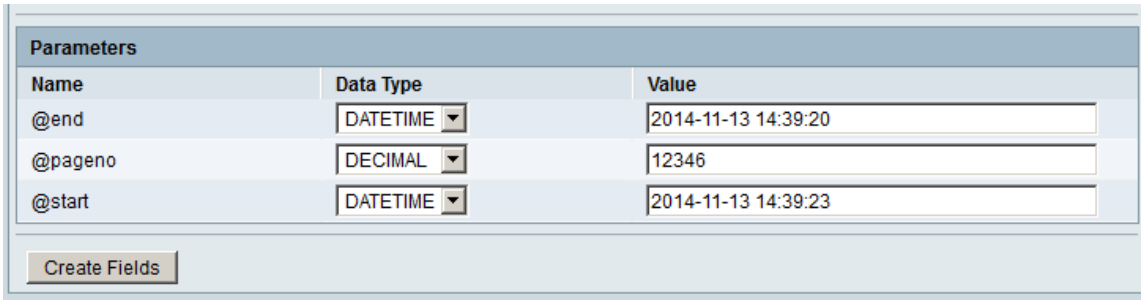
Select ‘Create Parameters’ button and let CUIC parse the query and identify the parameters:



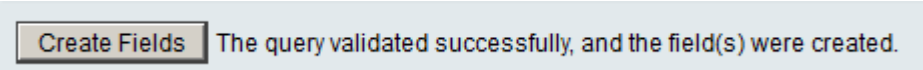
Parameters are identified however in wrong data type:



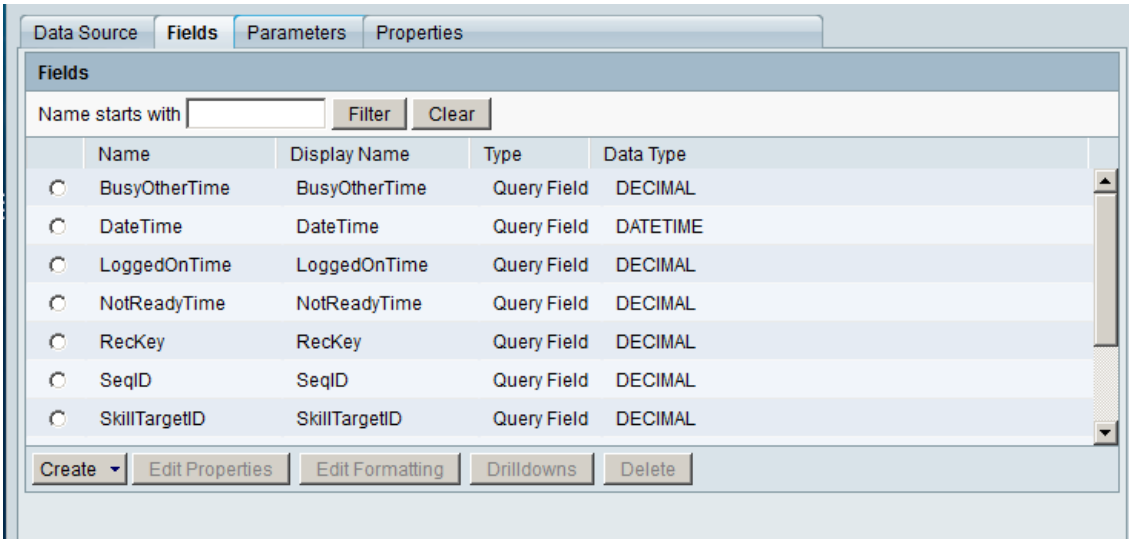
Modify the Parameters’ Data Type where necessary:



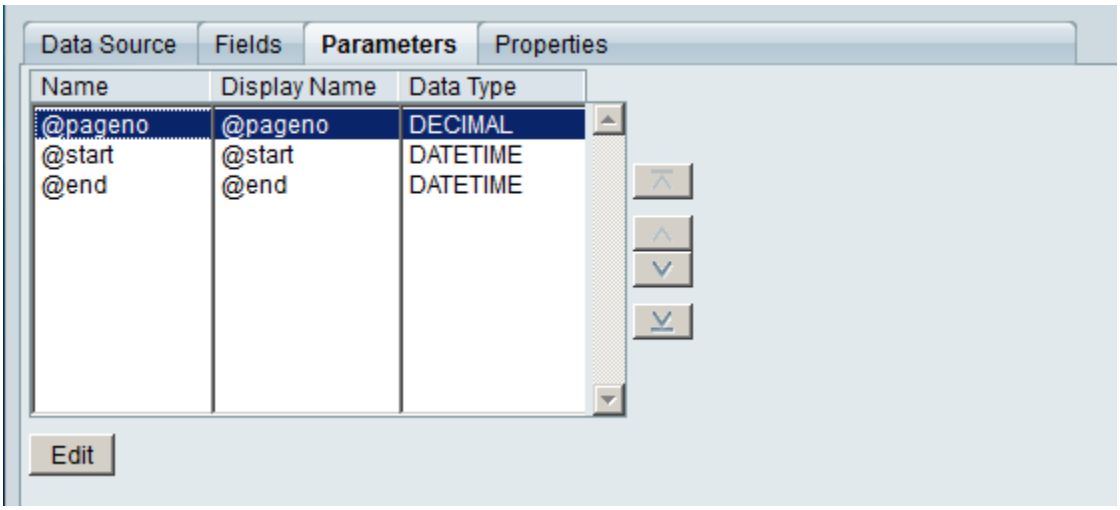
Select ‘Create Fields’



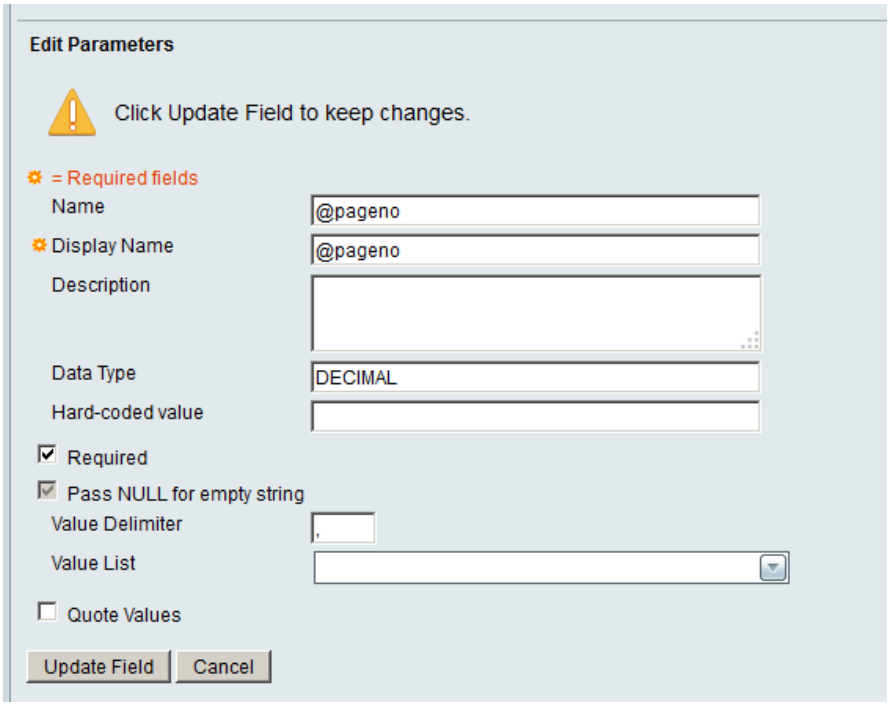
Fields are created:



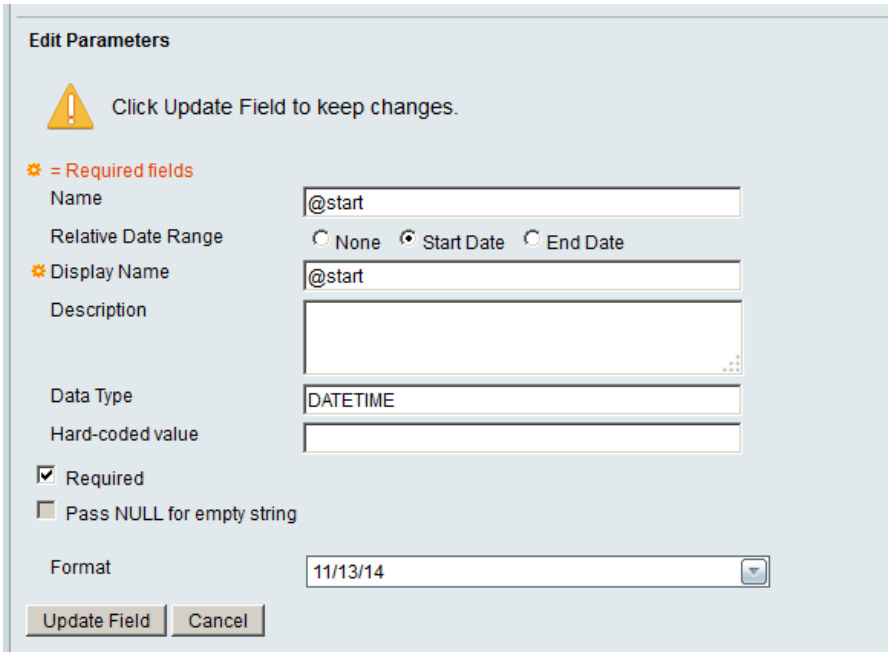
On ‘Parameters’ tab, input parameters can be ordered and modified:



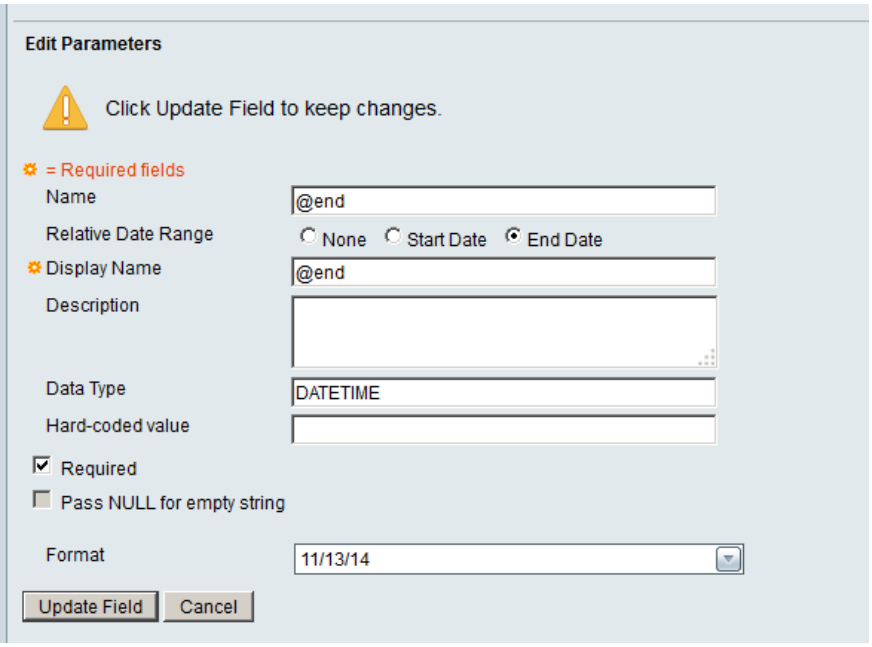
Parameter that defines the Page number:



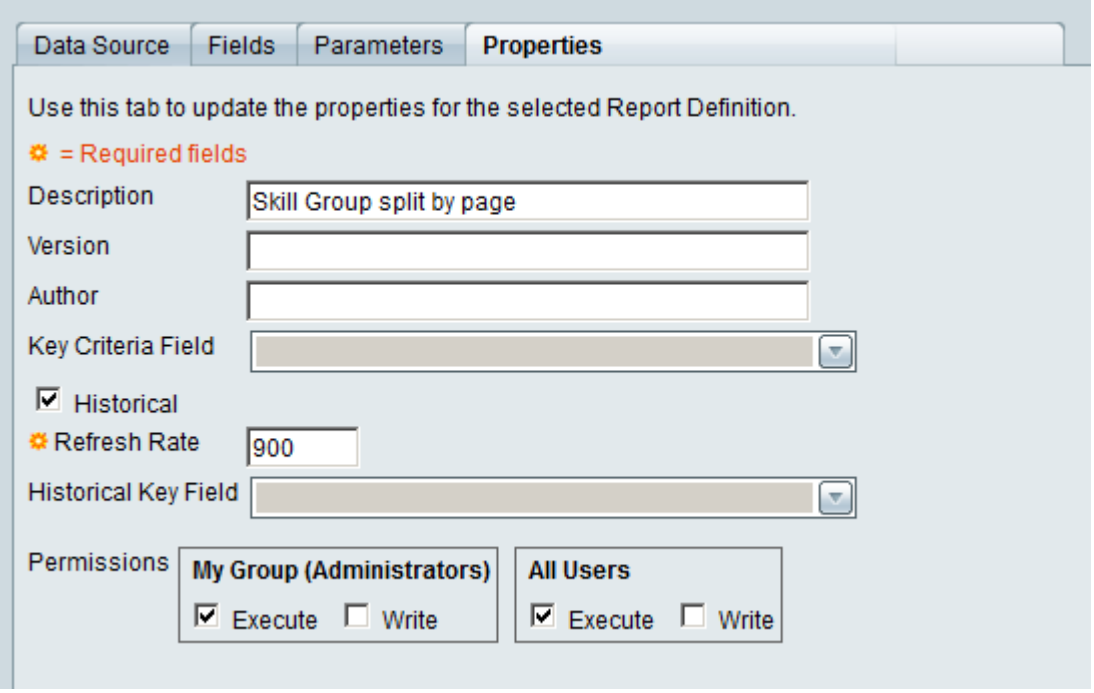
Parameter that defines the Start Date:



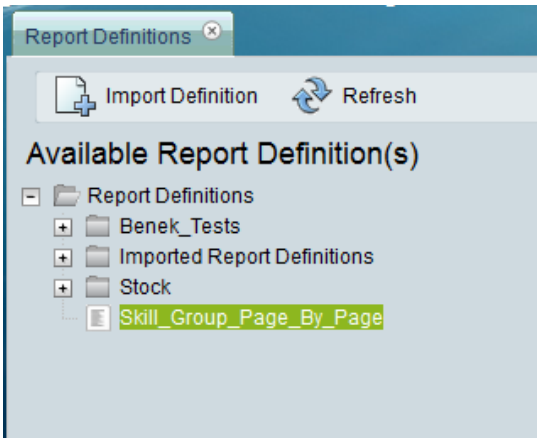
Parameter that defines the End Date:



'Properties' Tab:

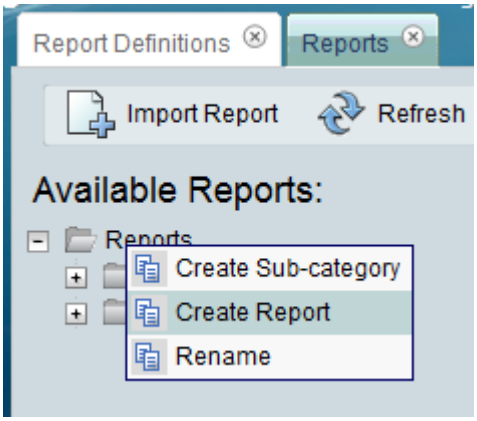


The Report Definition should be saved. It will now be visible under ‘Available Report Definitions’

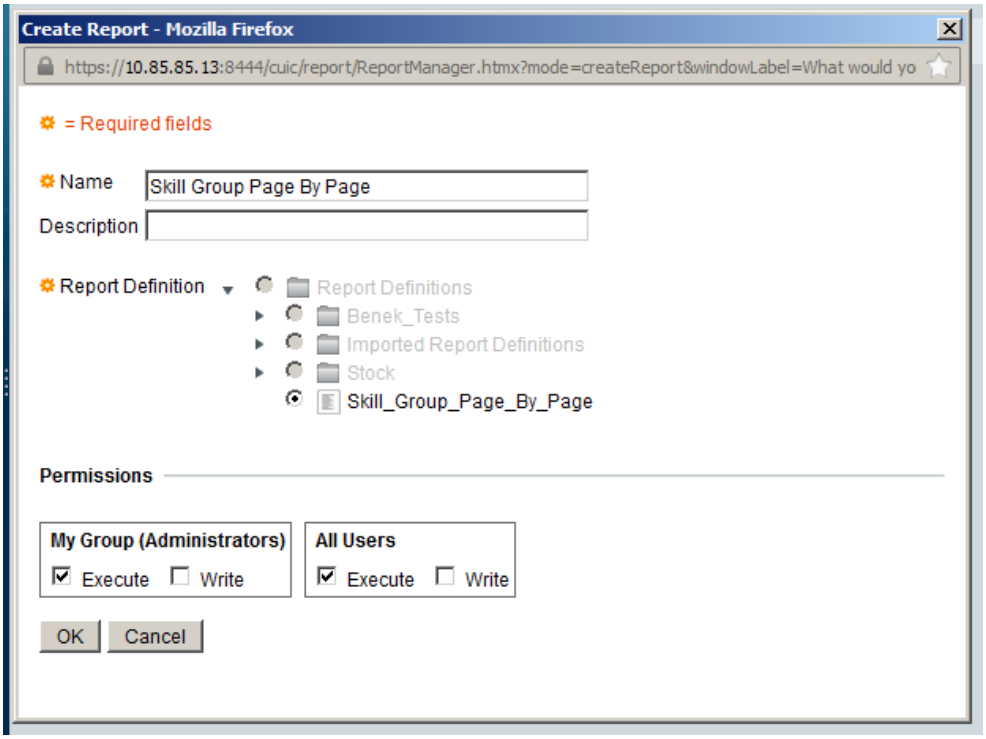


Creating Report Template

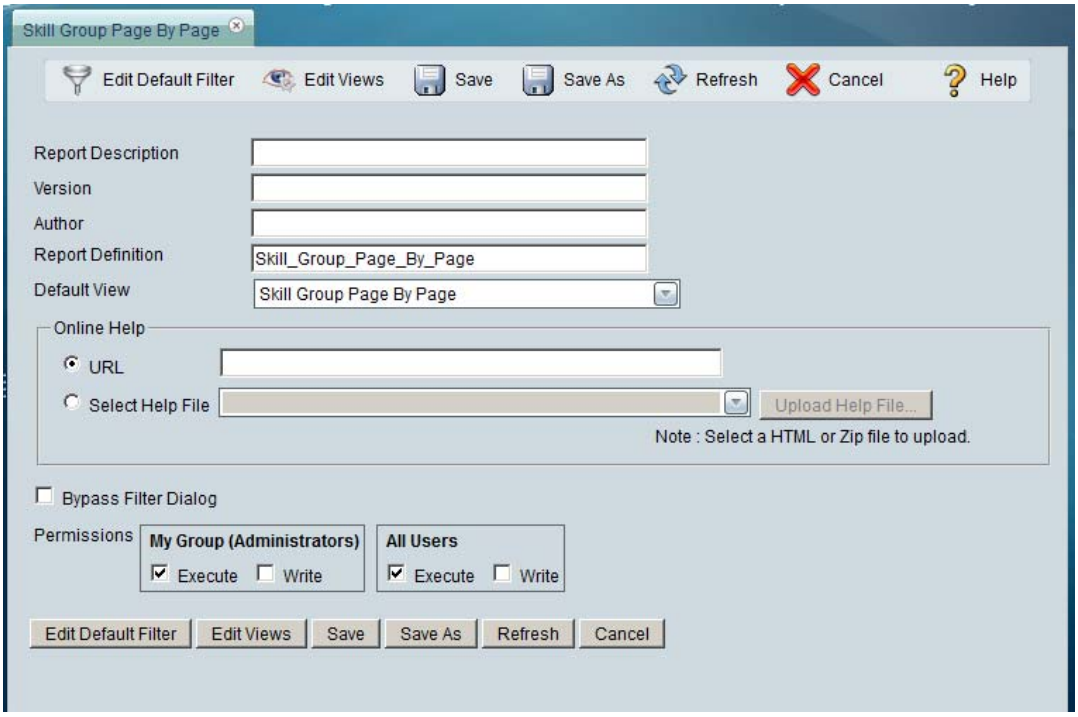
Go to Reports:



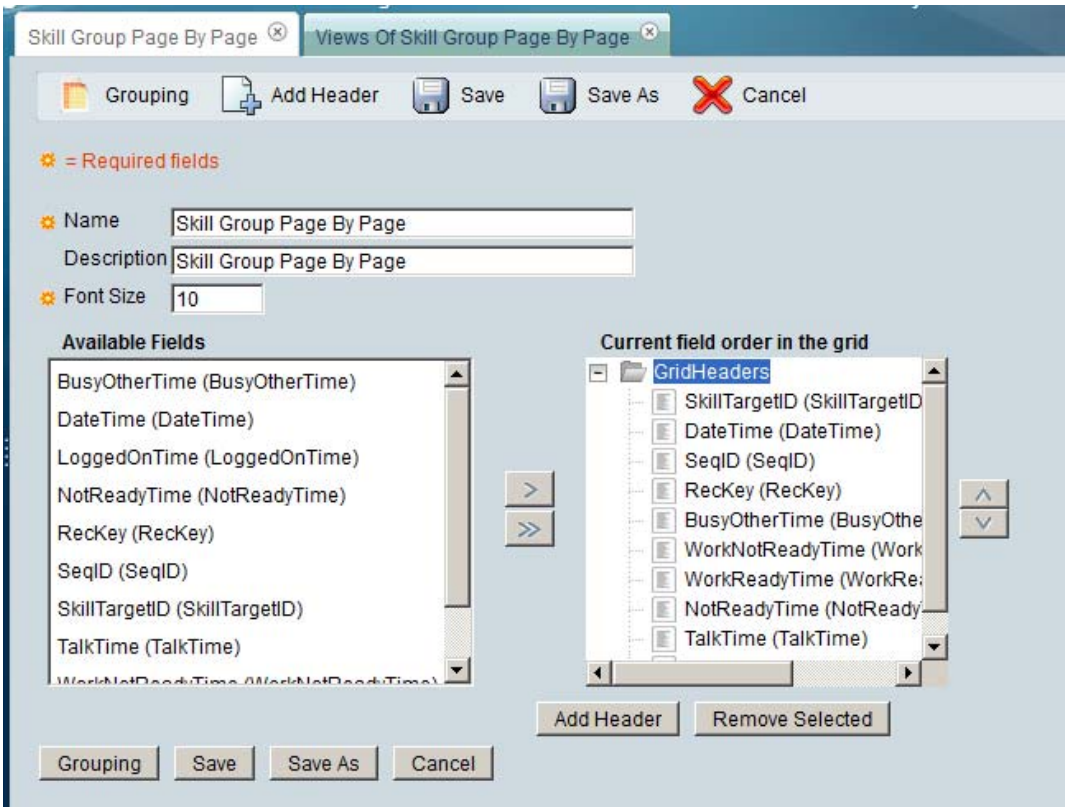
Choose a name and the Report Definition:



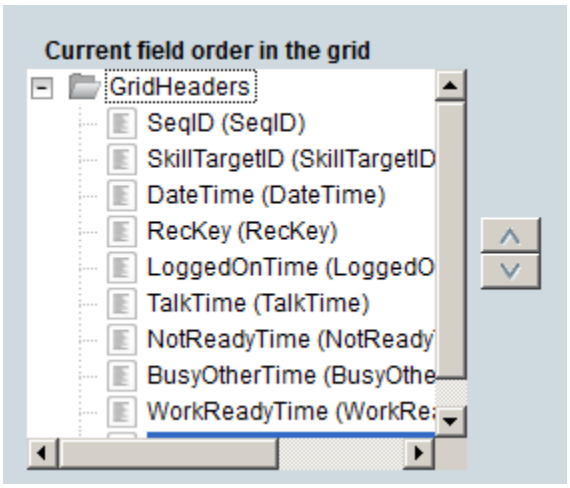
Report Configuration page:



Default View for the Report template:



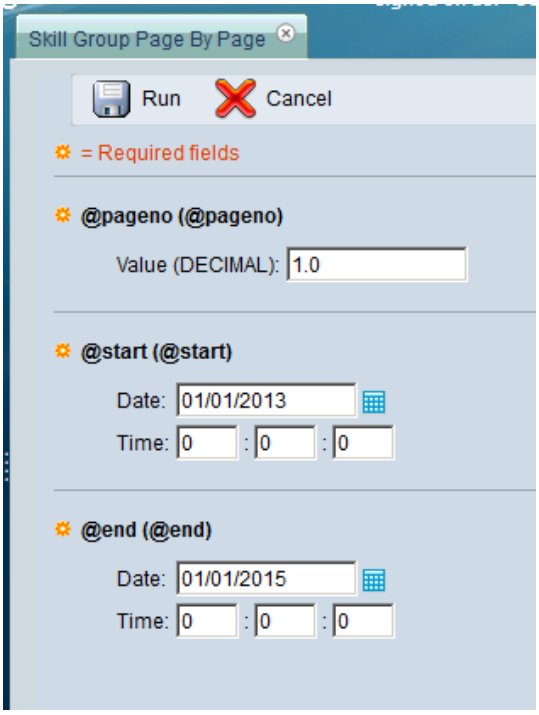
Order Columns as needed:



Save and Exit the Report template.

RUNNING REPORT TEMPLATE

Access the report and use the following inputs in the Filter page:



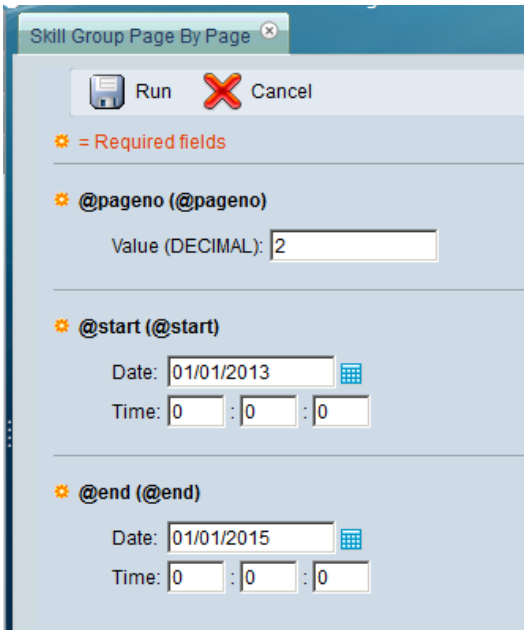
Select 'Run'. The result set will display first 8000 rows:

SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime
1	5004	3/14/13	6647164184001	0	0	0	0
2	5000	3/14/13	6647164184002	0	0	0	0
3	5004	3/14/13	6647164184003	0	0	0	0
4	5000	3/14/13	6647164184004	0	0	0	0
5	5004	3/14/13	6647164184005	0	0	0	0
6	5000	3/14/13	6647164184006	0	0	0	0
7	5004	3/14/13	6647164184007	0	0	0	0
8	5000	3/14/13	6647164184008	0	0	0	0
9	5004	3/14/13	6647164184009	564	0	8	0
10	5000	3/14/13	6647164184010	564	0	8	0
11	5004	3/14/13	6647164184011	900	0	0	0
12	5000	3/14/13	6647164184012	900	0	0	0
13	5004	3/14/13	6647164184013	900	0	0	0
14	5000	3/14/13	6647164184014	900	0	0	0
15	5004	3/14/13	6647164184015	900	0	0	0
16	5000	3/14/13	6647164184016	900	0	0	0
17	5004	3/14/13	6647164184017	900	0	0	0
18	5000	3/14/13	6647164184018	900	0	0	0
19	5004	3/14/13	6647164184019	900	0	0	0
20	5000	3/14/13	6647164184020	900	0	0	0
21	5004	3/14/13	6647164184021	900	0	0	0
22	5000	3/14/13	6647164184022	900	0	0	0

SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime
7979	5025	6/21/13	6746058710108	900	0	900	0
7980	5004	6/21/13	6746058710109	900	0	900	0
7981	5009	6/21/13	6746058710110	0	0	0	0
7982	5000	6/21/13	6746058710111	900	0	900	0
7983	5025	6/21/13	6746058710112	900	0	900	0
7984	5004	6/21/13	6746058710113	900	0	900	0
7985	5009	6/21/13	6746058710114	0	0	0	0
7986	5000	6/21/13	6746058710115	900	0	900	0
7987	5025	6/21/13	6746058710116	900	0	900	0
7988	5004	6/21/13	6746058710117	900	0	900	0
7989	5009	6/21/13	6746058710118	0	0	0	0
7990	5000	6/21/13	6746058710119	900	0	900	0
7991	5025	6/21/13	6746058710120	900	0	900	0
7992	5004	6/21/13	6746058710121	900	0	900	0
7993	5009	6/21/13	6746058710122	0	0	0	0
7994	5000	6/21/13	6746058710123	900	0	900	0
7995	5025	6/21/13	6746058710124	900	0	900	0
7996	5004	6/21/13	6746058710125	900	0	900	0
7997	5009	6/21/13	6746058710126	0	0	0	0
7998	5000	6/21/13	6746058710127	900	0	900	0
7999	5025	6/21/13	6746058710128	900	0	900	0
8000	5004	6/21/13	6746058710129	900	0	900	0

To see 8000 onwards (until 16000), Select Filter button on top:

Set the @pageno parameter to 2 for the second page:



As expected, second page is populated starting from Row ID 8001…

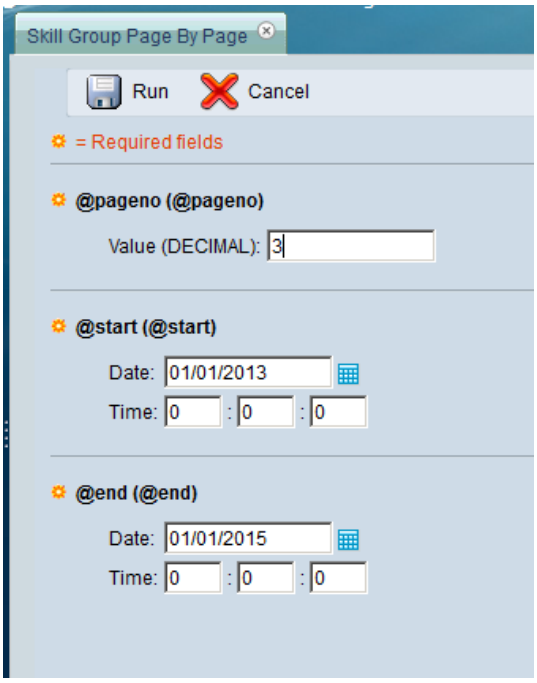
SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime
8001	5009	6/21/13	6746058710130	0	0	0	0
8002	5000	6/21/13	6746058710131	900	0	900	0
8003	5025	6/21/13	6746058710132	900	0	900	0
8004	5004	6/21/13	6746058710133	900	0	900	0
8005	5009	6/21/13	6746058710134	0	0	0	0
8006	5000	6/21/13	6746058710135	900	0	900	0
8007	5025	6/21/13	6746058710136	900	0	900	0
8008	5004	6/21/13	6746058710137	900	0	900	0
8009	5009	6/21/13	6746058710138	0	0	0	0
8010	5000	6/21/13	6746058710139	900	0	900	0
8011	5025	6/21/13	6746058710140	900	0	900	0
8012	5004	6/21/13	6746058710141	900	0	900	0
8013	5009	6/21/13	6746058710142	0	0	0	0
8014	5000	6/21/13	6746058710143	900	0	900	0
8015	5025	6/21/13	6746058710144	900	0	900	0
8016	5004	6/21/13	6746058710145	900	0	900	0
8017	5009	6/21/13	6746058710146	0	0	0	0
8018	5000	6/21/13	6746058710147	900	0	900	0
8019	5025	6/21/13	6746058710148	900	0	900	0
8020	5004	6/21/13	6746058710149	900	0	900	0
8021	5009	6/21/13	6746058710150	0	0	0	0
8022	5000	6/21/13	6746058710151	900	0	900	0

…all the way to 16000:

SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime
15979	5009	5/22/14	7081111474008	0	0	0	0
15980	5023	5/22/14	7081111474009	0	0	0	0
15981	5000	5/22/14	7081111474010	1800	0	884	875
15982	5010	5/22/14	7081111474011	0	0	0	0
15983	5014	5/22/14	7081111474012	0	0	0	0
15984	5018	5/22/14	7081111474013	0	0	0	0
15985	5004	5/22/14	7081111474014	1800	0	900	0
15986	5009	5/22/14	7081111474015	0	0	0	0
15987	5023	5/22/14	7081111474016	0	0	0	0
15988	5000	5/22/14	7081111474017	1800	0	900	900
15989	5010	5/22/14	7081111474018	0	0	0	0
15990	5014	5/22/14	7081111474019	0	0	0	0
15991	5018	5/22/14	7081111474020	0	0	0	0
15992	5004	5/22/14	7081111474021	1800	0	907	3
15993	5009	5/22/14	7081111474022	0	0	0	0
15994	5023	5/22/14	7081111474023	0	0	0	0
15995	5000	5/22/14	7081111474024	1800	3	907	615
15996	5010	5/22/14	7081111474025	0	0	0	0
15997	5014	5/22/14	7081111474026	0	0	0	0
15998	5018	5/22/14	7081111474027	0	0	0	0
15999	5004	5/22/14	7081132426000	822	0	411	0
16000	5009	5/22/14	7081132426001	0	0	0	0

Change filter one more time:

Change @pageno input to 3:



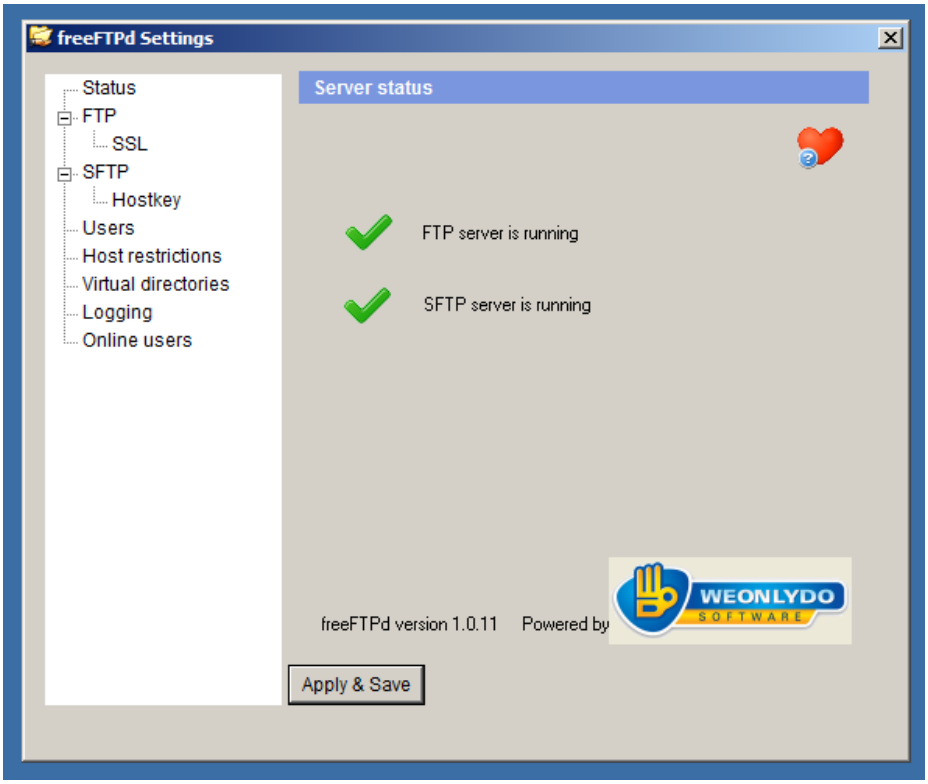
Last 2790 record is displayed:

SeqID	SkillTargetID	DateTime	RecKey	LoggedOnTime	TalkTime	NotReadyTime	BusyOtherTime
18770	5013	11/6/14	7249054684051				
18771	5017	11/6/14	7249054684052				
18772	5021	11/6/14	7249054684053				
18773	5013	11/6/14	7249054684054				
18774	5017	11/6/14	7249054684055				
18775	5021	11/6/14	7249054684056				
18776	5013	11/6/14	7249054684057				
18777	5017	11/6/14	7249054684058				
18778	5021	11/6/14	7249054684059				
18779	5013	11/6/14	7249054684060				
18780	5017	11/6/14	7249054684061				
18781	5021	11/6/14	7249054684062				
18782	5013	11/6/14	7249054684063				
18783	5017	11/6/14	7249054684064				
18784	5021	11/6/14	7249054684065				
18785	5013	11/6/14	7249054684066				
18786	5017	11/6/14	7249054684067				
18787	5021	11/6/14	7249054684068				
18788	5013	11/6/14	7249054684090				
18789	5017	11/6/14	7249054684091				
18790	5021	11/6/14	7249054684092				

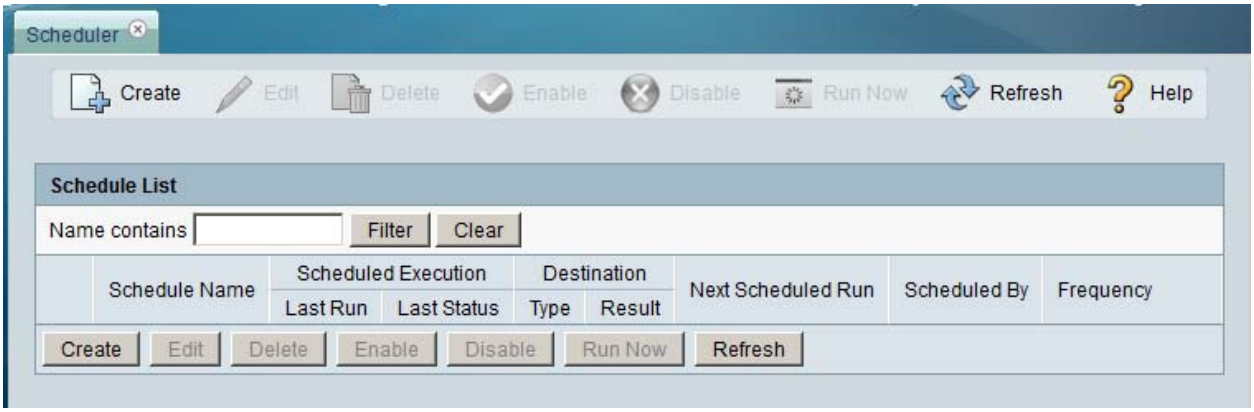
AUTOMATION WITH SCHEDULES

While an end user can easily benefit from this paging approach, the reports with excessive row count and require this may be scheduled and automated for CSV file creation. This way, users may find CSV files already populated in FTP folders for historical reports the next day when they are scheduled overnight.

In this example, the free FTP software; freeFTPd is configured:

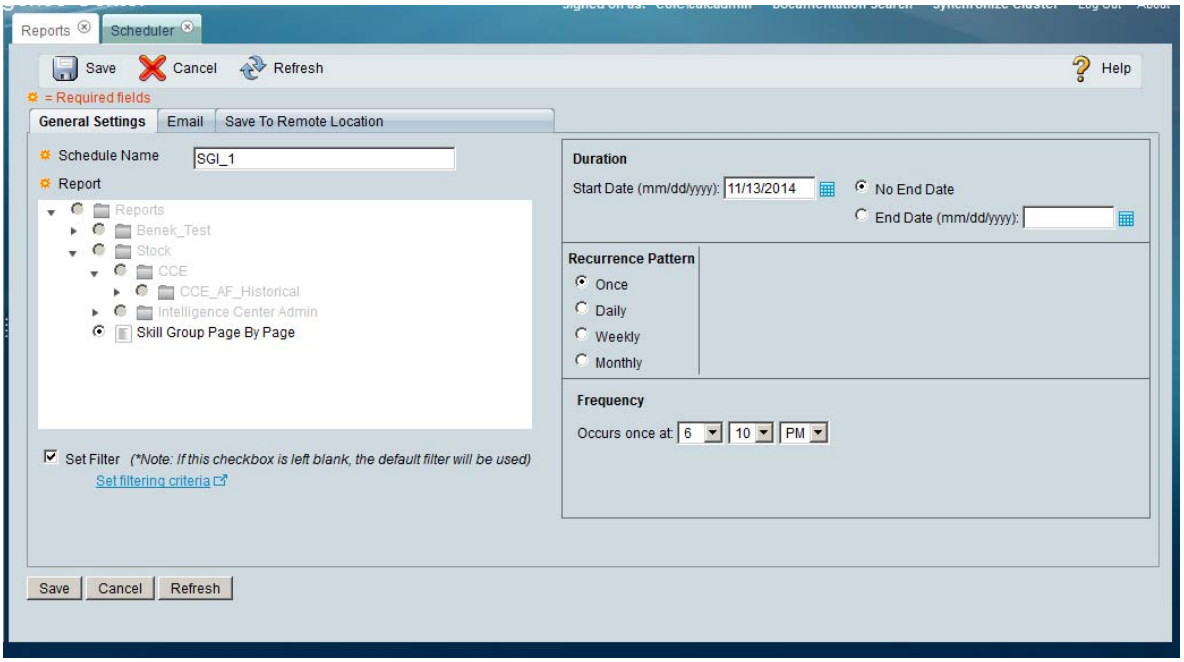


On CUIC, *Scheduler* must be accessed:

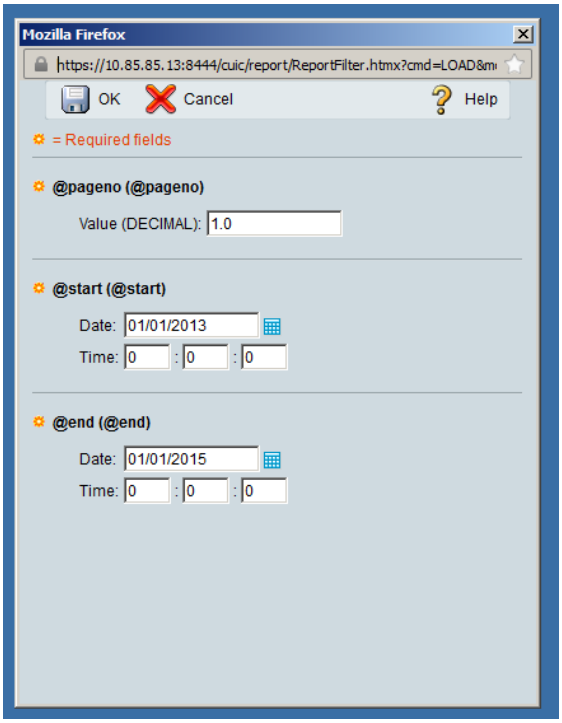


Select ‘Create’. Enter a ‘Schedule Name’ and Choose Report named ‘Skill Group Page By Page’. Set the Duration, Recurrence Pattern and Frequency as required. In this example, the report will be run once at 18:10.

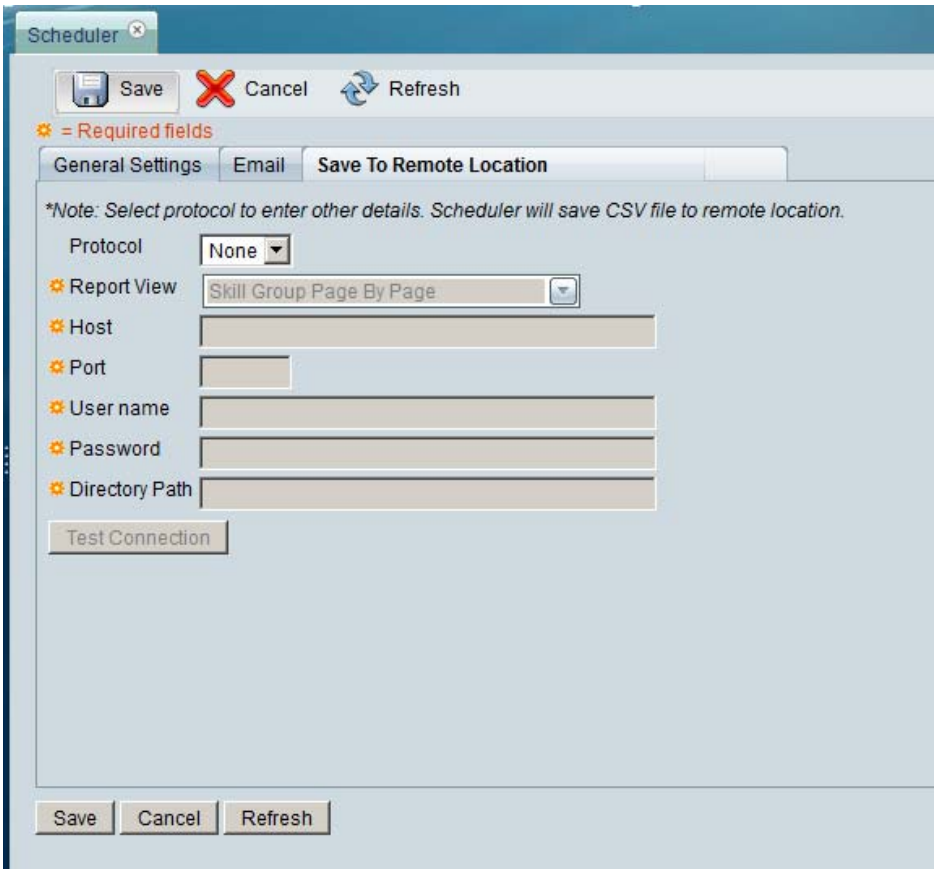
Check the ‘Set Filter’ option to provide filtering criteria.



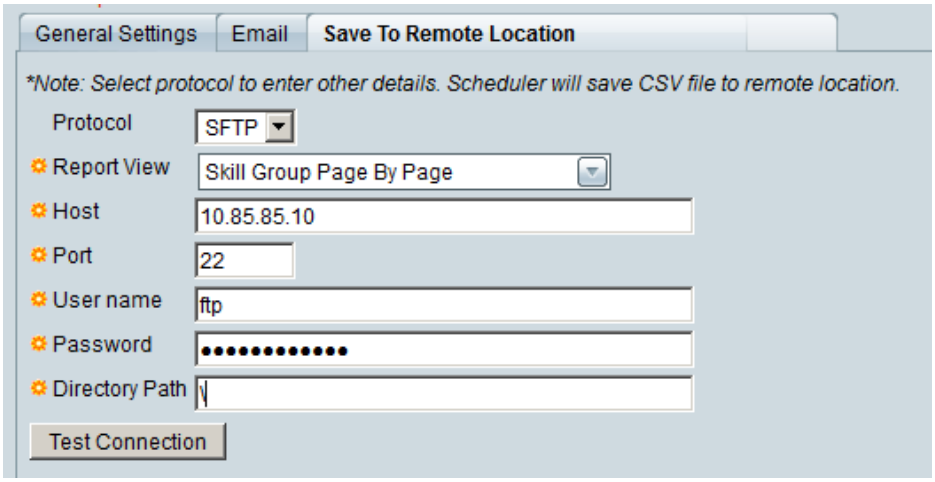
Fill the parameters in the Filter screen and select OK. For first schedule, pageno parameter will be 1.



Go to ‘Save to Remote Location’ Tab



Enter required SFTP credentials:

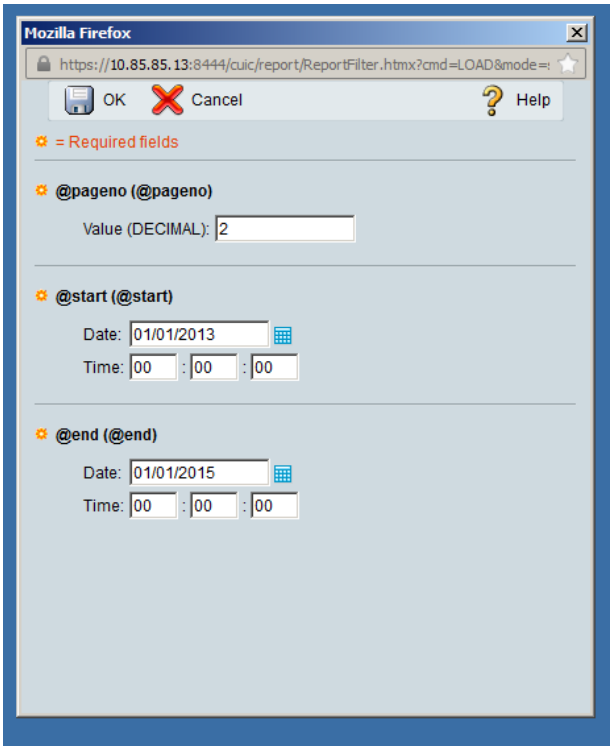


Use Test Connection to validate the inputs provided:

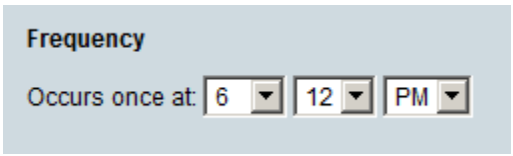
Selecting ‘Save’ will configure the Schedule.

Two more Schedules will be configured identical, however with the following differences on ‘Set Filter’ option and occurrence.

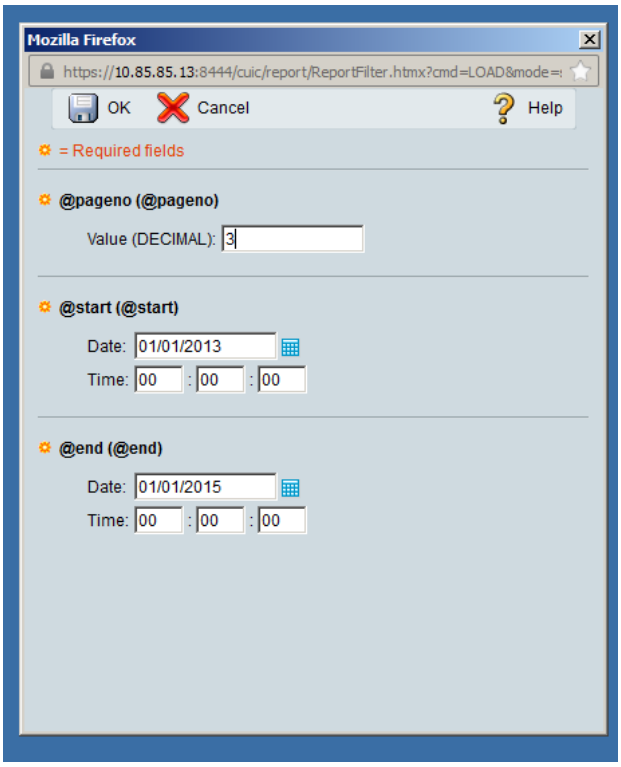
Second Schedule parameters:



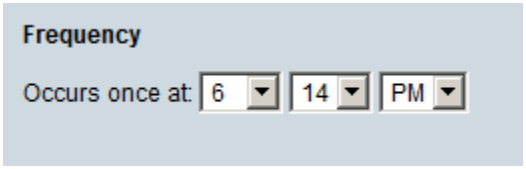
The Frequency and occurrence of Second Schedule:



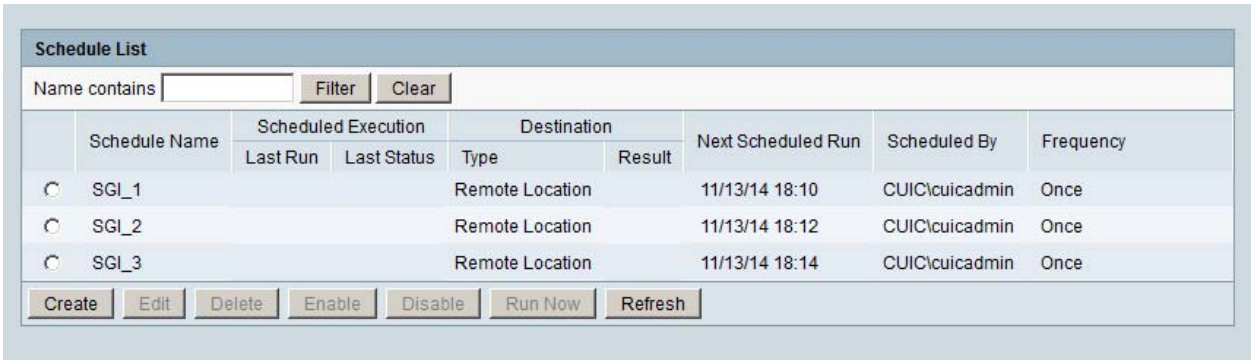
Third Schedule parameters:



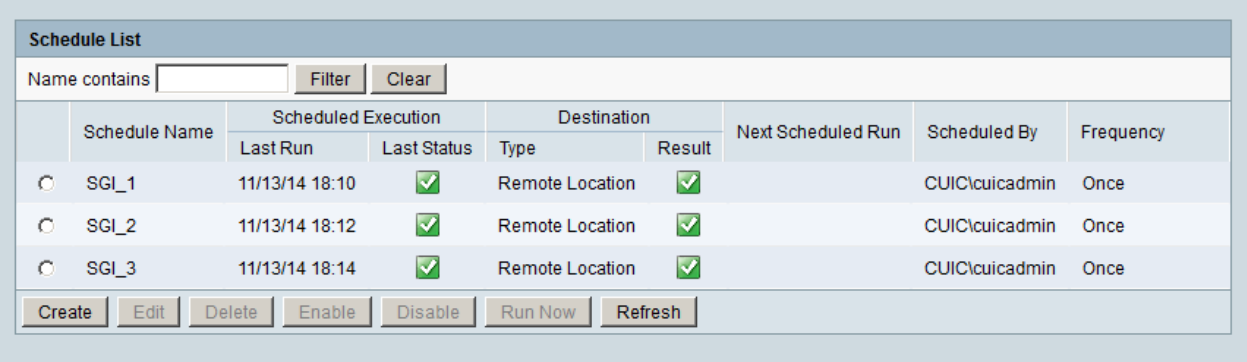
The Frequency of Third Schedule:



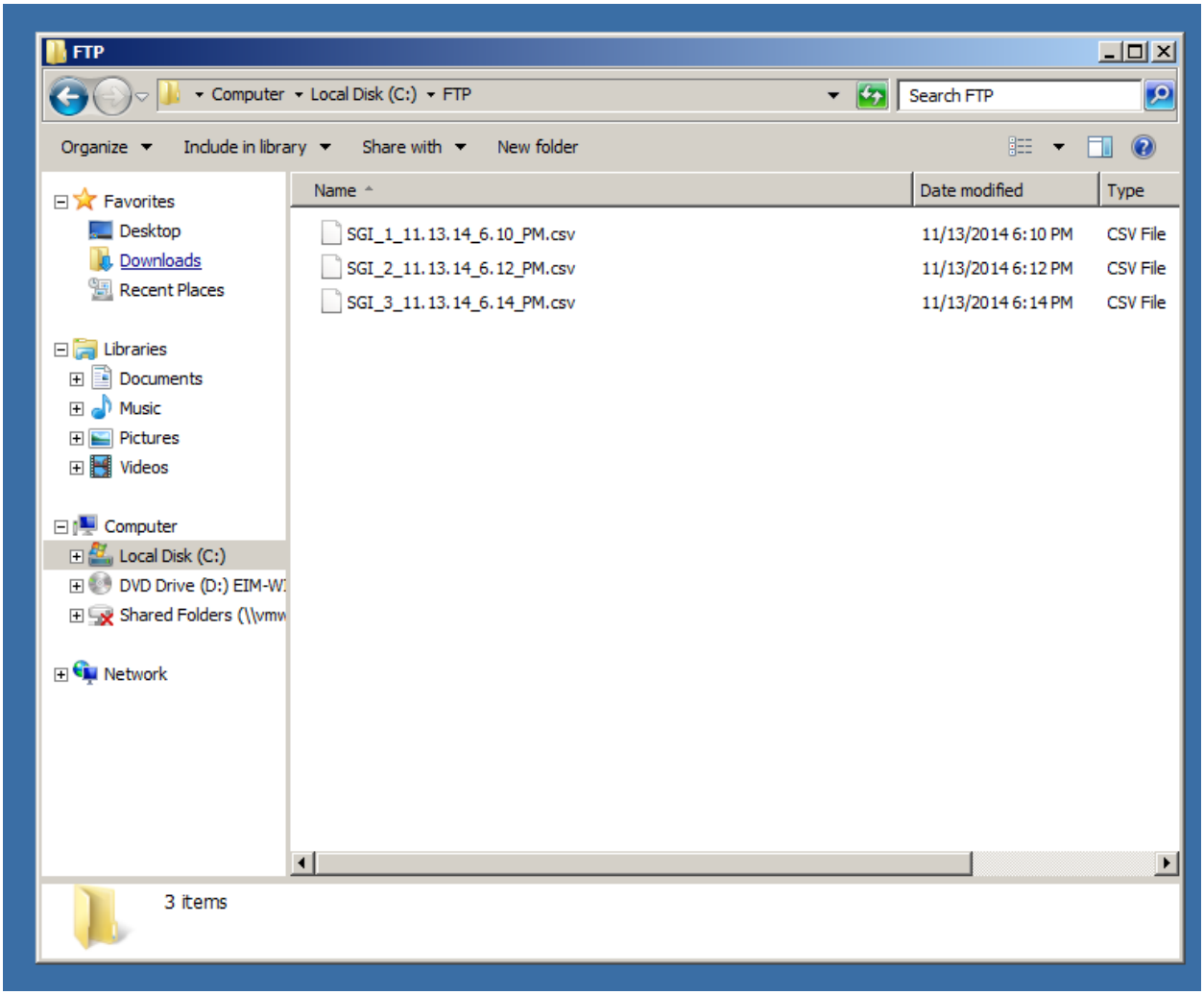
All Schedules are configured to run Once:



When the scheduled time arrives, the Schedules are run as expected in their respectively set times.



The files can be found in the configured FTP location after the runs are completed:



MERGING FILES

Following on the Schedules after run is completed; the end user can further be exempted from dealing with multiple Excel CSV files of the same report (with multiple pages) by merging the output files into one. To achieve this goal, the internet is an endless point of resources with both commercial and free solutions to choose from.

A very simple windows batch script will be (modified and) used for this example. The original script code was found in the following address:

http://stackoverflow.com/questions/14690276/combine-csv-files-in-windows-cmd-or-bat-file-preferably

Original Code:

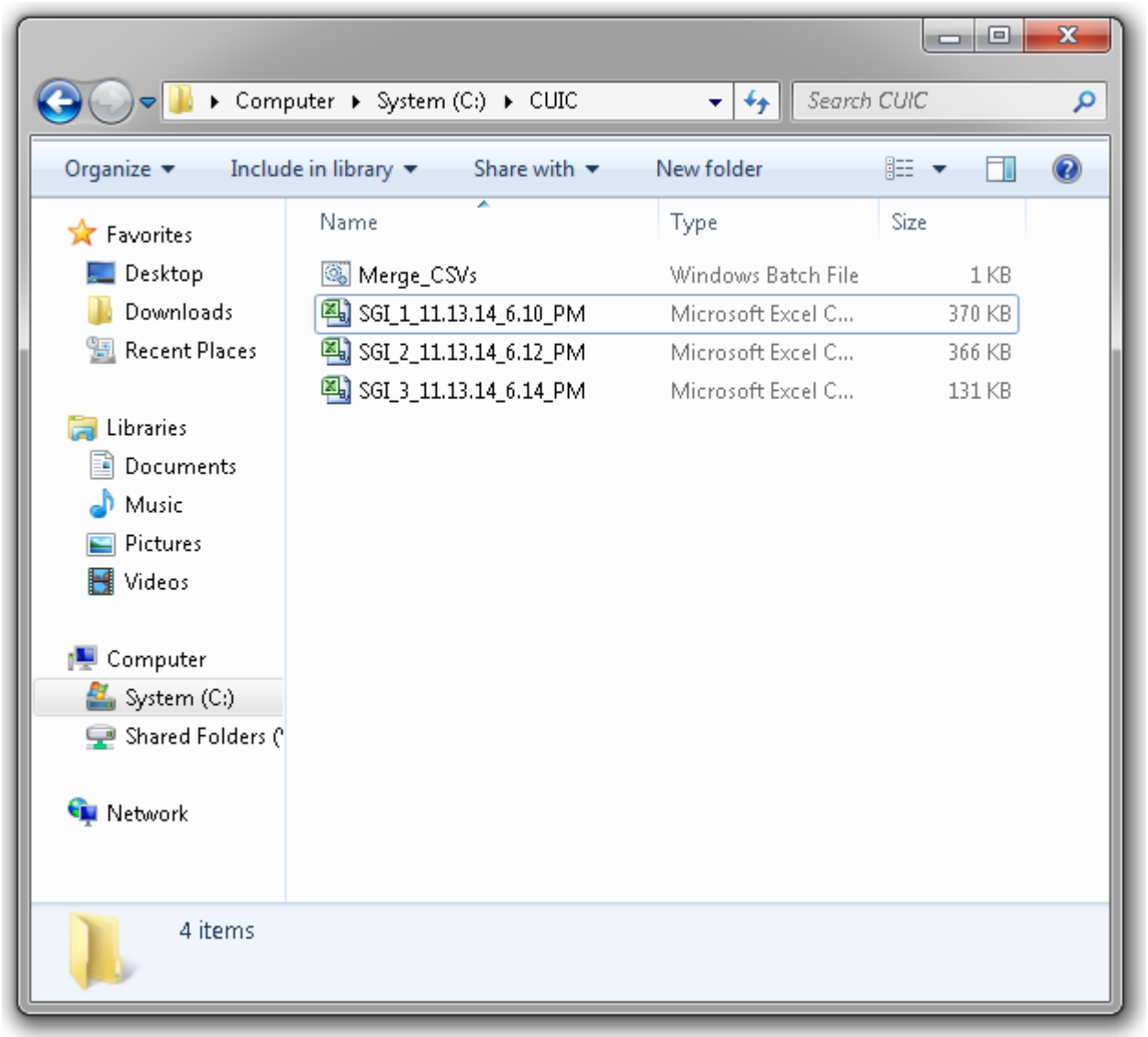
```
@echo off

setlocal ENABLEDELAYEDEXPANSION

set cnt=1
cd "Desktop\[csv-files]"

for %%i in (*.csv) do (
  if !cnt!==1 (
    for /f "delims=" %%j in ('type "%%i"') do echo %%j >> my-new-file.txt
  ) else (
    for /f "skip=1 delims=" %%j in ('type "%%i"') do echo %%j >> my-new-
file.txt
  )
  set /a cnt+=1
)

setlocal

ren my-new-file.txt my-new-file.csv
```

The code is modified for folder location of the CSV files and saved as ‘Merge_CSVs.bat’

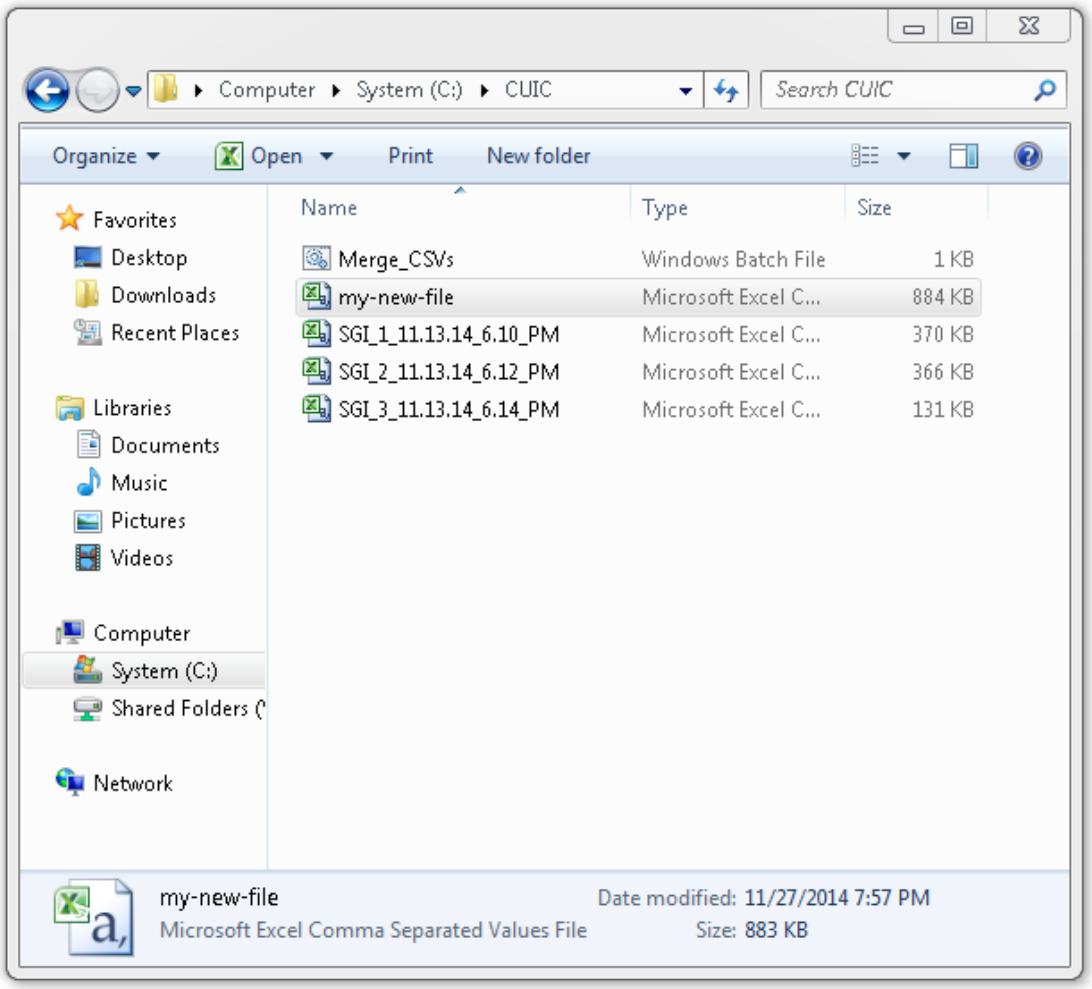
```
@echo off

setlocal ENABLEDELAYEDEXPANSION

set cnt=1
:: Set the Folder Location for saved Excel CSV files
cd "C:\CUIC"

for %%i in (*.csv) do (
  if !cnt!==1 (
    for /f "delims=" %%j in ('type "%%i"') do echo %%j >> my-new-file.txt
  ) else (
    for /f "skip=1 delims=" %%j in ('type "%%i"') do echo %%j >> my-new-
file.txt
  )
  set /a cnt+=1
```

```
)

setlocal

ren my-new-file.txt my-new-file.csv
```

The location of CSV files:

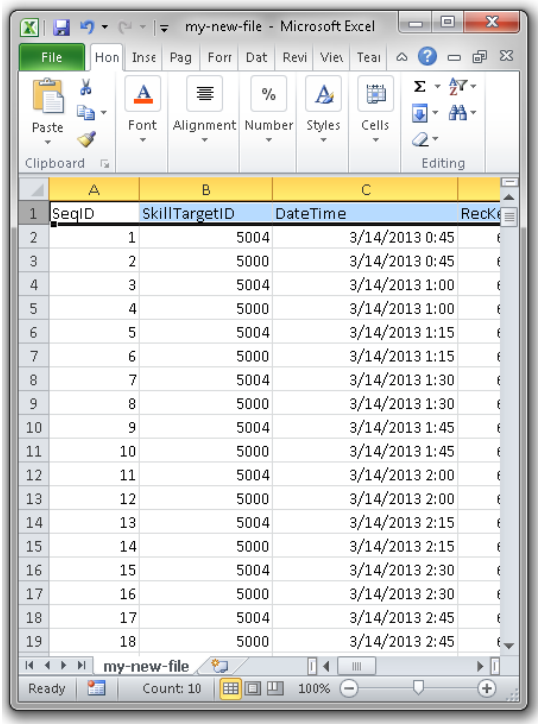


When the batch file runs, a new CSV file is created and saved under the same location:



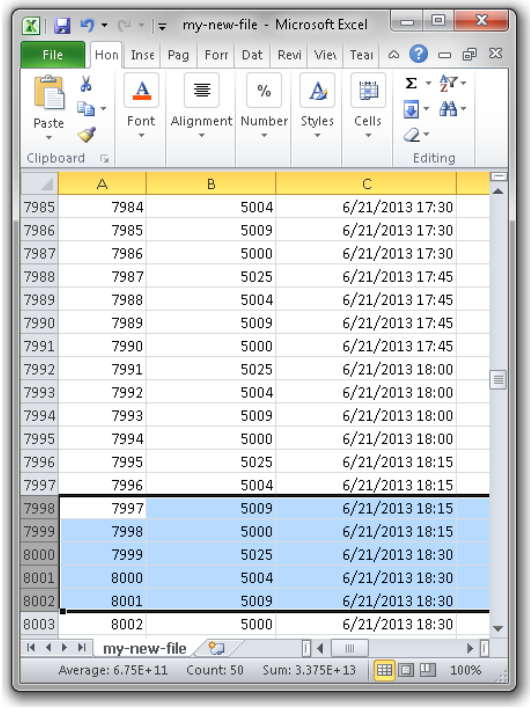
As expected, the newly created my-new-file.csv has all the rows merged:

Beginning of first page:



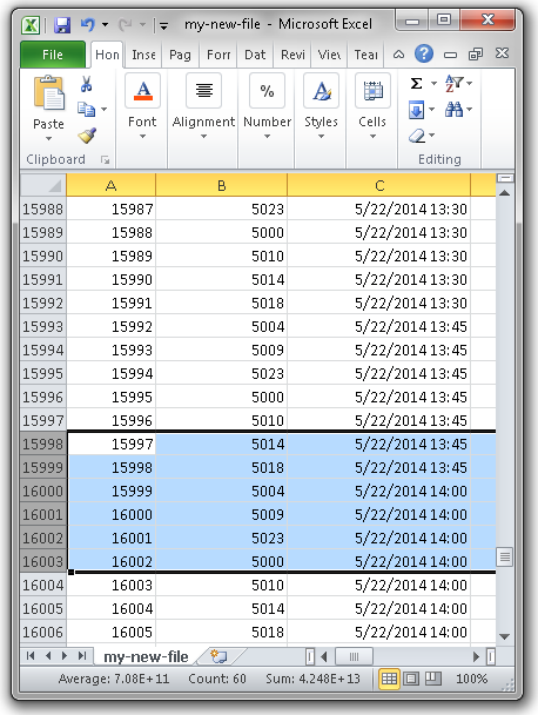
	A	B	C
1	SeqID	SkillTargetID	DateTime
2	1	5004	3/14/2013 0:45
3	2	5000	3/14/2013 0:45
4	3	5004	3/14/2013 1:00
5	4	5000	3/14/2013 1:00
6	5	5004	3/14/2013 1:15
7	6	5000	3/14/2013 1:15
8	7	5004	3/14/2013 1:30
9	8	5000	3/14/2013 1:30
10	9	5004	3/14/2013 1:45
11	10	5000	3/14/2013 1:45
12	11	5004	3/14/2013 2:00
13	12	5000	3/14/2013 2:00
14	13	5004	3/14/2013 2:15
15	14	5000	3/14/2013 2:15
16	15	5004	3/14/2013 2:30
17	16	5000	3/14/2013 2:30
18	17	5004	3/14/2013 2:45
19	18	5000	3/14/2013 2:45

Ending of first page/beginning of second page:



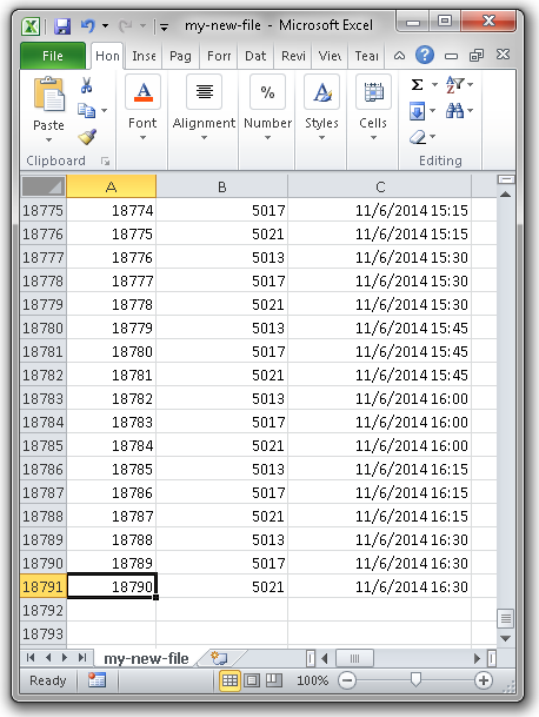
	A	B	C
7985	7984	5004	6/21/2013 17:30
7986	7985	5009	6/21/2013 17:30
7987	7986	5000	6/21/2013 17:30
7988	7987	5025	6/21/2013 17:45
7989	7988	5004	6/21/2013 17:45
7990	7989	5009	6/21/2013 17:45
7991	7990	5000	6/21/2013 17:45
7992	7991	5025	6/21/2013 18:00
7993	7992	5004	6/21/2013 18:00
7994	7993	5009	6/21/2013 18:00
7995	7994	5000	6/21/2013 18:00
7996	7995	5025	6/21/2013 18:15
7997	7996	5004	6/21/2013 18:15
7998	7997	5009	6/21/2013 18:15
7999	7998	5000	6/21/2013 18:15
8000	7999	5025	6/21/2013 18:30
8001	8000	5004	6/21/2013 18:30
8002	8001	5009	6/21/2013 18:30
8003	8002	5000	6/21/2013 18:30

Ending of second page/beginning of third (final) page:



	A	B	C
15988	15987	5023	5/22/2014 13:30
15989	15988	5000	5/22/2014 13:30
15990	15989	5010	5/22/2014 13:30
15991	15990	5014	5/22/2014 13:30
15992	15991	5018	5/22/2014 13:30
15993	15992	5004	5/22/2014 13:45
15994	15993	5009	5/22/2014 13:45
15995	15994	5023	5/22/2014 13:45
15996	15995	5000	5/22/2014 13:45
15997	15996	5010	5/22/2014 13:45
15998	15997	5014	5/22/2014 13:45
15999	15998	5018	5/22/2014 13:45
16000	15999	5004	5/22/2014 14:00
16001	16000	5009	5/22/2014 14:00
16002	16001	5023	5/22/2014 14:00
16003	16002	5000	5/22/2014 14:00
16004	16003	5010	5/22/2014 14:00
16005	16004	5014	5/22/2014 14:00
16006	16005	5018	5/22/2014 14:00

Ending of third (final) page and last 18290th row:



	A	B	C
18775	18774	5017	11/6/2014 15:15
18776	18775	5021	11/6/2014 15:15
18777	18776	5013	11/6/2014 15:30
18778	18777	5017	11/6/2014 15:30
18779	18778	5021	11/6/2014 15:30
18780	18779	5013	11/6/2014 15:45
18781	18780	5017	11/6/2014 15:45
18782	18781	5021	11/6/2014 15:45
18783	18782	5013	11/6/2014 16:00
18784	18783	5017	11/6/2014 16:00
18785	18784	5021	11/6/2014 16:00
18786	18785	5013	11/6/2014 16:15
18787	18786	5017	11/6/2014 16:15
18788	18787	5021	11/6/2014 16:15
18789	18788	5013	11/6/2014 16:30
18790	18789	5017	11/6/2014 16:30
18791	18790	5021	11/6/2014 16:30
18792			
18793			

Windows batch files can be scheduled using Windows Task Scheduler. A systematic work flow can be developed to fully automate the process. For example:

1. Create a schedule to extract the first page of the Report. Name it SGI_Schedule1. Provide filter parameters and remote location. Configure it to run at 02:00 am
2. Create another schedule to extract the second page of the Report. Name it SGI_Schedule2. Provide filter parameters and remote location. Configure it to run at 02:02 am
3. Create another schedule to extract the third page of the Report. Name it SGI_Schedule3. Provide filter parameters and remote location. Configure it to run at 02:04 am
4. On the remote location (the Windows based PC on which FTP software is installed and configured) create a Windows Scheduled Task using Task Scheduler. Using Action Tab set the 'Merge_CSVs.bat' as the script to be run by the scheduler. Set the Trigger for 02:10 am.

In the morning when end user checks the folder location, he/she is able to get one single CSV file of the report with all rows available.

Note: If the result set is expected to contain more rows, additional Schedules can be configured for additional pages; 4^{th}, 5^{th} and so on.

FINAL NOTES, CAUTIONS AND LIMITATIONS

- When running the same report with page numbers, only the pageno parameter must be changed. Start/End dates and other used parameters must be kept as is every time for every page.
- The report template constructed using the method on this document should not contain grouping or summary lines in the report template.
- As explained previously, 8000 Row Limitation is not an actual 'Limitation' but a justified product architecture boundary to provide acceptable performance and user response time. Reports and excel exports exceeding this count is rarely required and for some special report types or cases only. The UCCE reporting engineer should understand if the necessity really exists first before considering this approach.
- When an end user queries the report page by page, there are no critical considerations on the usage. How many users can concurrently use the same type of report templates (heavy data quantity) depend on CUIC Sizing guidelines.
- When the reports are scheduled, provide enough time between Schedule Runs so that tasks will not impact each other's operability. Multiple Schedules running at the same time may trigger CUIC's throttle mechanism and produce errors.
- This study is completed for one report template for three Schedules. Higher number of schedules and how they operate (one by one or concurrently) should be closely tested before production deployment.
- The approach can bring as many pages as required, each page called one at a time.
- When merging the files, the windows batch file code sample uses the file name order for merging sequence. Since the file names are created using the name of the Schedules, attention is required when giving names to the Schedules.
- Windows batch code for merging files merges all CSV content in the given folder. Only CSV files to be merged must exist under that folder. Or sample code should be modified to address requirements.