LANDESK

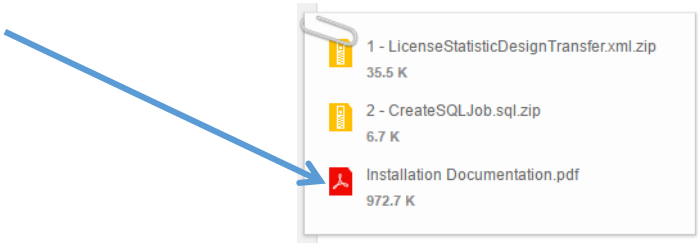
License Usage Statistics

Overview of Implementation for License Usage Statistics (Share-IT)

DOC-42852

This is not the Installation Document

If you were looking for the Document which explains how to install the License Usage Statistics, then please look at the – Installation Documentation.pdf – attached to DOC-42852



In this document we will discuss how the statistic tool is working on the insight.

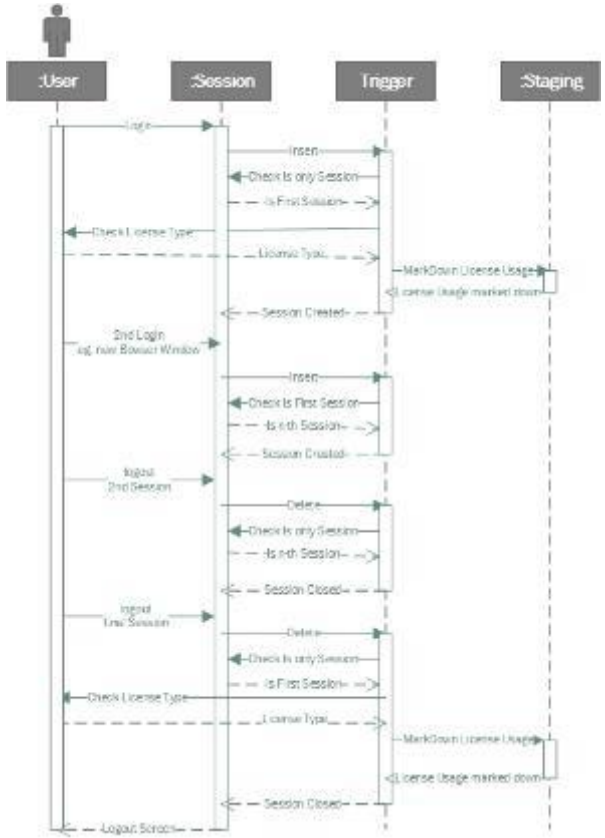
Therefore we will discuss the Sequence and key code snippets of the tool. After reading this document, you should be able to adjust the tool to your needs. At the end of this document we will see an POC license usage accounting example.

Prerequisites

To complete this Document you will need

1. DBA level of knowledge on how to query and update SQL data
2. Access to the SQL file attached to DOC-42852
    a. It would be much better to look at the code directly in SSMS as the code in the file is wrapped in Installation code for the trigger and SQL Server Agent Job. Readability of the code might be impacted.
3. Access at SA level to the MS SQL server you want to view/test the code in action.
4. A test system should you wish to alter the code and test the outcome.

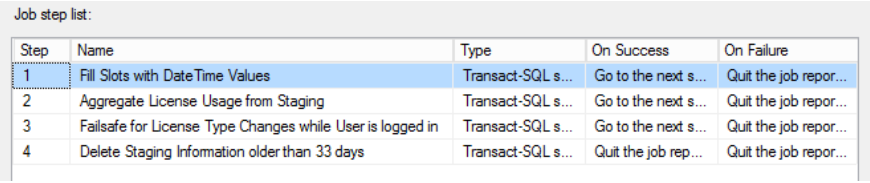
Sequence Overview (Trigger)



To the left you see the Sequence diagram of the statistic Tool.

- With every login the user writes an entry to the session object
- This starts the trigger
- The Trigger checks if this is the user's first session (license needed) or a consecutive Session.
- On the first Session the trigger gets the user's license Type and writes this information into the Staging Table
- Should the user close the session the entry from the session object will be deleted
- The trigger now checks if this was the user's final session
- On the closure of the final session the Trigger checks the user's Licence Type and writes the end of the usage into the Staging Table

Sequence Overview (SQL Job)


Job step list:

Step	Name	Type	On Success	On Failure
1	Fill Slots with DateTime Values	Transact-SQL s...	Go to the next s...	Quit the job repor...
2	Aggregate License Usage from Staging	Transact-SQL s...	Go to the next s...	Quit the job repor...
3	Failsafe for License Type Changes while User is logged in	Transact-SQL s...	Go to the next s...	Quit the job repor...
4	Delete Staging Information older than 33 days	Transact-SQL s...	Quit the job rep...	Quit the job repor...

- For the aggregation of the license usage into the time slots, the tools needs to know all the timeslots until the present moment. This has been done in the first Step
- Then the aggregation is processed
- Should a user change their license type whilst logged in, the tool will try to return this license type. Therefore we added a failsafe to close all License usage information should there be no user logged into the system
- The last step is to delete all data from the staging table which is older than 33 days (>1Month)

Key Code Trigger

One Trigger for Insert and Delete

```
CREATE TRIGGER Dbo.Trg_Stagelicinfo ON Dbo.Tps_User_Session
AFTER INSERT, DELETE
AS
   --Determine if this is an INSERT,UPDATE, or DELETE Action or a "failed
delete".
   DECLARE @action AS CHAR(1);
   SET @action = CASE
                 WHEN EXISTS(
                           SELECT *
                           FROM   Inserted)
                    AND EXISTS(
                              SELECT *
                              FROM   Deleted)
                 THEN 'U'  -- Set Action to Updated.
                 WHEN EXISTS(
                           SELECT *
                           FROM   Inserted)
                 THEN 'I'  -- Set Action to Insert.
                 WHEN EXISTS(
                           SELECT *
                           FROM   Deleted)
                 THEN 'D'  -- Set Action to Deleted.
                 ELSE NULL -- Skip. It may have been a "failed delete".
              END;
   IF @action = 'I' -- Session is opened
      BEGIN

      /* Do SomeThing */

      END;
   IF @action = 'D' -- Session is closed
      BEGIN

      /* Do SomeThing */

      END;
```

To compact the code and having only one Trigger the Trigger needs to know if is an Insert or Delete.

This is done by the case statement. By looking for the virtual tables for inserted and deleted data to exist, we can determine the type of DML which has been run, so that the rest of the code can be wrapped into two IF statements checking for the action to run the appropriate code.

First Session?

```
-- we only insert a new licnese count when
--- This user has allready a session open (user guid count <2 in tps_user_session)
--- There is no license with an end date at the current slot (concurrent License hand-over)
--- Exception: The license type is of type 'Named' as they are linked to the user
WITH First_User_Session_Cte -- filters inserted on duplicate session
    AS (SELECT *
        FROM   Inserted
        WHERE  [Tps_User_Guid] IN( -- lists only users which have only 1 session (or less) open
                            SELECT [T1].[Tps_User_Guid]
                            FROM   [Tps_User_Session] AS [T1]
                                JOIN [Inserted] AS [T2] ON [T1].[Tps_User_Guid] =
[T2].[Tps_User_Guid]
                            GROUP BY [T1].[Tps_User_Guid]
                            HAVING COUNT([T1].[Tps_User_Guid]) < 2))
    INSERT INTO @license_session_filtered
        SELECT [T1].[Tps_Guid],
               [Start_Slot] =
[Dbo].[Gettimeslotfromdatetime_Minutes]([T1].[Tps_Start_Datetime], DEFAULT),
               [Lictype] = CASE
                             WHEN [T3].[Tps_Name] != 'EndUser'
                             THEN COALESCE([T4].[Tps_Name], 'Concurrent')
                             ELSE 'Royalty-free'
                           END,
               [T2].[Tps_Guid]
        FROM   First_User_Session_Cte AS T1
               JOIN Tps_User AS T2 ON T1.Tps_User_Guid = T2.Tps_Guid
               JOIN Tps_User_Type AS T3 ON T2.Tps_User_Type_Guid = T3.Tps_Guid
               LEFT OUTER JOIN Tps_User_Licence_Type AS T4 ON
T2.Tps_User_Licence_Type_Guid = T4.Tps_Guid
        WHERE  [T3].[Tps_Name] != 'SystemUser';  -- Internal Users will not be acounted for
```

The license model of LANDesk works in such a way that each user's consecutive session does not require a new license. Therefore we need to filter these sessions out.

This is done by checking the session table for more than two entries for this user.

Only first session entries will be placed into @license_session_filtered so that we can be sure to work with first sessions only.

At this point we also verify the user's license type.

Last Session?

```
--- Deleting a Session
WITH First_User_Session_Cte -- filters inserted on duplicate session
    AS (SELECT *
        FROM   Deleted
        -- lists only users which have 0 sessions open
        WHERE  [Tps_User_Guid] NOT IN(
                                SELECT [T1].[Tps_User_Guid]
                                FROM   [Tps_User_Session] AS [T1]
                                    JOIN [Deleted] AS [T2] ON
[T1].[Tps_User_Guid] = [T2].[Tps_User_Guid]))
    INSERT INTO @returned_session_filtered
        SELECT [T1].[Tps_Guid],
            [End_Slot] =
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT),
            [Lictype] = CASE
                    WHEN [T3].[Tps_Name] != 'EndUser'
                    THEN COALESCE([T4].[Tps_Name], 'Concurrent')
                    ELSE 'Royalty-free'
                END,
            [T2].[Tps_Guid]
        FROM   First_User_Session_Cte AS T1
            JOIN Tps_User AS T2 ON T1.Tps_User_Guid = T2.Tps_Guid
            JOIN Tps_User_Type AS T3 ON T2.Tps_User_Type_Guid =
T3.Tps_Guid
            LEFT OUTER JOIN Tps_User_Licence_Type AS T4 ON
T2.Tps_User_Licence_Type_Guid = T4.Tps_Guid
        WHERE  [T3].[Tps_Name] != 'SystemUser';  -- Internal Users will
not be acounted for
```

To close the license usage information in the staging table, we need to know if the user closed their last session.

We do the same as with the detection of the first session, just that we now check that there are no entries of this user in the session table.

At this moment we also verify the license type of the user.

Named License Markdown (Session opened)

```
-- deleting Named and Royalty-free licenses
-- as they need to be checked against the user holding the license
DELETE @license_session_filtered
OUTPUT [Deleted].[Tps_Guid],
   [Deleted].[Tps_Start_Slot],
   [Deleted].[Lictype],
   [Deleted].[User_Guid]
   INTO @named_session_filtered
WHERE  [Lictype] != 'Concurrent';

-- the named licenses user could have closed their session in the same slot they return
-- so they should reuse their named license, to prevent double counting
DECLARE @named_lic_reused TABLE([Tps_Guid] UNIQUEIDENTIFIER);
-- we see if we can reuse licenses of these users
-- to know which license we could reuse we store them into a list
UPDATE Usr_Licensestatisticstaging
SET
   [Usr_Licenseusageenddatetime] = NULL
OUTPUT [Inserted].[Usr_Guid]
   INTO @named_lic_reused
WHERE  [Usr_Licensetype] IN('Named', 'Royalty-free')
AND [Usr_Licenseusageenddatetime] =
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT)
AND [Usr_User] IN(
            SELECT [User_Guid]
            FROM   @named_session_filtered);
-- we add the named licenses which could not reuse a returned license
INSERT INTO Usr_Licensestatisticstaging
([Usr_Guid],
[Usr_User],
[Usr_Licenseusagestartdatetime],
[Usr_Licensetype]
)
   SELECT [Tps_Guid],
      [User_Guid],
      [Tps_Start_Slot],
      [Lictype]
   FROM   @named_session_filtered
   WHERE  [Tps_Guid] NOT IN(
                  SELECT [Tps_Guid]
                  FROM   @named_lic_reused);
```

Named Licenses are linked to the user. Therefore we need to handle them differently than concurrent licences.

Therefore we split license types into a separate table variable to run further action only against named Licenses.

Should a user return during the current timeslot, we can re-use the old entry to avoid double counting of this license usage.

For the remaining named users who request a license we add an entry into the staging table for the current timeslot.

Concurrent License Markdown (Session opened)

```
-- Handling concurrent licenses
-- Either a concurrent license, is returned in the same slot, can be reused
-- or we need to add the difference to concurrent linceses in use
-- from which user we reuse the concurrent license is not important
DECLARE @concurrent_lic_needed_count INT= (
                                SELECT COUNT(*)
                                FROM   @license_session_filtered); -- number of
concurrent sessions needed
-- we (try) to update the amount of concurrent licenses which can be
reused
-- if they exists in the current slot
UPDATE TOP (@concurrent_lic_needed_count)
Usr_Licensestatisticstaging
SET
   [Usr_Licenseusageenddatetime] = NULL
WHERE                           [Usr_Licensetype] = 'Concurrent'
                            AND [Usr_Licenseusageenddatetime] =
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT);
SET @concurrent_lic_needed_count = @concurrent_lic_needed_count -
@@rowcount; -- should be zero if we had enough reusable licenses
-- should we still need licenses @concurrent_lic_needed_count > 0
-- than we insert them into the staging table
INSERT TOP (@concurrent_lic_needed_count)
INTO   Usr_Licensestatisticstaging
([Usr_Guid],
[Usr_Licenseusagestartdatetime],
[Usr_Licensetype],
[Usr_User]
)
   SELECT [Tps_Guid],
          [Tps_Start_Slot],
          [Lictype],
          [User_Guid]
   FROM   @license_session_filtered;
```

As concurrent licenses are not bound to a user we do not need to check if the user has already used a license in this timeslot. We only need to check for right amount of concurrent licenses;

therefore we re-use licences in the staging table, which have been return in the current timeslot.

We then markdown new license usage for those into the staging table for the remaining license amount needed.

Named License Markdown (Session closed)

```
UPDATE Usr_Licensestatisticstaging
SET
  [Usr_Licenseusageenddatetime] =
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT)
WHERE  [Usr_User] IN( -- returning for users only which have closed all
session
            SELECT [User_Guid]
            FROM   @returned_session_filtered
            WHERE  [Lictype] IN('Named', 'Royalty-free'))
AND [Usr_Licenseusageenddatetime] IS NULL;
```

To return a named license we need to find the rows with open license usage belonging to the user and set the license usage to ‘ended’ in the current timeslot.

Concurrent License Markdown (Session closed)

```
DECLARE @returned_concurrent_licences_count INT= (
                                SELECT COUNT(*)
                                FROM   @returned_session_filtered
                                WHERE  [Lictype] = 'Concurrent');
UPDATE TOP (@returned_concurrent_licences_count)
Usr_Licensestatisticstaging
SET
  [Usr_Licenseusageenddatetime] =
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT)
WHERE                           [Usr_Licenseusageenddatetime] IS NULL
                            AND [Usr_Licensetype] = 'Concurrent';
```

Returning a concurrent license is done by setting as many rows as needed, with current license usage, in the staging table to have an ended licenses usage.

There cannot be more licenses returned than previously taken; therefore we do not need to check for this circumstance.

Key Code SQL Job

Functions to get current TimeSlot

```
CREATE FUNCTION [Dbo].[Gettimeslotfromdatetime_Minutes]
(@datetimeinput     DATETIME = NULL, -- Defaults to current system time in
code
 @slotsizeinminutes INT      = 15
)  -- Default Slotsize 15 minutes
RETURNS DATETIME
AS
    BEGIN
        DECLARE @slot NVARCHAR(MAX);
        SELECT @datetimeinput = COALESCE(@datetimeinput,
GETUTCDATE());
        SELECT @slot = CONVERT( NVARCHAR(MAX),
CAST(CAST(@datetimeinput AS DATE) AS NVARCHAR(MAX))+'
'+CAST(DATEPART(HOUR, @datetimeinput) AS
NVARCHAR(MAX))+':'+CAST(ROUND((DATEPART(MINUTE, @datetimeinput)
/ @slotsizeinminutes), 0, 1) * @slotsizeinminutes AS
NVARCHAR(MAX))+':00', 120);
        RETURN CAST(@slot AS DATETIME);
    END;
```

To get the current slot (e.g. 15min slot) from a given DateTime, or of the current SystemTime, we use specialist functions.

This limits the risk of typos and improvs performance due to pre-compiled execution plans.

The function gets a DateTime attribute as a parameter. Should the DEFAULT have been used this is the current SystemTime.

Only the function for minutes takes a second argument, which sets the size of the slot. The Default here is 15min.

The DateTime value is the split up into its components and the slot has been calculated.

The function then returns the slot as DateTime attribute.

The other functions for hours, days, months work in the same way.

Getting all TimeSlots

```
WITH Cte_Fullrange
   AS
   (
   SELECT [Dbo].[Gettimeslotfromdatetime_Minutes](MAX([Usr_Timeslot]),
DEFAULT) AS [Slot]
   FROM    [Usr_Licensestatistic15min]
   UNION ALL
   SELECT DATEADD(MINUTE, 15, [Slot])
   FROM   Cte_Fullrange
   WHERE  DATEADD(MINUTE, 15, [Slot]) <= GETUTCDATE())
   -- adding slots from full range into Target which do not exist
   INSERT INTO Usr_Licensestatistic15min
   ([Usr_Timeslot],
    [Usr_Guid]
   )
        SELECT [Slot],
             NEWID()
        FROM   Cte_Fullrange AS T2
        WHERE  NOT EXISTS(
                     SELECT [Slot]
                     FROM   [Usr_Licensestatistic15min] AS [T1]
                     WHERE  [T1].[Usr_Timeslot] = [T2].[Slot])
        OPTION(MAXRECURSION 1000); -- we allow the recursion to run for
values up to 10 days in the past before stopping the loop
```

To get the full set of Timeslots, we use a recursive query using Common Table Expressions.

The Recursion starts with the last timeslot added to the statistic table.

Until the termination condition, of reaching the current slot has been reached, the function will call itself.

The calculated slots are then added to the statistic table.

This is done for all the statistic tables. The statements to do this differ only by the name of the function to get the current slot, to ensure correct data.

Calculating license usage from staging

```
WITH Licensetype_Cte -- transforms licnesetype into summable int
    AS (SELECT [Usr_Timeslot],
            [Concurrent] = CASE
                    WHEN [T1].[Usr_Licensetype] = 'Concurrent'
                    THEN 1
                    ELSE 0
                END,
            [Named] = CASE
                    WHEN [T1].[Usr_Licensetype] = 'Named'
                    THEN 1
                    ELSE 0
                END,
            [Rf] = CASE
                    WHEN [T1].[Usr_Licensetype] = 'Royalty-free'
                    THEN 1
                    ELSE 0
                END
        FROM   Usr_Licensestatisticstaging AS T1
            JOIN Usr_Licensestatistic15min AS T2 ON T2.Usr_Timeslot
BETWEEN T1.Usr_Licenseusagestartdatetime AND
COALESCE(T1.Usr_Licenseusageenddatetime,
[Dbo].[Gettimeslotfromdatetime_Minutes](DEFAULT, DEFAULT))),
    Licenseperslot_Cte
    AS (SELECT [Usr_Timeslot],
            SUM([Concurrent]) AS [Concurrent],
            SUM([Named]) AS     [Named],
            SUM([Rf]) AS        [Rf]
        FROM   Licensetype_Cte
        GROUP BY [Usr_Timeslot])
```

To calculate the license usage we convert the strings representing the license type into an integer via a case statement, so that we can sum up license usage into a representation of the license usage.

To show this information per timeslot we group this information over the timeslots.

Aggregating Concurrent and RF license usage

```
WITH Inner_Cte
    AS (SELECT [T2].[Usr_Timeslot],
            [Concurrent] = MAX([T1].[Usr_Numberofconcurrentlicenses]),
            [Rf] = MAX([T1].[Usr_Numberofenduserlicenses])
        FROM   Usr_Licensestatistic15min AS T1
            JOIN Usr_Licensestatistic1hour AS T2 ON
[Dbo].[Gettimeslotfromdatetime_Hours](T1.Usr_Timeslot) =
T2.Usr_Timeslot
        GROUP BY [T2].[Usr_Timeslot])
    UPDATE T1
     SET
        [Usr_Numberofconcurrentlicenses] = (CASE
                                    WHEN [T1].[Usr_Numberofconcurrentlicenses]
> [T2].[Concurrent]
                                    THEN [T1].[Usr_Numberofconcurrentlicenses]
                                    ELSE [T2].[Concurrent]
                                END),
        [Usr_Numberofenduserlicenses] = (CASE
                                    WHEN [T1].[Usr_Numberofenduserlicenses] >
[T2].[Rf]
                                    THEN [T1].[Usr_Numberofenduserlicenses]
                                    ELSE [T2].[Rf]
                                END)
    FROM [Usr_Licensestatistic1hour] [T1]
        JOIN [Inner_Cte] [T2] ON [T1].[Usr_Timeslot] = [T2].[Usr_Timeslot]
    WHERE  [T1].[Usr_Numberofconcurrentlicenses] < [T2].[Concurrent]
        OR [T1].[Usr_Numberofenduserlicenses] < [T2].[Rf];
```

As concurrent and royalty-free licenses are not user bound, we just need to get the maximum value of used licenses from the smaller timeslots over the bigger timeslots.

To limit writes to the database we check if the new values are bigger than the currently stored values and only write to DB if this is the case.

The same statement is used for the other aggregated timespans statistic tables.

Aggregating Named license usage

```
WITH Inner_Cte
   AS (SELECT [T2].[Usr_Timeslot],
           COUNT(*) AS [Named]
      FROM   Usr_Licensestatisticstaging AS T1
           JOIN Usr_Licensestatistic1hour AS T2 ON T2.Usr_Timeslot
BETWEEN T1.Usr_Licenseusagestartdatetime AND
T1.Usr_Licenseusageenddatetime
      WHERE  [T1].[Usr_Licensetype] = 'Named'
      GROUP BY [T2].[Usr_Timeslot])
   UPDATE T1
    SET
       [Usr_Numberofnamedlicenses] = [T2].[Named]
   FROM [Usr_Licensestatistic1hour] [T1]
      JOIN [Inner_Cte] [T2] ON [T1].[Usr_Timeslot] = [T2].[Usr_Timeslot]
   WHERE  [T2].[Named] > [T1].[Usr_Numberofnamedlicenses];
```

As users re-use their named license when they return, the aggregation of the named licenses can be simplified as the count of all named license entries within a time period.

The aggregation happens by joining over the bigger timeslots of the aggregation table.

Here we also limit the writes to the database, by only writing to the DB when the new values are bigger than the old ones.

POC License Accounting

Adding accounting data to accounting table (Trigger)

```
WITH Inner_Cte
 AS(SELECT                    [Tps_Guid] = NEWID(),
                              [T2].[User_Guid],
                              [T1].[Tps_Name],
                              [T1].[Tps_Primary_Group_Guid],
                              [Primarygroupname] = (
                                        SELECT [Tps_Name]
                                        FROM   [Tps_Group]
                                        WHERE  [Tps_Guid] =
[T1].[Tps_Primary_Group_Guid]),
                              [Tps_Start_Slot] =
[Dbo].[Gettimeslotfromdatetime_Day]([T2].[Tps_Start_Slot])
    FROM Tps_User AS T1
        JOIN @license_session_filtered AS T2 ON T1.Tps_Guid = T2.User_Guid)
 INSERT INTO Usr_Licenseaccounting1day
 ([Usr_Guid],
  [Usr_User],
  [Usr_Usernamepersitent],
  [Usr_Primarygroup],
  [Usr_Maingroupnamepersitent],
  [Usr_Timeslot]
 )
      SELECT [Tps_Guid],
           [User_Guid],
           [Tps_Name],
           [Tps_Primary_Group_Guid],
           [Primarygroupname],
           [Tps_Start_Slot]
      FROM   Inner_Cte
      WHERE  NOT EXISTS(
                   SELECT [Usr_Timeslot]
                   FROM   [Usr_Licenseaccounting1day] AS [A2]
                   WHERE  [A2].[Usr_Timeslot] = [Inner_Cte].[Tps_Start_Slot]
                        AND [A2].[Usr_User] = [Inner_Cte].[User_Guid]);
```

The business may wish to have some kind of accounting for used licenses. E.g. to let the business units pay for users using the system.

Whilst the element the accounting has been made against can differ from company to company this cannot be set in advance.

In the POC code available, the users primary group is used here fore. This way the cost could be assigned to the number of users of each group (cost centre).

The code checks if a user has already been accounted for in the current timeslot, from the list of users who start their first session.

If not, the user is added to the current slot.

To ensure that the information of the current assigned primary group stays unchanged, the name of the group along with the user becomes persistant into the database. This way the users can change their primary group without trouble to the accounting.

Adding timeslots to the accounting table (SQL Job)

```
INSERT INTO Usr_Licenseaccounting1day
([Usr_Guid],
 [Usr_Timeslot]
)
     SELECT NEWID(),
          [Usr_Timeslot]
     FROM   Usr_Licensestatistic1day
     WHERE  NOT EXISTS(
                    SELECT [Usr_Timeslot]
                    FROM   [Usr_Licenseaccounting1day] AS [T1]
                    WHERE  [T1].[Usr_Timeslot] =
[Usr_Licensestatistic1day].[Usr_Timeslot]);
```

The accounting statement in the trigger needs the timeslots to be known in the accounting table, to join the values correctly.

As these timeslots are already calculated and added to the license usage table of the same timespan, we can copy all timeslots from this table into the accounting table.

This concludes the Function documentation