

Perceptive Content Database IN_DB_UTIL Package 1.0 PostgreSQL

Reference Guide

Version: Foundation 23.1

Documentation Notice

Information in this document is subject to change without notice. The software described in this document is furnished only under a separate license agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in the license agreement. This document or accompanying materials may contain certain information which is confidential information of Hyland Software, Inc. and its affiliates, and which may be subject to the confidentiality provisions agreed to by you.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. or one of its affiliates.

Hyland, HXP, OnBase, Alfresco, Nuxeo, and product names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2024 Hyland Software, Inc. and its affiliates.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity's local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity's jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

Table of Contents

Documentation Notice	2
Overview	5
Benefits	5
Create the IN_DB_UTIL package	5
IN_DB_UTIL database objects	5
IN_DB_UTIL framework tables	7
IN_DB_UTIL_SESSION_ATTRIBUTES.....	7
IN_DB_UTIL_CALLSTACK.....	10
IN_DB_UTIL_LOG.....	11
IN_DB_UTIL framework table type	14
IN_DB_UTIL_TY_SESSION_ATTR.....	14
IN_DB_UTIL framework stored procedures and functions	15
IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE.....	15
IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE.....	15
IN_DB_UTIL_SP_CREATE_LOG_TABLE.....	16
IN_DB_UTIL_SP_SESSION_SETUP	17
IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES.....	20
IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES	22
IN_DB_UTIL_SP_CALLSTACK_EXTEND.....	23
IN_DB_UTIL_SP_CALLSTACK_TRIM	24
IN_DB_UTIL_SP_CALLSTACK_PRINT	25
IN_DB_UTIL_SP_CLEAR_IDENTIFIER	26
IN_DB_UTIL_SP_SESSIONS_DISPLAY.....	30
IN_DB_UTIL_SP_CHECK_DEPENDENCIES	31
IN_DB_UTIL_FN_CHECK_PARAMETER	32
IN_DB_UTIL_SP_CHECK_TIME	34
IN_DB_UTIL_SP_LOGGER.....	35
IN_DB_UTIL_FN_GET_UTIL_VERSION.....	36
IN_DB_UTIL schema metadata stored procedures and functions	37
IN_DB_UTIL_FN_FORMAT_SCHEMA_VERSION	37
IN_DB_UTIL_FN_GET_INOW_VERSION_BASE	38
IN_DB_UTIL_FN_GET_INOW_VERSION.....	38
IN_DB_UTIL_FN_SCHEMA_CHECK_VERSION	39
IN_DB_UTIL_FN_SCHEMA_GET_HIST	39
IN_DB_UTIL_FN_SCHEMA_GET_HIST_TAB	40

IN_DB_UTIL_SP_GET_DDL_SEQUENCE	40
IN_DB_UTIL_SP_GET_DDL_TABLE	42
IN_DB_UTIL_SP_GET_DDL_INDEX.....	43
IN_DB_UTIL_SP_GET_DDL_PK.....	44
IN_DB_UTIL_SP_GET_DDL_FK.....	46
IN_DB_UTIL_SP_DROP_INDEX.....	47
IN_DB_UTIL_SP_DROP_PK.....	49
IN_DB_UTIL_SP_DROP_FK.....	50
IN_DB_UTIL database metadata functions.....	52
IN_DB_UTIL_FN_GET_DB_INFO	52
IN_DB_UTIL_FN_GET_FK_INFO.....	53
IN_DB_UTIL_FN_DB_CONNECTIONS_COUNT.....	54
IN_DB_UTIL_FN_DB_CONNECTIONS_DISPLAY	55
IN_DB_UTIL_FN_DROP_UPGRADE_PROCS	55
IN_DB_UTIL metrics functions	56
IN_DB_UTIL_FN_GET_DURATION	56
IN_DB_UTIL_FN_GET_RPM.....	57
IN_DB_UTIL_FN_GET_USEC.....	58
IN_DB_UTIL database cleanup stored procedures	58
IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA.....	58
IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA.....	64
Troubleshooting commands.....	70
Cleanup after a disconnected or killed session	70
Debugging.....	71
Removing the IN_DB_UTIL package	71
Additional examples	73
Calculating and displaying throughput metrics (RPM)	73
Generating schema DDL.....	74

Overview

The following document provides a reference for usage details for version 1.0 of the Perceptive Content IN_DB_UTIL database package for PostgreSQL.

The IN_DB_UTIL database package is a collection of integrated procedures and functions that comprise a framework to facilitate interprocess communication between procedures as well as provide a set of scripts that are commonly used during database schema and maintenance operations.

The IN_DB_UTIL database package is leveraged by the Perceptive Content database IN_DB_UPGRADE package during database upgrades but can also be utilized independently to perform various maintenance tasks or leveraged as a framework for independent scripting initiatives.

Benefits

- Provides a framework to pass specific, common parameters between procedures and functions.
- Provides a framework to leverage MAX_MINUTES within scripts to define an end time for scripts that have incremental workloads.
- Provides a framework to help prevent against the accidental overlap of executing scripts when leveraging one of the defined identifiers.
- Provides a collection of procedures and functions for common database tasks.
- Provides a collection of procedures and functions to assist with calculating throughput metrics to report on the performance of operations within scripts.
- Provides a collection of procedures and functions to perform various database maintenance tasks such as deleting old audit data and archived workflow data or rebuilding indexes.

Create the IN_DB_UTIL package

This database package is currently supported on PostgreSQL versions 13 and up. Please review the Perceptive Content Technical Specifications documentation for your specific version of Perceptive Content to confirm which versions of PostgreSQL are supported.

Execute the following script to create the IN_DB_UTIL database package.

- Using psql, execute the following script as the inuser user.

```
PerceptiveContentDB_IN_DB_UTIL_PostgreSQL_Package_v1.0.sql
```

IN_DB_UTIL database objects

The following table contains a list of all the database objects that comprise the IN_DB_UTIL package and are created when the script above is executed.

Note that the two framework tables are not created during the execution of the script above but are created when the IN_DB_UTIL framework is first initialized.

Object Name	Object Type	Scope
IN_DB_UTIL_SESSION_ATTRIBUTES	Table	Framework
IN_DB_UTIL_CALLSTACK	Table	Framework

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

IN_DB_UTIL_LOG	Table	Framework
IN_DB_UTIL_TY_SESSION_ATTR	Table Type	Framework
IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE	Stored Procedure	Framework
IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE	Stored Procedure	Framework
IN_DB_UTIL_SP_CREATE_LOG_TABLE	Stored Procedure	Framework
IN_DB_UTIL_SP_SESSION_SETUP	Stored Procedure	Framework
IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES	Stored Procedure	Framework
IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES	Stored Procedure	Framework
IN_DB_UTIL_SP_CALLSTACK_EXTEND	Stored Procedure	Framework
IN_DB_UTIL_SP_CALLSTACK_TRIM	Stored Procedure	Framework
IN_DB_UTIL_SP_CALLSTACK_PRINT	Stored Procedure	Framework
IN_DB_UTIL_SP_CLEAR_IDENTIFIER	Stored Procedure	Framework
IN_DB_UTIL_SP_SESSIONS_DISPLAY	Stored Procedure	Framework
IN_DB_UTIL_SP_CHECK_DEPENDENCIES	Stored Procedure	Framework
IN_DB_UTIL_FN_CHECK_PARAMETER	Function	Framework
IN_DB_UTIL_SP_CHECK_TIME	Stored Procedure	Framework
IN_DB_UTIL_SP_LOGGER	Stored Procedure	Framework
IN_DB_UTIL_FN_FORMAT_SCHEMA_VERSION	Function	Framework
IN_DB_UTIL_FN_GET_UTIL_VERSION	Function	Framework Metadata
IN_DB_UTIL_FN_GET_INOW_VERSION	Function	Schema Metadata
IN_DB_UTIL_FN_GET_INOW_VERSION_BASE	Function	Schema Metadata
IN_DB_UTIL_FN_SCHEMA_CHECK_VERSION	Function	Schema Metadata
IN_DB_UTIL_FN_SCHEMA_GET_HIST	Function	Schema Metadata
IN_DB_UTIL_FN_SCHEMA_GET_HIST_TAB	Function	Schema Metadata
IN_DB_UTIL_SP_GET_DDL_SEQUENCE	Stored Procedure	Schema Metadata
IN_DB_UTIL_SP_GET_DDL_TABLE	Stored Procedure	Schema Metadata
IN_DB_UTIL_SP_GET_DDL_PK	Stored Procedure	Schema Metadata

IN_DB_UTIL_SP_GET_DDL_INDEX	Stored Procedure	Schema Metadata
IN_DB_UTIL_SP_GET_DDL_FK	Stored Procedure	Schema Metadata
IN_DB_UTIL_SP_DROP_INDEX	Stored Procedure	Schema Change
IN_DB_UTIL_SP_DROP_PK	Stored Procedure	Schema Change
IN_DB_UTIL_SP_DROP_FK	Stored Procedure	Schema Change
IN_DB_UTIL_FN_GET_FK_INFO	Function	Schema Metadata
IN_DB_UTIL_FN_GET_DB_INFO	Function	Database Information
IN_DB_UTIL_FN_DB_CONNECTIONS_COUNT	Function	Database Check
IN_DB_UTIL_FN_DB_CONNECTIONS_DISPLAY	Function	Database Check
IN_DB_UTIL_FN_DROP_UPGRADE_PROCS	Function	Upgrade
IN_DB_UTIL_FN_GET_DURATION	Function	Metrics
IN_DB_UTIL_FN_GET_RPM	Function	Metrics
IN_DB_UTIL_FN_GET_USEC	Function	Metrics
IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA	Stored Procedure	Maintenance
IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA	Stored Procedure	Maintenance

IN_DB_UTIL framework tables

The IN_DB_UTIL framework uses the following two tables to manage the attributes and call stack for the sessions that are utilizing the available identifiers of the framework. These tables are created when the IN_DB_UTIL framework is initialized for the first time.

IN_DB_UTIL_SESSION_ATTRIBUTES

This table contains the session parameters/attributes for each of the supported identifiers (IN_DB_UTIL, SYNC, UPGRADE) and is maintained by the various procedures and functions of the IN_DB_UTIL package. You should never need to manually update the rows in this table.

The [IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE](#) procedure is used to create this table.

You can use the following framework procedures to interface with this table:

- [IN_DB_UTIL_SP_SESSION_SETUP](#)
- [IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES](#)
- [IN_DB_UTIL_SP_SESSIONS_DISPLAY](#)
- [IN_DB_UTIL_SP_CLEAR_IDENTIFIER](#)

This table contains the following structure, which aligns with the columns defined in the [IN_DB_UTIL_TY_SESSION_ATTR](#) user-defined table type and the START_TIME and MAX_TIME columns.

```
CREATE TABLE inuser.IN_DB_UTIL_SESSION_ATTRIBUTES
(
  IDENTIFIER          citext_max_16  DEFAULT 'IN_DB_UTIL',
  STATUS              citext_max_9   DEFAULT 'ISNULL',
  PROCEDURE           citext_max_64  DEFAULT 'ISNULL',
  CALLER              citext_max_64,
  START_TIME          timestamp with time zone,
  MAX_TIME            timestamp with time zone,
  OBJECT              citext_max_64,
  SUBOBJECT           citext_max_64,
  TABLESPACE_DATA   citext_max_64,
  TABLESPACE_INDX   citext_max_64,
  MAX_MINUTES         integer,
  MAX_DOP              integer,
  FILLFACTOR          integer,
  BATCH_SIZE          integer,
  SCHEMA_TARGET       citext_max_40,
  SCHEMA_VERSION      citext_max_40,
  SCHEMA_BASE         citext_max_40,
  SCHEMA_TYPE         citext_max_2,
  DATATYPE            citext_max_16,
  IS_SYNC_UPGRADE     citext_max_6,
  PID                 integer DEFAULT 0,
  DEBUG               citext_max_6,
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_IDENTIFIER
    CHECK (IDENTIFIER IN ('IN_DB_UTIL', 'SYNC', 'UPGRADE')),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_STATUS
    CHECK (STATUS IN ('IDLE', 'ISNULL', 'EXECUTING')),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_MAX_MINUTES
    CHECK (MAX_MINUTES >= 0),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_MAX_DOP
    CHECK (MAX_DOP >= 0),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_FILLFACTOR
    CHECK (FILLFACTOR >= 0),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_BATCH_SIZE
    CHECK (BATCH_SIZE >= 0),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_SCHEMA_TYPE
    CHECK (SCHEMA_TYPE IN ('', 'u', 'a', 'au')),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_DATATYPE
    CHECK (DATATYPE IN ('citext_max_', 'varchar')),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_IS_SYNC_UPGRADE
    CHECK (IS_SYNC_UPGRADE IN ('YES', 'NO')),
  CONSTRAINT IN_DB_UTIL_SESSION_ATTRIBUTES_CHECK_DEBUG
    CHECK (DEBUG IN ('Y', 'YES', 'N', 'NO')),
  CONSTRAINT PK_IN_DB_UTIL_SESSION_ATTRIBUTES
    PRIMARY KEY (IDENTIFIER, PROCEDURE, PID)
);
```

The following table contains a description of each column for the IN_DB_UTIL_SESSION_ATTRIBUTES table.

Column Name	Description
IDENTIFIER	The name of the identifier to which the session is associated with. Used for defining the scope of the session. Possible options are IN_DB_UTIL, SYNC, and UPGRADE. The default is IN_DB_UTIL.

Column Name	Description
STATUS	The status of the session associated with the specified identifier. Possible options are IDLE, ISNULL, and EXECUTING. The default is ISNULL.
PROCEDURE	The name of the procedure that is executing under the specified identifier.
CALLER	The name of the procedure that called the procedure (CALLER) that is executing under the specified identifier.
START_TIME	The timestamp for when the session was initialized. The START_TIME is also updated to the clock_timestamp() anytime the IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES procedure is executed for the identifier.
MAX_TIME	For procedures that include the MAX_MINUTES parameter; if a value greater than zero was supplied for the MAX_MINUTES parameter during session initialization then this will contain the calculated MAX_TIME timestamp. This defines an end time for when new operations can be picked up for execution. If MAX_TIME has elapsed, then no new operations will be started, and the procedure will terminate cleanly without completing any remaining tasks that might have otherwise been executed had time not expired.
OBJECT	The primary or parent object name affected by the current step of the procedure. This is usually a table name.
SUBOBJECT	The name of the object affected by the current step of the procedure. This could be an index, or column, or constraint.
TABLESPACE_DATA	If applicable, the name of the data tablespace that contains the OBJECT affected by the current step of the procedure.
TABLESPACE_INDX	If applicable, the name of the index tablespace that contains the SUBOBJECT affected by the current step of the procedure.
MAX_MINUTES	The number of minutes used to define an end time that a procedure is allowed to start new operations. If MAX_MINUTES has elapsed and the current time is greater than the calculated end time (MAX_TIME), then no new operations will be started, and the procedure will terminate cleanly without completing any remaining tasks that might have otherwise been executed had time not expired. The default is 0 which is equal to unlimited.
MAX_DOP	This is not currently being used. The degree of parallelism for the session. Review the postgresql.conf file to view and set the various parameters related to parallel execution. The default is 0.
FILLFACTOR	The percentage to try and fill index pages. Used during the creation of indexes. This only currently applies to indexes although you can also specify the fill factor for tables on PostgreSQL. The default is 90 percent.

Column Name	Description
BATCH_SIZE	The batch size (number of rows) to populate work queues (cursors). Currently only used by IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA and IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA during IN_DB_UPGRADE SYNC operations. Could also be used for TOP N/LIMIT N constructs for SELECT, INSERT, UPDATE, DELETE, and MERGE operations to iterate over the data in smaller batches to keep transaction sizes smaller. The default is 10,000 rows.
SCHEMA_TARGET	The target schema version. This parameter is primarily used by the IN_DB_UPGRADE package during Perceptive Content database upgrades.
SCHEMA_VERSION	The current or starting schema version. This parameter contains the full schema version number including the type designator of 'u' for Unicode.
SCHEMA_BASE	The base version of the schema which only includes the version number without the schema type designator such as 'u' for Unicode. This is used for a simplified schema check during conditional execution based on the schema version.
SCHEMA_TYPE	The schema type designation such as 'u' for Unicode.
DATATYPE	This parameter can be used to dynamically specify the datatype for character-based columns when creating tables. The INOW database utilizes user-defined domains that use the citext extension type, which is a case insensitive extension of the text datatype. The domains are prefixed with "citext_max_" followed by max length allowed for the domain as is enforced with a check constraint. Example: citext_max_23
IS_SYNC_UPGRADE	This parameter is used by the IN_DB_UPGRADE package to determine if the upgrade path uses the synchronization framework (SYNC) for the upgrade. This is for upgrades with a starting schema version less than 7.5. Possible options are YES or NO. The default is NO.
PID	This is the PostgreSQL background process identifier for the current user process that is using the framework for the associated identifier. This is based on the value returned by pg_backend_pid() for the session that instantiated the framework using the IN_DB_UTIL_SP_SESSION_SETUP procedure.
DEBUG	Specifies whether to display additional logging for debugging purposes. Includes state changes, call stack, SQL statements and other details. Possible options are YES or NO. The default is NO.

IN_DB_UTIL_CALLSTACK

This table contains the session call stack for each of the supported identifiers (IN_DB_UTIL, SYNC, UPGRADE) and is maintained by the various procedures and functions of the IN_DB_UTIL framework. You should never need to manually update the rows in this table.

The [IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE](#) procedure is used to create this table.

The following framework procedures can be used to interface with this table:

- [IN_DB_UTIL_SP_CALLSTACK_PRINT](#)
- [IN_DB_UTIL_SP_CALLSTACK_EXTEND](#)
- [IN_DB_UTIL_SP_CALLSTACK_TRIM](#)

This table contains the following structure:

```
CREATE TABLE inuser.IN_DB_UTIL_CALLSTACK
(
  identifier citext_max_16 NOT NULL,
  depth      smallint      NOT NULL CHECK (depth > 0),
  procedure  citext_max_64 NOT NULL,
  pid        integer        NOT NULL,
  CONSTRAINT IN_DB_UTIL_CALLSTACK_CHECK_IDENTIFIER
    CHECK (identifier IN ('IN_DB_UTIL', 'SYNC', 'UPGRADE')),
  CONSTRAINT IN_DB_UTIL_CALLSTACK_UNIQUE_CALLER
    UNIQUE (identifier, procedure, pid),
  CONSTRAINT PK_IN_DB_UTIL_CALLSTACK
    PRIMARY KEY (identifier, depth, procedure, pid)
);
```

Column Name	Description
IDENTIFIER	The name of the identifier for which the call stack entry is associated with. Possible options are IN_DB_UTIL, SYNC, UPGRADE. The default is IN_DB_UTIL.
DEPTH	The position of the PROCEDURE in the call stack order for the IDENTIFIER.
PROCEDURE	The name of the procedure that is associated with the call stack entry for the IDENTIFIER.
PID	The PID of the session utilizing the IN_DB_UTIL framework.

IN_DB_UTIL_LOG

This table contains logging information for IN_DB_UTIL and IN_DB_UPGRADE.

The [IN_DB_UTIL_SP_CREATE_LOG_TABLE](#) procedure is used to create this table.

The [IN_DB_UTIL_SP_LOGGER](#) procedure is used to insert records into this table.

This table contains the following structure, which includes the columns defined in the [IN_DB_UTIL_TY_SESSION_ATTR](#) user-defined table type plus additional columns used to identify the source of the logged record.

```
CREATE TABLE inuser.IN_DB_UTIL_LOG
(
  LOG_ID          bigint GENERATED ALWAYS AS IDENTITY,
  LOG_TIME        timestamp with time zone,
  DB_NAME         citext_max_64,
  SESSION_USR    citext_max_64,
  CURRENT_USR     citext_max_64,
  PID             integer,
  TAG             citext_max_6,
  IDENTIFIER      citext_max_16,
  PROCEDURE       citext_max_64,
  CALLER         citext_max_64,
  START_TIME      timestamp with time zone,
  MAX_TIME        timestamp with time zone,
  OBJECT          citext_max_64,
```

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

```

SUBJECT          citext_max_64,
TABLESPACE_DATA citext_max_64,
TABLESPACE_INDX citext_max_64,
MAX_MINUTES      integer,
MAX_DOP          integer,
FILFACTOR        integer,
BATCH_SIZE       bigint,
SCHEMA_TARGET    citext_max_40,
SCHEMA_VERSION   citext_max_40,
SCHEMA_BASE      citext_max_40,
SCHEMA_TYPE      citext_max_2,
DATATYPE         citext_max_16,
IS_SYNC_UPGRADE citext_max_6,
DEBUG            citext_max_6,
INFO_MESSAGE     citext,
ERROR_MESSAGE    citext,
SQL_COMMAND      citext,
CONSTRAINT PK_DB_UTIL_LOG PRIMARY KEY (LOG_ID)
);

```

Column Name	Description
LOG_ID	Identity column used as the primary key for the table and indicates the order in which rows were inserted into the table.
LOG_TIME	The date and time the row was inserted into the table.
DB_NAME	The database name.
SESSION_USR	The server login that executed the IN_DB_UTIL_SP_LOGGER stored procedure. For IN_DB_UTIL and IN_DB_UPGRADE procedures this will always be "inuser".
CURRENT_USR	The database user that executed the IN_DB_UTIL_SP_LOGGER stored procedure. For IN_DB_UTIL and IN_DB_UPGRADE procedures this will always be "inuser".
PID	The value returned by pg_backend_pid() for the session that executed the IN_DB_UTIL_SP_LOGGER stored procedure.
TAG	The unique TAG generated and assigned to the session using the IN_DB_UTIL framework.
IDENTIFIER	The name of the identifier for which the session was associated with.
PROCEDURE	The name of the procedure associated with the log entry.
CALLER	The name of the procedure that called the PROCEDURE.
START_TIME	The timestamp for when the session was initialized.
MAX_TIME	For procedures that include the MAX_MINUTES parameter; if a value greater than zero was supplied for the MAX_MINUTES parameter during session initialization then this will contain the calculated MAX_TIME timestamp. This defines an end point for when new operations can be picked up for execution. If MAX_TIME has elapsed, then no new operations will be started, and the procedure will terminate cleanly.
OBJECT	The primary or parent object name affected by the current actions. This is usually a table name.

Column Name	Description
SUBOBJECT	The name of the object affected by the current actions. This could be an index, or column, or constraint.
TABLESPACE_DATA	If applicable, the name of the tablespace that contains the OBJECT affected by the current actions executing under the specified identifier.
TABLESPACE_INDX	If applicable, the name of the tablespace that contains the SUBOBJECT affected by the current actions executing under the specified identifier.
MAX_MINUTES	The number of minutes used to define the MAX_TIME that the procedure is allowed to start new operations. If MAX_MINUTES has elapsed and the current time is greater than the calculated end time, then no new operations will be started and the procedure will gracefully end without completing any remaining tasks that might have otherwise been executed had time not expired. The default is 0 which is equal to unlimited.
MAX_DOP	This is not currently being used. The degree of parallelism for the session. Review the postgresql.conf file to view and set the various parameters related to parallel execution. The default is 0.
FILLFACTOR	The percentage to try and fill index pages. Used during the creation of indexes. This only currently applies to indexes although you can also specify the fillfactor for tables in Postgres. The default fillfactor for indexes is 90 percent. The default fillfactor for tables is 100 percent.
BATCH_SIZE	The batch size (number of rows) to populate work queues (cursors). Currently only used by the IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA and IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA maintenance procedures and during IN_DB_UPGRADE SYNC operations. Could also be used for LIMIT N constructs for SELECT, INSERT, UPDATE, DELETE, and MERGE operations to iterate over the data in smaller batches to keep transaction sizes smaller. The default is 10,000 rows.
SCHEMA_TARGET	The target schema version. This parameter is primarily used by the IN_DB_UPGRADE package during Perceptive Content database upgrades.
SCHEMA_VERSION	The current or starting schema version. This parameter contains the full schema version number including the type designators such as 'u' for Unicode.
SCHEMA_BASE	The base version of the schema which only includes the version number without the schema type designators such as 'u' for Unicode. This is used for a simplified schema check during conditional execution based on the schema version.
SCHEMA_TYPE	The schema type designation such as 'u' for Unicode.
DATATYPE	This parameter can be used to dynamically specify the datatype for character-based columns when creating tables. The INOW database utilizes user-defined domains that use the citext extension type, which is a case insensitive extension of the text

Column Name	Description
	datatype. The domains are prefixed with "citext_max_" followed by max length allowed for the domain as is enforced with a check constraint. Example: citext_max_23
IS_SYNC_UPGRADE	This parameter is used by the IN_DB_UPGRADE package to determine if the upgrade path uses the synchronization framework (SYNC) for the upgrade. This is for upgrades with a starting schema version less than 7.5. Possible options are YES or NO. The default is NO.
DEBUG	The value of the DEBUG parameter during execution.
INFO_MESSAGE	The informational message passed into the IN_DB_UTIL_SP_LOGGER procedure to be logged.
ERROR_MESSAGE	The error message passed into the IN_DB_UTIL_SP_LOGGER procedure to be logged.
SQL_COMMAND	The SQL command passed into the IN_DB_UTIL_SP_LOGGER procedure to be logged.

IN_DB_UTIL framework table type

The IN_DB_UTIL framework utilizes a user-defined table type called IN_DB_UTIL_TY_SESSION_ATTR, which is used to declare a table type variable to store and pass the various parameters and their values between the stored procedures and functions within the IN_DB_UTIL framework.

IN_DB_UTIL_TY_SESSION_ATTR

The IN_DB_UTIL_TY_SESSION_ATTR user-defined table type contains the following columns.

Note that the column descriptions are the same as those described above for the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table.

```
CREATE TYPE inuser.IN_DB_UTIL_TY_SESSION_ATTR
AS
(
  identifier          citext_max_16,
  status              citext_max_9,
  procedure            citext_max_64,
  caller              citext_max_64,
  object              citext_max_64,
  subobject           citext_max_64,
  tablespace_data     citext_max_64,
  tablespace_indx     citext_max_64,
  max_minutes         integer,
  max_dop              integer,
  fillfactor          integer,
  batch_size          integer,
  schema_target       citext_max_40,
  schema_version      citext_max_40,
  schema_base         citext_max_40,
  schema_type         citext_max_2,
  datatype            citext_max_16,
  is_sync_upgrade     citext_max_6,
  pid                 integer,
  debug               citext_max_6
)
```

```
);
```

IN_DB_UTIL framework stored procedures and functions

The following stored procedures and functions are created as part of the IN_DB_UTIL package and provide various functionality to support the creation and management of the framework.

IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE

The IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE stored procedure is used to create the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table. This procedure is executed by several of the framework procedures including the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure and does not need to be manually executed.

Parameter	Description	
in debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the results of the table creation.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_create_session_attr_table(null, null);
```

```
CALL in_db_util_sp_create_session_attr_table(v_debug, v_call_results);
```

IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE

The IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE stored procedure is used to create the [IN_DB_UTIL_CALLSTACK](#) table. This procedure is executed by several of the framework procedures including the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure and does not need to be manually executed.

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6

Parameter	Description	
	Default value:	No
	Description:	Specifies whether to display the results of the table creation.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_create_callstack_table(null, null);
```

```
CALL in_db_util_sp_create_callstack_table(v_debug, v_call_results);
```

IN_DB_UTIL_SP_CREATE_LOG_TABLE

The IN_DB_UTIL_SP_CREATE_LOG_TABLE stored procedure is used to create the [IN_DB_UTIL_LOG](#) table. This procedure is executed by the [IN_DB_UTIL_SP_LOGGER](#) stored procedure which is used by many other stored procedures to log qualifying actions such as setting up a session ([IN_DB_UTIL_SP_SESSION_SETUP](#)) or tearing down a session ([IN_DB_UTIL_SP_CLEAR_IDENTIFIER](#)) or to log any errors that are encountered.

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the results of the table creation.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_create_log_table(null, null);
CALL in_db_util_sp_create_log_table(v_debug, v_call_results);
```

IN_DB_UTIL_SP_SESSION_SETUP

The IN_DB_UTIL_SP_SESSION_SETUP stored procedure should be called when initializing a session within the IN_DB_UTIL framework for the specified identifier(s), which include IN_DB_UTIL, SYNC, UPGRADE, or ALL (SYNC and UPGRADE).

The IN_DB_UTIL identifier allows for multiple sessions to be established and execute at the same time as long as the calling procedures are different. For example, it allows the [IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA](#) and [IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA](#) stored procedures to overlap when executed from different sessions. It will not allow the same stored procedure to run at the same time (overlap) from two different sessions. The SYNC, UPGRADE, ALL identifiers are not allowed to overlap with each other or with the IN_DB_UTIL identifier. This is to ensure that any UPGRADE or SYNC related steps are not impacted by maintenance related stored procedures that might be running under the IN_DB_UTIL identifier.

The IN_DB_UTIL framework uses the PostgreSQL advisory lock procedures to lock the various resources (stored procedures and framework identifiers) while they are being used. The [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure uses the pg_try_advisory_lock procedure, during session initialization, to lock the resources. These application resource locks are held until the procedure has completed and called the [IN_DB_UTIL_SP_CLEAR_IDENTIFIER](#) procedure to cleanup the session and release the resource locks using the pg_advisory_unlock procedure. If a procedure is cancelled prior to completion then these application locks will persist until the session is disconnected or until the [IN_DB_UTIL_SP_CLEAR_IDENTIFIER](#) procedure is executed within the same session.

The IN_DB_UTIL_SP_SESSION_SETUP procedure accepts the [IN_DB_UTIL_TY_SESSION_ATTR](#) user-defined table type parameter which is used to pass specific, commonly used parameter values between the procedures interacting with the framework.

The IN_DB_UTIL_SP_SESSION_SETUP procedure is responsible for the following framework initialization steps:

- Acquire the advisory resource locks for the stored procedure and identifiers using the pg_try_advisory_lock procedure.
- Populate any common framework attributes, that are not provided, with default values.
- Validate any parameters passed into the framework
 - [IN_DB_UTIL_FN_CHECK_PARAMETER](#)
- Get the current Perceptive Content schema version
 - [IN_DB_UTIL_FN_GET_INOW_VERSION\(\)](#)
- Get schema version base (without type), and type (u)
 - [IN_DB_UTIL_FN_GET_INOW_VERSION_BASE\(\)](#)
- Get formatted schema version base used for lexical ordering and comparison
 - [IN_DB_UTIL_FN_FORMAT_SCHEMA_VERSION\(\)](#)
- Check dependencies (existence of all IN_DB_UTIL stored procedures, functions, types)
 - [IN_DB_UTIL_SP_CHECK_DEPENDENCIES](#)
- Create the table used to store the session attributes

- [IN_DB_UTIL_SP_CREATE_SESSION_ATTR_TABLE](#)
- Create the table used to store the callstack
 - [IN_DB_UTIL_SP_CREATE_CALLSTACK_TABLE](#)
- Sets the parameters for the specified identifier(s)
 - [IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES](#)
- Extends the framework call stack
 - [IN_DB_UTIL_SP_CALLSTACK_EXTEND](#)
- Logs the session setup to the IN_DB_UTIL_LOG table
 - [IN_DB_UTIL_SP_LOGGER](#)

Parameter	Description	
in_session_parameters	Type:	Input
	Datatype:	in_db_util_ty_session_attr
	Default value:	Null
	Description:	User-defined table type variable used to pass the values of all the supported attributes for the IN_DB_UTIL framework. See the IN_DB_UTIL_TY_SESSION_ATTR definition above for more details.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Example

The following example demonstrates the typical initialization of a session within the IN_DB_UTIL framework.

For each procedure or script that will utilize the IN_DB_UTIL framework, you will want to declare a table variable that uses the [IN_DB_UTIL_TY_SESSION_ATTR](#) type as the definition:

```
DECLARE v_session_parameters in_db_util_ty_session_attr;
```

For each procedure or script that will utilize the IN_DB_UTIL framework, you will want to declare some or all of the following parameters to hold the individual parameters that are passed back and forth between procedures using the v_session_parameters table variable.

```
DECLARE
v_identifier      citext_max_16;
v_status         citext_max_9;
v_proc_name      citext_max_64;
v_caller        citext_max_64;
v_object         citext_max_64;
```

```

v_subobject          citext_max_64;
v_tablespace_data   citext_max_64;
v_tablespace_indx   citext_max_64;
v_max_minutes        integer;
v_max_dop            integer;
v_fillfactor         integer;
v_batch_size         integer;
v_schema_target      citext_max_40;
v_schema_version     citext_max_40;
v_schema_base        citext_max_40;
v_schema_type        citext_max_2;
v_datatype           citext_max_16;
v_is_sync_upgrade    citext_max_6;
v_pid                integer;
v_debug              citext_max_6;

```

Before loading the parameters into the **v_session_parameters** table variable you may want to pre-populate some of the parameters such as **v_proc_name** and possibly **v_caller** if the procedure is being called by another procedure already established in the call stack. You may want to also populate any other parameters with the desired values unless you want to take the default values.

Examples:

```

v_proc_name := 'in_db_util_sp_cleanup_audit_data';
v_identifier := 'in_db_util';
v_batch_size := 10000;
v_max_minutes := 60;

```

Once the parameters above are populated as needed, you can use them to insert into the **v_session_parameters** table variable to be passed to the IN_DB_UTIL_SP_SESSION_SETUP procedure to initialize the session within the IN_DB_UTIL framework.

All the parameters are listed below but you only need to populate the parameters that you will be passing to the IN_DB_UTIL_SP_SESSION_SETUP procedure.

```

-- Setup Standard Session Parameters

v_session_parameters.identifier := v_identifier;
v_session_parameters.procedure := v_proc_name;
v_session_parameters.caller     := v_caller;
v_session_parameters.object     := v_object;
v_session_parameters.subobject  := v_subobject;
v_session_parameters.tablespace_data := v_tablespace_data;
v_session_parameters.tablespace_indx := v_tablespace_indx;
v_session_parameters.max_minutes := v_max_minutes;
v_session_parameters.max_dop    := v_max_dop;
v_session_parameters.fillfactor := v_fillfactor;
v_session_parameters.batch_size := v_batch_size;
v_session_parameters.schema_target := v_schema_target;
v_session_parameters.schema_version := v_schema_version;
v_session_parameters.schema_base := v_schema_base;
v_session_parameters.schema_type := v_schema_type;
v_session_parameters.datatype := v_datatype;
v_session_parameters.is_sync_upgrade := v_is_sync_upgrade;
v_session_parameters.pid := v_pid;
v_session_parameters.debug := v_debug;

```

Once the parameter values have been inserted into the **v_session_parameters** table variable it can be passed into to the IN_DB_UTIL_SP_SESSION_SETUP procedure to initialize the session within the IN_DB_UTIL framework.

```

CALL in_db_util_sp_session_setup(v_session_parameters, v_call_setup);

```

Once the session has been initialized, the session parameters (attributes) will be inserted into the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table which can be referenced and selected from to finish populating any remaining parameters that are available and needed.

The IN_DB_UTIL_SESSION_ATTRIBUTES table will contain a record for each of the three possible identifiers so you must predicate on the appropriate identifier for the duration of your session.

Note The functional difference between predicating with IDENTIFIER = V_SCOPE versus IDENTIFIER = V_IDENTIFIER in the example query below. V_SCOPE is the specific identifier such as IN_DB_UTIL, SYNC, or UPGRADE. V_IDENTIFIER could be any of those three or ALL to indicate that the session identifies with both the SYNC and UPGRADE identifiers. This convention is used within the IN_DB_UPGRADE package when a procedure, such as the DOWNTIME_STEPS procedure, is initialized and intends to consume both the SYNC and UPGRADE identifiers to prevent any synchronization events from running at the same time. In most cases, V_SCOPE and V_IDENTIFIER will be the same.

Note The ALL identifier does not include the IN_DB_UTIL identifier.

The following example shows how to populate some of the session parameters using the values that are stored in the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table after the session has been initialized.

```
-- Retrieve the Standard/Default Parameters from IN_DB_UTIL_SESSION_ATTRIBUTES table

SELECT
    tablespace_data,
    tablespace_indx,
    max_minutes,
    max_dop,
    fillfactor,
    batch_size,
    schema_target,
    schema_version,
    schema_base,
    schema_type,
    datatype,
    is_sync_upgrade,
    debug
INTO
    v_tablespace_data,
    v_tablespace_indx,
    v_max_minutes,
    v_max_dop,
    v_fillfactor,
    v_batch_size,
    v_schema_target,
    v_schema_version,
    v_schema_base,
    v_schema_type,
    v_datatype,
    v_is_sync_upgrade,
    v_debug
FROM in_db_util_session_attributes
WHERE identifier = v_scope;
```

IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES

The IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES stored procedure is used to set the session attributes (parameters) of the IN_DB_UTIL framework during session initialization for the specified identifier(s).

It is called by the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure when initializing a session within the IN_DB_UTIL framework.

The IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES procedure should never need to be manually executed.

The IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES procedure is responsible for updating the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table for the specified identifier(s).

The MAX_TIME property is set by this procedure if not already defined by a previous procedure in the current call stack for the session. The MAX_TIME parameter is derived from the MAX_MINUTES session parameter if provided during session initialization.

The IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES stored procedure will set the following session attributes:

STATUS, PROCEDURE, CALLER, START_TIME, MAX_TIME, OBJECT, SUBOBJECT, TABLESPACE_DATA, TABLESPACE_IND \bar{X} , MAX_MINUTES, MAX_DOP, FILLFACTOR, BATCH_SIZE, SCHEMA_TARGET, SCHEMA_VERSION, SCHEMA_BASE, SCHEMA_TYPE, DATATYPE, IS_SYNC_UPGRADE, PID, DEBUG.

Parameter	Description	
in_session_parameters	Type:	Input
	Datatype:	IN_DB_UTIL_TY_SESSION_ATTR
	Default value:	Null
	Description:	User-defined table type variable used to pass the values of all the supported attributes for the IN_DB_UTIL framework. See the IN_DB_UTIL_TY_SESSION_ATTR definition for more details.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Example

The following example shows the V_SESSION_PARAMETERS table variable, containing all the session attributes (parameters), being passed into IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES procedure to update the session attributes in the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table.

The **out_return_val** parameter is being recorded into the **v_call_results** variable and if a value of false is returned, signifying an error, then the procedure should exit cleanly.

```
CALL in_db_util_sp_session_set_attributes(v_session_parameters, v_call_results);
```

IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES

The IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES stored procedure can be used for updating a subset of the IN_DB_UTIL framework session attributes (parameters) at any point after session initialization.

This procedure should only be executed after session initialization has occurred and if you want to update any session attributes in the IN_DB_UTIL_SESSION_ATTRIBUTES table for the specified identifier(s).

It is expected that other session attributes such as CALLER, CALLER_HIST, MAX_MINUTES, BATCH_SIZE, and any other attributes not being updated, have already been defined during session initialization by the [IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES](#) procedure.

The IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES procedure will update the following session attributes:

STATUS, START_TIME, MAX_TIME, OBJECT, SUBOBJECT, TABLESPACE_DATA, TABLESPACE_INDX, MAX_MINUTES, MAX_DOP, FILLFACTOR, BATCH_SIZE, DATATYPE, DEBUG

Parameter	Description	
in_session_parameters	Type:	Input
	Datatype:	IN_DB_UTIL_TY_SESSION_ATTR
	Default value:	Null
	Description:	User-defined table type variable used to pass the values of all the supported attributes for the IN_DB_UTIL framework. See the IN_DB_UTIL_TY_SESSION_ATTR definition for more details.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Example

The following example updates the OBJECT, SUBOBJECT, and TABLESPACE_INDX attributes (parameters) within the V_SESSION_PARAMETERS table variable and then passes it to the IN_DB_UTIL_SP_SESSION_UPDATE_ATTRIBUTES procedure to update the session attributes in the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table.

Once the following is completed, the subsequent execution of the [IN_DB_UTIL_SP_SESSIONS_DISPLAY](#) procedure would reflect the updates and show that the

associated session is working on the specified OBJECT (table), SUBOBJECT (index) and TABLESPACE_INDX. This facilitates a certain degree of visibility from any other session into the actions being preformed by the session consuming the specified identifier(s).

```
-- Update Session Attributes
v_session_parameters.object      := 'in_doc';
v_session_parameters.subobject   := 'doc_idx2';
v_session_parameters.tablespace_indx := 'INOW_Indx';

CALL in_db_util_sp_session_update_attributes(v_session_parameters, v_call_results);
```

IN_DB_UTIL_SP_CALLSTACK_EXTEND

The IN_DB_UTIL_SP_CALLSTACK_EXTEND stored procedure is responsible for incrementing the call stack depth for the specified identifier(s) when the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure is executed during session initialization.

The IN_DB_UTIL_SP_CALLSTACK_EXTEND procedure should never need to be manually executed.

The call stack for each identifier is stored in the [IN_DB_UTIL_CALLSTACK](#) table.

Parameter	Description	
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	Null
	Description:	Specifies the identifier for which to extend the call stack.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)
in_procedure	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	Specifies the name of the procedure that called the IN_DB_UTIL_SP_SESSION_SETUP procedure to initialize the session within the framework, which includes extending the call stack for the specified identifier(s).
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug info during execution.

	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Example

```
CALL in_db_util_sp_callstack_extend(v_identifer, v_caller, v_debug, v_call_results);
```

IN_DB_UTIL_SP_CALLSTACK_TRIM

The IN_DB_UTIL_SP_CALLSTACK_TRIM stored procedure is responsible for decrementing the call stack depth for the specified identifier(s) when the IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure is executed during session tear down.

The IN_DB_UTIL_SP_CALLSTACK_TRIM procedure should never need to be manually executed.

The call stack for each identifier is stored in the IN_DB_UTIL_CALLSTACK table.

Parameter	Description	
in_identifer	Type:	Input
	Datatype:	citext_max_16
	Default value:	Null
	Description:	Specifies the identifier for which to trim the call stack.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug info during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean

	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Example

```
CALL in_db_util_sp_callstack_trim(v_identifier, v_debug, v_call_results);
```

IN_DB_UTIL_SP_CALLSTACK_PRINT

The IN_DB_UTIL_SP_CALLSTACK_PRINT stored procedure is used to fetch and print the current call stack depth for the specified identifier(s).

The call stack details for each identifier are stored in the [IN_DB_UTIL_CALLSTACK](#) table.

Parameter	Description	
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	ALL
	Description:	Specifies the identifier for which to print the call stack.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug info during execution.
	Options:	Yes No

Example

```
CALL in_db_util_sp_callstack_print();
```

Example output

Example output when executing during the IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA procedure.

```
***** Call Stack Start *****
INFO: Identifier  Depth  PID      Procedure
INFO: -----
INFO: IN_DB_UTIL   1      13064    IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA
INFO:
***** Call Stack End *****
```

Example output at a specific point during the execution of the IN_DB_UPGRADE_SP_UPTIME_STEPS procedure.

Note The IN_DB_UPGRADE_SP_UPTIME_STEPS procedure uses the ALL identifier and uses both the SYNC and UPGRADE identifiers to prevent the execution of any SYNC and UPGRADE operations from any another session.

```

***** Call Stack Start *****
INFO:  Identifier    Depth    PID      Procedure
INFO:  -----
INFO:  SYNC            1        29424    IN_DB_UPGRADE_SP_UPTIME_STEPS
INFO:  SYNC            2        29424    IN_DB_UPGRADE_SP_CREATE_INDEXES_IN_DOC
INFO:  UPGRADE         1        29424    IN_DB_UPGRADE_SP_UPTIME_STEPS
INFO:  UPGRADE         2        29424    IN_DB_UPGRADE_SP_SCHEMA_UPGRADE_7403_7500
INFO:  UPGRADE         3        29424    IN_DB_UPGRADE_SP_CREATE_INDEXES_IN_DOC
INFO:
***** Call Stack End *****

```

IN_DB_UTIL_SP_CLEAR_IDENTIFIER

The IN_DB_UTIL_SP_CLEAR_IDENTIFIER stored procedure is responsible cleaning up the IN_DB_UTIL framework when a session is tearing down. This procedure should be executed when a procedure or session has finished its unit of work.

The IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure is responsible for the following framework de-initialization steps.

- Decrement the call stack for the specified identifier(s)
 - [IN_DB_UTIL_SP_CALLSTACK_TRIM](#)
- Clear the session attributes for the specified identifier(s)
 - [IN_DB_UTIL_SESSION_ATTRIBUTES](#)
- Release the application resource locks for the stored procedure and identifier(s) using the pg_advisory_unlock procedure.

The IN_DB_UTIL framework uses the PostgreSQL advisory lock procedures to lock the various resources (stored procedures and framework identifiers) while they are being used. The [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure uses the sp_getaplock procedure, during session initialization, to lock the resources. These application resource locks are held until the procedure has completed and successfully called the IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure to cleanup the session and release the advisory locks using the pg_advisory_unlock procedure. If a procedure is cancelled prior to completion then these application locks will persist until the session is disconnected or until the IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure is executed within the same session.

Under normal circumstances, the IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure should never need to be manually executed. However, it can be used to manually clear the call stack and session attribute tables and pg_advisory locks as needed due to an unhandled exception or after manually cancelling an operation before completion.

- If the **in_procedure** parameter is not provided then the entire call stack will be cleared and all attributes will be reset to IDLE for the specified identifier(s) and all advisory locks will be released for the current session. Note that advisory locks cannot be cleared for other sessions so if there are advisory locks still being held by another session then IN_DB_UTIL_SP_CLEAR_IDENTIFIER must be executed from that session or the session must be disconnected to release the advisory locks.

If the **in_procedure** parameter is provided, then the call stack will be decremented to reflect that the procedure has finished for the identifier(s).

Important Do not manually execute this procedure for an identifier if any operations are still in progress as it will clear the call stack and session attributes which will lead to undesired results and unhandled exceptions.

Parameter	Description	
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	Null
	Description:	Specifies the identifier for which to clear the attributes.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)
in_procedure	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	Specifies the procedure name that is clearing the identifier. The procedure name is used to clear advisory locks that were created during session initialization for the specified identifier.
in_pid	Type:	Input
	Datatype:	Integer
	Default value:	Null
	Description:	The PID of the session that is currently utilizing the identifier. The inclusion of the PID is only necessary when clearing the IN_DB_UTIL identifier and is not required for the SYNC, UPGRADE, or ALL identifiers.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug info during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean

	Default value:	Null
	Description:	Indicates whether the procedure was successful or not. True = Success False = Error

Examples

The following example demonstrates how this procedure is called from within a procedure after it has completed all its work.

```
CALL in_db_util_sp_clear_identifier(v_identifier, v_procedure, v_pid, v_debug, v_call_results);
```

The following example demonstrates how to use this procedure to manually clear the call stack and session attributes for a specific identifier, IN_DB_UTIL in this case, whose PID is 22456.

Important Do not manually execute this procedure if any operations are still in progress.

```
CALL in_db_util_sp_clear_identifier('in_db_util', 'in_db_util_sp_cleanup_audit_data', 22456, 'yes', null);
```

Example output

The following example demonstrates the clearing of “ALL” identifiers (SYNC and UPGRADE) after manually executing the IN_DB_UTIL_SP_CLEAR_IDENTIFIER procedure to clear the call stack and session attributes after having cancelled the IN_DB_UPGRADE_SP_UPTIME_STEPS procedure before it completed.

```
CALL in_db_util_sp_clear_identifier('all',null,null,'yes',null);
```

```
Procedure Name: in_db_util_sp_clear_identifier
```

```
***** DEBUG INFORMATION *****
```

```
***** Call Stack Start *****
```

```
INFO: Identifier   Depth  PID      Procedure
INFO: -----
INFO: SYNC          1      1736     IN_DB_UPGRADE_SP_UPTIME_STEPS
INFO: UPGRADE       1      1736     IN_DB_UPGRADE_SP_UPTIME_STEPS
INFO:
```

```
***** Call Stack End *****
```

```
INFO:
```

```
INFO: Current Activity for the SYNC Identifier:
```

```
INFO:
```

```
INFO: SYNC STATUS:           EXECUTING
INFO: SYNC PROCEDURE:       in_db_upgrade_sp_uptime_steps
INFO: SYNC CALLED BY:       NULL
INFO: SYNC START TIME:      2024-05-23 11:52:29.395421-05
INFO: SYNC MAX TIME:        2024-05-23 11:53:29.395421-05
INFO: SYNC MAX_MINUTES:     1
INFO: SYNC BATCH SIZE:      10000
INFO: SYNC BACKGROUND_PID:  1736
INFO:
```

```
INFO:
```

```
INFO:
```

```
INFO: Current Activity for the UPGRADE Identifier:
```

```
INFO:
```

```
INFO: UPGRADE STATUS:       EXECUTING
INFO: UPGRADE PROCEDURE:    in_db_upgrade_sp_uptime_steps
INFO: UPGRADE CALLED BY:    NULL
INFO: UPGRADE START TIME:   NULL
INFO: UPGRADE MAX TIME:     2024-05-23 11:53:29.395421-05
```

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

```
INFO: UPGRADE MAX_MINUTES:      1
INFO: UPGRADE OBJECT:           NULL
INFO: UPGRADE SUBOBJECT:        NULL
INFO: UPGRADE TABLESPACE_DATA: INOW_Data
INFO: UPGRADE TABLESPACE_INDX: INOW_Indx
INFO: UPGRADE MAX_DOP:          0
INFO: UPGRADE FILLFACTOR:       90
INFO: UPGRADE SCHEMA_TARGET:    7.9.0.0
INFO: UPGRADE SCHEMA_BASE:      7.1.3.3
INFO: UPGRADE SCHEMA_VERSION:   7.1.3.3u
INFO: UPGRADE IS_SYNC_UPGRADE:  YES
INFO: UPGRADE BACKGROUND_PID:   1736
INFO:
INFO: IN_DB_UTIL_SP_CLEAR_IDENTIFIER_4
INFO: Releasing all advisory locks for the session.
INFO: Successfully cleared the session attributes and callstack for the SYNC and
UPGRADE identifiers.
INFO: Advisory lock count for current session (pid 24644) is 0.
INFO:
INFO: The CALLSTACK is empty.

INFO:
INFO: Current Activity for the SYNC Identifier:
INFO:
INFO: SYNC STATUS:                IDLE
INFO: SYNC PROCEDURE:            ISNULL
INFO: SYNC CALLED BY:            NULL
INFO: SYNC START TIME:           NULL
INFO: SYNC MAX TIME:             NULL
INFO: SYNC MAX_MINUTES:          NULL
INFO: SYNC BATCH SIZE:           NULL
INFO: SYNC BACKGROUND_PID:       0
INFO:
INFO:
INFO: Current Activity for the UPGRADE Identifier:
INFO:
INFO: UPGRADE STATUS:             IDLE
INFO: UPGRADE PROCEDURE:         ISNULL
INFO: UPGRADE CALLED BY:         NULL
INFO: UPGRADE START TIME:        NULL
INFO: UPGRADE MAX TIME:          NULL
INFO: UPGRADE MAX_MINUTES:       NULL
INFO: UPGRADE OBJECT:           NULL
INFO: UPGRADE SUBOBJECT:         NULL
INFO: UPGRADE TABLESPACE_DATA:  NULL
INFO: UPGRADE TABLESPACE_INDX:  NULL
INFO: UPGRADE MAX_DOP:           NULL
INFO: UPGRADE FILLFACTOR:        NULL
INFO: UPGRADE SCHEMA_TARGET:     NULL
INFO: UPGRADE SCHEMA_BASE:       NULL
INFO: UPGRADE SCHEMA_VERSION:    NULL
INFO: UPGRADE IS_SYNC_UPGRADE:   NULL
INFO: UPGRADE BACKGROUND_PID:    0
INFO:
INFO:
***** DEBUG INFORMATION END *****
```

IN_DB_UTIL_SP_SESSIONS_DISPLAY

The IN_DB_UTIL_SP_SESSIONS_DISPLAY stored procedure is used to fetch and print the current session attributes for the specified identifier(s).

It can be used to display the current session details for each of the three supported identifiers, IN_DB_UTIL, SYNC, and UPGRADE.

The session attributes for each identifier is stored in the [IN_DB_UTIL_SESSION_ATTRIBUTES](#) table.

Parameter	Description	
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	ALL
	Description:	Specifies the identifier for which to display the session attributes.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the procedure name during execution.
	Options:	Yes No

Examples

```
CALL inuser.in_db_util_sp_sessions_display();
CALL inuser.in_db_util_sp_sessions_display('all');
CALL inuser.in_db_util_sp_sessions_display('sync');
CALL inuser.in_db_util_sp_sessions_display('upgrade');
CALL inuser.in_db_util_sp_sessions_display('in_db_util');
```

Example output

Example output from the IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA procedure.

```
INFO: Procedure Name: in_db_util_sp_sessions_display
INFO:
INFO: Current Database Time: 2024-05-23 13:20:17.844086-05
INFO:
INFO: Current Activity for the IN_DB_UTIL Identifier:
INFO:
INFO: IN_DB_UTIL STATUS: EXECUTING
```

```

INFO: IN_DB_UTIL PROCEDURE:      in_db_util_sp_cleanup_audit_data
INFO: IN_DB_UTIL START TIME:     2024-05-23 13:20:17.842037-05
INFO: IN_DB_UTIL MAX TIME:       2024-05-23 14:20:17.814756-05
INFO: IN_DB_UTIL MAX_MINUTES:    60
INFO: IN_DB_UTIL BATCH SIZE:     10000
INFO: IN_DB_UTIL OBJECT:         in_audit, in_audit_obj, in_audit_detail
INFO: IN_DB_UTIL SUBOBJECT:      NULL
INFO: IN_DB_UTIL TABLESPACE_DATA: INOW_Data
INFO: IN_DB_UTIL TABLESPACE_INDX: INOW_Indx
INFO: IN_DB_UTIL FILLFACTOR:     90
INFO: IN_DB_UTIL BACKGROUND_PID: 26860
    
```

Example output from a database schema upgrade using the IN_DB_UPGRADE_SP_UPTIME_STEPS procedure.

```

INFO: Current Activity for the UPGRADE Identifier:
INFO:
INFO: UPGRADE STATUS:             executing
INFO: UPGRADE PROCEDURE:          in_db_upgrade_sp_create_indexes_in_doc
INFO: UPGRADE CALLED BY:          in_db_upgrade_sp_uptime_steps
INFO: UPGRADE START TIME:         2024-05-23 13:29:09.232033-05
INFO: UPGRADE MAX TIME:           2024-05-23 14:29:08.171154-05
INFO: UPGRADE MAX_MINUTES:        60
INFO: UPGRADE OBJECT:            new_doc
INFO: UPGRADE SUBOBJECT:          new_doc_idx2
INFO: UPGRADE TABLESPACE_DATA:   INOW_Data
INFO: UPGRADE TABLESPACE_INDX:  INOW_Indx
INFO: UPGRADE FILLFACTOR:         90
INFO: UPGRADE SCHEMA_TARGET:      7.9.0.0
INFO: UPGRADE SCHEMA_BASE:        7.1.3.3
INFO: UPGRADE SCHEMA_VERSION:     7.1.3.3u
INFO: UPGRADE IS_SYNC_UPGRADE:    YES
INFO: UPGRADE BACKGROUND_PID:    26976
    
```

IN_DB_UTIL_SP_CHECK_DEPENDENCIES

The IN_DB_UTIL_SP_CHECK_DEPENDENCIES stored procedure contains the list of all the IN_DB_UTIL objects and is used to validate that each defined object exists. If there are any missing dependencies then it will return False and all the missing dependencies will be listed.

The IN_DB_UTIL_SP_CHECK_DEPENDENCIES procedure is called when the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure is executed during session initialization.

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the procedure name during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean

	Default value:	NULL
	Description:	Indicates whether the procedure was successful or not. True = No missing dependencies False = Missing dependencies were found

Examples

```
CALL in_db_util_sp_check_dependencies('yes', null);
```

```
CALL in_db_util_sp_check_dependencies(v_debug, v_call_results);
```

IN_DB_UTIL_FN_CHECK_PARAMETER

The IN_DB_UTIL_FN_CHECK_PARAMETER function can be used to validate the existence of various types of database objects as well as check the values of various other parameters that are often used within the IN_DB_UTIL framework.

- List of supported database objects that have an owner and require the format of OWNER.OBJECT_NAME (FQN)
 - CONSTRAINT
 - FUNCTION
 - INDEX
 - PROCEDURE
 - SEQUENCE
 - TABLE
 - TRIGGER
 - TYPE
- List of supported database objects that do not require an owner and use the format of OBJECT_NAME
 - TABLESPACE
 - SCHEMA
- List of supported IN_DB_UTIL framework parameters
 - BATCH_SIZE
 - DEBUG
 - EVENT
 - FILLFACTOR
 - IDENTIFIER
 - IS_SYNC_UPGRADE
 - KEEP_SRC_TABLE
 - MAX_DOP
 - MAX_MINUTES

Parameter	Description	
in_check_type	Type:	Input
	Datatype:	citext
	Default value:	Null
	Description:	Specifies the parameter type being checked.
	Options:	See the list of supported parameters above.
in_check_value	Type:	Input
	Datatype:	Citext
	Default value:	Null
	Description:	Specifies the value of the parameter type being checked.
	Options:	Certain parameters are restricted to specific values.
out_check_status	Type:	Output
	Datatype:	Smallint
	Default value:	Null
	Description:	Returned status containing the parameter check results. The returned status depends on the type of object that is checked.
	Options:	For database objects it returns the following: 0 = Object does not exist 1 = Object exists -1 = Ambiguous Object Name (more than one) For parameter checks that are not database objects then 0 = Success (Valid parameter value) -1 = Error (Invalid parameter value)
out_check_message	Type:	Output
	Datatype:	citext
	Default value:	NULL
	Description:	Returned message containing the parameter check results.

Examples

```
SELECT * FROM in_db_util_fn_check_parameter('constraint',
'inuser.fk_doc_instance_id');
```

```
DO $$
DECLARE
    v_check_type      citext;
    v_check_value     citext;
    v_check_status    smallint;
    v_check_message   citext;
BEGIN
    v_check_type := 'constraint';
    v_check_value := 'inuser.fk_doc_instance_id';

    SELECT * FROM in_db_util_fn_check_parameter(v_check_type, v_check_value::citext)
INTO v_check_status, v_check_message ;

    RAISE INFO 'check status = %', v_check_status;
    RAISE INFO 'check message = %', v_check_message;
END;
$$;

INFO:  check status = 1
INFO:  check message = The inuser.fk_doc_instance_id constraint exists.
DO
```

IN_DB_UTIL_SP_CHECK_TIME

The IN_DB_UTIL_SP_CHECK_TIME stored procedure is used to determine if the MAX_TIME session parameter has elapsed for the specified identifier.

This only applies if the MAX_MINUTES session parameter has been set to a value greater than zero when the [IN_DB_UTIL_SP_SESSION_SET_ATTRIBUTES](#) procedure is called by the [IN_DB_UTIL_SP_SESSION_SETUP](#) procedure when initializing a session within the IN_DB_UTIL framework.

Note This procedure, in conjunction with MAX_MINUTES, should not be used with downtime events as it could lead to incomplete operations.

Parameter	Description	
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	Null
	Description:	Specifies the identifier for which to check the MAX_TIME session attribute.
	Options:	IN_DB_UTIL SYNC UPGRADE
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the timestamps used when checking if MAX_TIME has elapsed.

	Options:	Yes = Displays the current and maximum timestamps. No = No additional information is displayed.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies if the MAX_TIME session parameter has elapsed or not. True = MAX_TIME has not yet elapsed or is undefined False = MAX_TIME has elapsed.

Examples

```
CALL in_db_util_sp_check_time('in_db_util', 'yes', null);

CALL in_db_util_sp_check_time(v_identifier, v_debug, v_check_time);

DO $$
DECLARE
    v_check_time boolean;
BEGIN
    CALL in_db_util_sp_check_time('ALL', 'YES', v_check_time);

    IF v_check_time = true THEN
        RAISE NOTICE 'Time has not expired so execution will continue.';
    ELSIF v_check_time = false THEN
        RAISE NOTICE 'Time has expired so execution will be terminated.';
    END IF;
END $$;
```

IN_DB_UTIL_SP_LOGGER

The IN_DB_UTIL_SP_LOGGER stored procedure is used to log specified actions to the [IN_DB_UTIL_LOG](#) table. This procedure is executed by many other stored procedures to log qualifying actions such as setting up a session (IN_DB_UTIL_SP_SESSION_SETUP) or tearing down a session (IN_DB_UTIL_SP_CLEAR_IDENTIFIER) or to log any errors that are encountered.

Parameter	Description	
in_procedure	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	Can be used to pass in a procedure name or some other application name to be associated with the logged record.
in_identifier	Type:	Input
	Datatype:	citext_max_16
	Default value:	Null

	Description:	Can be used to pass in a NULL or valid identifier. For valid identifiers other than NULL then additional session attributes associated with the identifier will be fetched and logged.
	Options:	NULL IN_DB_UTIL ALL UPGRADE SYNC
in_info_msg	Type:	Input
	Datatype:	Citext
	Default value:	NULL
	Description:	Can be used to pass in an informational message to be logged.
in_error_msg	Type:	Input
	Datatype:	Citext
	Default value:	Null
	Description:	Can be used to pass in an error number and/or error message to be logged.
in_sql_cmd	Type:	Input
	Datatype:	Citext
	Default value:	NULL
	Description:	Can be used to pass in a SQL statement or other command to be logged.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display available debugging information
	Options:	Yes No

Examples

```
-- To log a record to the IN_DB_UTIL_LOG table
CALL in_db_util_sp_logger(v_proc_name, v_identifier, v_info_msg, v_error_msg,
v_exec_sql, v_debug);

-- To view the contents of the IN_DB_UTIL_LOG table
SELECT * FROM IN_DB_UTIL_LOG ORDER BY LOG_ID;
```

IN_DB_UTIL_FN_GET_UTIL_VERSION

The IN_DB_UTIL_FN_GET_UTIL_VERSION function returns the current version number for the IN_DB_UTIL package.

Parameter	Description	
UTIL_VERSION	Type:	Return
	Datatype:	Citext
	Default value:	1.0x
	Description:	Returns the version number of the IN_DB_UTIL package.

Example

```
SELECT * FROM in_db_util_fn_get_util_version();
```

IN_DB_UTIL schema metadata stored procedures and functions

The following collection of stored procedures and functions are used to perform various database schema checks, generate schema DDL and facilitate schema modifications. These are commonly used by the IN_DB_UPGRADE package during database schema upgrades.

IN_DB_UTIL_FN_FORMAT_SCHEMA_VERSION

The IN_DB_UTIL_FN_FORMAT_SCHEMA_VERSION function returns a formatted version of the base schema version number that can be used for lexical ordering and comparison of the schema version numbers.

Parameter	Description	
in_schema_version	Type:	Input
	Datatype:	citext_max_40
	Default value:	Null
	Description:	Accepts the standard version number as stored in the IN_PRODUCT_MOD_HIST.PRODUCT_VERSION column.
v_version_format	Type:	Return
	Datatype:	citext_max_40
	Default value:	Null
	Description:	Returns the formatted base version of the schema number which can be used for lexical ordering and comparing with other formatted version numbers. Example of formatted results returned for version 7.1.5.2u is 0007.0001.0005.0002

Examples

```
SELECT in_db_util_fn_format_schema_version() AS CURRENT_VERSION_FMT;
```

```
SELECT in_db_util_fn_format_schema_version('7.3.0.2u') AS VERSION_FMT;
```

IN_DB_UTIL_FN_GET_INOW_VERSION_BASE

The IN_DB_UTIL_FN_GET_INOW_VERSION_BASE function returns the base version number from the IN_PRODUCT_MOD_HIST table or for the provided schema version number. The base version does not include the schema identifiers such as 'u' (Unicode).

Parameter	Description	
in_schema_version	Type:	Input
	Datatype:	citext_max_40
	Default value:	Null
	Description:	Accepts the standard version number as stored in the IN_PRODUCT_MOD_HIST.PRODUCT_VERSION column.
v_version_base	Type:	Return
	Datatype:	citext_max_40
	Default value:	Null
	Description:	Returns the base version of the schema number without the schema identifiers. Example of base version for 7.1.5.2u is 7.1.5.2.

Examples

```
SELECT in_db_util_fn_get_inow_version_base() AS CURRENT_VERSION_BASE;
SELECT in_db_util_fn_get_inow_version_base('7.2.0.3u') AS VERSION_BASE;
```

IN_DB_UTIL_FN_GET_INOW_VERSION

The IN_DB_UTIL_FN_GET_INOW_VERSION function returns the current version of the inuser schema from the IN_PRODUCT_MOD_HIST table. This function returns the entire schema version number along with the encoding type.

Note If you need to return the base schema version for conditional logic then use the IN_DB_UTIL_FN_GET_INOW_VERSION_BASE stored procedure instead.

Parameter	Description	
inow_version	Type:	Return
	Datatype:	citext_max_40
	Default value:	Null
	Description:	The full schema version as returned by the following query: select product_version from inuser.in_product_mod_hist order by mod_time desc limit 1;

Example

```
SELECT in_db_util_fn_get_inow_version() AS CURRENT_VERSION;
```

IN_DB_UTIL_FN_SCHEMA_CHECK_VERSION

The IN_DB_UTIL_FN_SCHEMA_CHECK_VERSION function will check if the current schema version matches the provided schema version. If the **in_schema_version** parameter matches the current schema version then the procedure will return a value of true, otherwise it will return a value of false.

Parameter	Description	
in_schema_version	Type:	Input
	Datatype:	citext_max_40
	Default value:	None
	Description:	The schema version number being compared against the current inuser schema version.
v_return_val	Type:	Return
	Datatype:	Boolean
	Default value:	Null
	Description:	True = The in_schema_version matches the current inuser schema version. False = The in_schema_version does not match the current inuser schema version.

Examples

```
SELECT in_db_util_fn_schema_check_version('7.1.3.3u')
```

```
SELECT in_db_util_fn_schema_check_version('7.1.3.3')
```

IN_DB_UTIL_FN_SCHEMA_GET_HIST

The IN_DB_UTIL_FN_SCHEMA_GET_HIST function will display the inuser schema history.

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the function name during execution.
	Options:	Yes No
v_return_val	Type:	Return

	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Example

```
SELECT * FROM in_db_util_fn_schema_get_hist();
```

IN_DB_UTIL_FN_SCHEMA_GET_HIST_TAB

The IN_DB_UTIL_FN_SCHEMA_GET_HIST_TAB function will return the contents (SETOF) of the in_product_mod_hist table, which contains the inuser schema history. It is equivalent to executing the following query: SELECT * FROM in_product_mod_hist ORDER BY mod_time;

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the function name during execution.
	Options:	Yes No

Example

```
SELECT * FROM in_db_util_fn_schema_get_hist();
```

IN_DB_UTIL_SP_GET_DDL_SEQUENCE

The IN_DB_UTIL_SP_GET_DDL_SEQUENCE stored procedure generates the DDL for the specified sequence. If no sequence name is specified, then all sequences in the inuser and inemuser schemas will be generated.

Parameter	Description	
in_sequence_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	NULL
	Description:	The name of the sequence for which to generate the DDL.
	Options:	Any valid sequence in the inuser or inemuser schema.
in_reset_seq	Type:	Input

	Datatype:	Boolean
	Default value:	True
	Description:	Specifies whether to reset or use the next value as the starting value for the generated DDL.
	Options:	True = Generate the sequence with START WITH value of 1. False = Preserve the current value of the sequence by using the value returned by the nextval() function. This does not consider cached values.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether or not to display debug information during execution.
	Options:	Yes No
out_ddl	Type:	Output
	Datatype:	Citext
	Default value:	Null
	Description:	Contains the generated DDL for the sequence(s).
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_get_ddl_sequence(null, null, null, null, null);

CALL in_db_util_sp_get_ddl_sequence('in_message_sequence', null, null, null, null);

DO $$
DECLARE
    v_sequence_name citext_max_64 := 'in_message_sequence';
    v_reset_seq     boolean      := true;
    v_debug         citext_max_6  := 'no';
    v_sequence_ddl citext;
    v_call_results  boolean;
BEGIN
```

```
CALL in_db_util_sp_get_ddl_sequence(v_sequence_name, v_reset_seq, v_debug,
v_sequence_ddl, v_call_results);
RAISE INFO '%', v_sequence_ddl;
RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_GET_DDL_TABLE

The IN_DB_UTIL_SP_GET_DDL_TABLE stored procedure generates the DDL for the specified table. If no table name is specified, then all tables in the inuser and inemuser schemas will be generated.

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to generate the DDL.
	Options:	Any valid table in the inuser or inemuser schema.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether or not to display debug information during execution.
	Options:	Yes No
out_ddl	Type:	Output
	Datatype:	Citext
	Default value:	Null
	Description:	Contains the generated DDL for the table(s).
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_get_ddl_table(null, null, null, null);
```

```
CALL in_db_util_sp_get_ddl_table('in_doc', null, null, null);

DO $$
DECLARE
    v_table_name      citext_max_64 := 'in_doc';
    v_debug           citext_max_6  := 'no';
    v_table_ddl       citext;
    v_call_results    boolean;
BEGIN
    CALL in_db_util_sp_get_ddl_table(v_table_name, v_debug, v_table_ddl,
v_call_results);
    RAISE INFO '%', v_table_ddl;
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_GET_DDL_INDEX

The IN_DB_UTIL_SP_GET_DDL_INDEX stored procedure generates and displays the index DDL for the specified index or table. If a table name and index name is not specified, then all indexes for all tables in the inuser and inemuser schemas will be generated.

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	Specifies the table name for which to generate the index DDL.
	Options:	Any valid table in the inuser or inemuser schema.
in_index_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	Specifies the index name for which to generate the DDL. If the index name is not specified, then all indexes for the specified table will be generated and displayed.
	Options:	Any valid index for the specified table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display available debug information during execution.
	Options:	Yes No

out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_get_ddl_index('in_doc', null, null, null, null);

CALL in_db_util_sp_get_ddl_index(null, null, null, null, null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_doc';
    v_index_name    citext_max_64 := null;
    v_debug         citext_max_6  := 'no';
    v_index_ddl     citext;
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_get_ddl_index(v_table_name, v_index_name, v_debug, v_index_ddl,
    v_call_results);
    RAISE INFO '%', v_index_ddl;
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_GET_DDL_PK

The IN_DB_UTIL_SP_GET_DDL_PK stored procedure generates the primary key DDL for the specified table name or primary key name.

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to generate the primary key DDL.
	Options:	Any valid table in the inuser or inemuser schema.
in_pk_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null

	Description:	The name of the primary key constraint for which to generate the DDL.
	Options:	Any valid primary key in the inuser or inemuser schema.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
out_ddl	Type:	Output
	Datatype:	Citext
	Default value:	Null
	Description:	Contains the generated DDL.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_get_ddl_pk('in_doc', null, null, null, null);

CALL in_db_util_sp_get_ddl_pk(null, null, null, null, null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_doc';
    v_pk_name       citext_max_64 := null;
    v_debug         citext_max_6  := 'no';
    v_pk_ddl        citext;
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_get_ddl_pk(v_table_name, v_pk_name, v_debug, v_pk_ddl,
v_call_results);
    RAISE INFO '%', v_pk_ddl;
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_GET_DDL_FK

The IN_DB_UTIL_SP_GET_DDL_FK stored procedure generates the foreign key DDL for the specified foreign key name or all the foreign keys for the specified table depending on the specified scope.

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The table name for which to generate the foreign key DDL.
	Options:	Any valid table in the inuser schema.
in_fk_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The foreign key name for which to generate the DDL. If the FK_NAME is not specified, then all foreign keys on the specified table is displayed (SCOPE=ALL).
	Options:	Any valid foreign key for the specified table.
in_scope	Type:	Input
	Datatype:	citext_max_6
	Default value:	ALL
	Description:	The scope of the foreign keys, in relation to the specified table.
	Options:	ALL = All foreign keys TO and FROM the specified table. TO = Foreign keys that reference the specified table. FROM = Foreign keys on the specified table that reference other tables.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
out_ddl	Type:	Output

	Datatype:	Citext
	Default value:	Null
	Description:	Contains the generated DDL.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_get_ddl_fk(null, 'fk_dk_doc_id', null, null, null, null);
CALL in_db_util_sp_get_ddl_fk('in_doc', null, null, null, null, null);
CALL in_db_util_sp_get_ddl_fk('in_doc', null, 'all', null, null, null);
CALL in_db_util_sp_get_ddl_fk('in_doc', null, 'from', null, null, null);
CALL in_db_util_sp_get_ddl_fk('in_doc', null, 'to', null, null, null);
CALL in_db_util_sp_get_ddl_fk(null, null, null, null, null, null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_doc';
    v_fk_name       citext_max_64 := 'fk_d_doc_type_id';
    v_scope         citext_max_6  := null;
    v_debug         citext_max_6  := 'no';
    v_fk_ddl        citext;
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_get_ddl_fk(v_table_name, v_fk_name, v_scope, v_debug, v_fk_ddl,
v_call_results);
    RAISE INFO '%', v_fk_ddl;
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_DROP_INDEX

The IN_DB_UTIL_SP_DROP_INDEX stored procedure is used to drop indexes on the specified table. This procedure is intended to be used by the IN_DB_UPGRADE package for schema upgrade actions.

This procedure cannot be used to drop a primary key constraint.

Parameter	Description	
in_table_name	Type:	Input

Parameter	Description	
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to drop its indexes. If no index name is specified, then all indexes will be dropped for the specified table.
in_index_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the index to drop. If no index name is specified, then all indexes will be dropped for the specified table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
CALL in_db_util_sp_drop_index('in_alarm', null, null, null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_alarm';
    v_index_name    citext_max_64 := null;
    v_debug         citext_max_6  := 'no';
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_drop_index(v_table_name, v_index_name, v_debug, v_call_results);
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_DROP_PK

The IN_DB_UTIL_SP_DROP_PK stored procedure is used to drop the Primary Key constraint on the specified table. This procedure is intended to be used by the IN_DB_UPGRADE package for schema upgrade actions.

All tables in the inuser schema have a primary key constraint.

Any referencing foreign key constraints must be dropped prior to being able to drop a primary key constraint.

You may specify true for the **in_cascade** parameter to also drop any foreign key constraints that are currently referencing the primary key constraint that is being dropped.

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to drop the primary key constraint.
in_cascade	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to drop any existing foreign key constraints that are currently referencing the primary key that is to be dropped.
	Options:	True = Drop all referencing foreign key constraints. False = Do not drop any referencing foreign key constraints. If there are any, then dropping the primary key constraint will fail.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null

	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error
--	--------------	--

Examples

```
CALL in_db_util_sp_drop_pk('in_alarm', true, 'no', null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_alarm';
    v_cascade       boolean       := false;
    v_debug         citext_max_6  := 'no';
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_drop_pk(v_table_name, v_debug, v_call_results);
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL_SP_DROP_FK

The IN_DB_UTIL_SP_DROP_FK stored procedure is used to drop foreign key constraints on the specified table. This procedure is intended to be used by the IN_DB_UPGRADE package for schema upgrade actions.

You must provide the table name and either the foreign key name to be dropped OR indicate the SCOPE of constraints to be dropped for the table. [ALL|FROM|TO]

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to drop the foreign keys from. The in_fk_name or the in_scope parameter must also be provided.
in_fk_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the foreign key to drop. If no foreign key name is specified, then the scope must be specified.
in_scope	Type:	Input
	Datatype:	citext_max_6
	Default value:	Null

Parameter	Description	
	Description:	The scope of the foreign keys, in relation to the specified table.
	Options:	TO = Drops all foreign keys that reference the specified table. FROM = Drops all foreign keys on the specified table that reference other tables. ALL = Drops all foreign keys TO and FROM the specified table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
-- Drop all foreign keys that reference the IN_AUDIT table
CALL in_db_util_sp_drop_fk('in_audit', null, 'to', 'no', null);

-- Drop the FK_AD_A_ID foreign key from the IN_AUDIT_DETAIL table
CALL in_db_util_sp_drop_fk('in_audit_detail', 'fk_ad_a_id', null, 'no', null);

DO $$
DECLARE
    v_table_name    citext_max_64 := 'in_audit_detail';
    v_fk_name       citext_max_64 := 'fk_ad_a_id';
    v_scope         citext_max_6  := null;
    v_debug         citext_max_6  := 'no';
    v_call_results  boolean;
BEGIN
    CALL in_db_util_sp_drop_fk(v_table_name, v_fk_name, v_scope, v_debug,
v_call_results);
    RAISE INFO '%', v_call_results;
END;
$$;
```

IN_DB_UTIL database metadata functions

The following collection of functions are used to gather database information and perform various database level checks.

IN_DB_UTIL_FN_GET_DB_INFO

The IN_DB_UTIL_FN_GET_DB_INFO function will display configuration details and information about the database. There are three modes that provide varying degrees of information ranging from high level information (INFO) about the database as well as basic (BASIC) information to help with troubleshooting performance, and detailed (DETAILED) information including table and index sizes, statistics and current settings for the various database parameters.

Parameter	Description	
in_mode	Type:	Input
	Datatype:	citext_max_9
	Default value:	BASIC
	Description:	The mode determines the level of details and information to collect and display.
	Options:	INFO = Display minimal information about the database. BASIC = Displays various details about the database to help with troubleshooting performance. DETAILED = Includes configuration details about the database including table and index statistics and parameter values.
v_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```

SELECT inuser.in_db_util_fn_get_db_info();
SELECT inuser.in_db_util_fn_get_db_info('INFO');
SELECT inuser.in_db_util_fn_get_db_info('BASIC');
SELECT inuser.in_db_util_fn_get_db_info('DETAILED');

DO $$
BEGIN
    PERFORM inuser.in_db_util_fn_get_db_info('DETAILED');
END;
$$;
```

IN_DB_UTIL_FN_GET_FK_INFO

The IN_DB_UTIL_FN_GET_FK_INFO function lists the foreign key constraints that currently exist for the specified table. It also displays the current properties of the foreign keys, indicating if it has been validated and if it has been deferred or is deferrable.

By default, the function will display the status of the foreign keys to and from the specified table (SCOPE = 'ALL').

Parameter	Description	
in_table_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the table for which to get the foreign key information.
	Options:	Any valid table in the inuser schema.
in_fk_name	Type:	Input
	Datatype:	citext_max_64
	Default value:	Null
	Description:	The name of the foreign key constraint for which to get the information to display. If the foreign key name is not specified, then all foreign keys for the specified table will be generated and displayed (SCOPE = ALL).
	Options:	Any valid foreign key for the specified table.
in_scope	Type:	Input
	Datatype:	citext_max_6
	Default value:	ALL
	Description:	The scope of the foreign keys, in relation to the specified table.
	Options:	TO = Foreign keys that reference the specified table. FROM = Foreign keys on the specified table that reference other tables. ALL = All foreign keys TO and FROM the specified table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display debug information during execution.

Parameter	Description	
	Options:	Yes No
v_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Examples

```
SELECT inuser.in_db_util_fn_get_fk_info(null, null, null, null);

SELECT inuser.in_db_util_fn_get_fk_info('in_doc', null, null, null);

SELECT inuser.in_db_util_fn_get_fk_info(null, 'fk_dk_doc_id', null, null);
```

Example output

```
INFO:
INFO: Referential Integrity (FK) Constraint Report
INFO:
INFO: -----
INFO: Parent Table Name   FK Name           Child Table Name  FK Name           Validated  Deferred  Deferrable Update Rule  Delete Rule  Match Option
INFO: -----
INFO: in_doc               pk_doc            in_doc_kw         fk_dk_doc_id      true       false     false                    no action   no action   simple
INFO: in_doc               pk_doc            in_rc_record_doc  fk_rcrd_d_id      true       false     false                    no action   no action   simple
INFO: in_doc               pk_doc            in_sig            fk_s_doc_id       true       false     false                    no action   no action   simple
INFO: in_doc               pk_doc            in_task           fk_t_doc_id       true       false     false                    no action   no action   simple
INFO: in_doc               pk_doc            in_version        fk_v_doc_id       true       false     false                    no action   no action   simple
INFO: in_doc_type          pk_doc_type       in_doc            fk_d_doc_type_id  true       false     false                    no action   no action   simple
INFO: in_drawer            pk_drawer         in_doc            fk_doc_drawer_id  true       false     false                    no action   no action   simple
INFO: in_instance          pk_instance       in_doc            fk_doc_instance_id true        false     false                    no action   no action   simple
INFO:
INFO: 8 Foreign Keys were found.
INFO:
```

IN_DB_UTIL_FN_DB_CONNECTIONS_COUNT

The IN_DB_UTIL_FN_DB_CONNECTIONS_COUNT function will return the total number and active number of connections to the database. The returned counts exclude the session running the function.

Parameter	Description	
total_connections	Type:	Output
	Datatype:	Integer
	Default value:	Null
	Description:	Returns the total number of connections to the database.
active_connections	Type:	Output
	Datatype:	Integer
	Default value:	Null
	Description:	Returns the total number of active connections to the database.

Examples

```
SELECT * FROM in_db_util_fn_db_connections_count();

DO $$
DECLARE
    v_total integer;
    v_active integer;
BEGIN
    SELECT INTO v_total, v_active * FROM in_db_util_fn_db_connections_count();
    RAISE NOTICE '% active connections out of % total database connections', v_active,
v_total;
END;
$$;
```

IN_DB_UTIL_FN_DB_CONNECTIONS_DISPLAY

The IN_DB_UTIL_SP_DB_CONNECTIONS_DISPLAY stored procedure will display a summary of the user connections to the database.

Parameter	Description	
v_return_val	Type:	Return
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the function was successful or not. True = Success False = Error

Examples

```
SELECT in_db_util_fn_db_connections_display();

DO $$
BEGIN
    PERFORM in_db_util_fn_db_connections_display();
END;
$$;
```

IN_DB_UTIL_FN_DROP_UPGRADE_PROCS

The IN_DB_UTIL_FN_DROP_UPGRADE_PROCS function will drop all the objects that are created as part of the IN_DB_UPGRADE package. This can be executed after the inuser database schema has been upgraded to the target schema version and you want to remove all the extra functions and procedures associated with the IN_DB_UPGRADE package.

Parameter	Description	
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No

Parameter	Description	
	Description:	Specifies whether to display debug information during execution.
	Options:	Yes No
v_return_val	Type:	Return
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the function was successful or not. True = Success False = Error

Examples

```
SELECT inuser.in_db_util_fn_drop_upgrade_procs();
SELECT inuser.in_db_util_fn_drop_upgrade_procs('yes');
```

IN_DB_UTIL metrics functions

The following functions can be used to display throughput metrics of various scripted operations for logging and informational purposes.

IN_DB_UTIL_FN_GET_DURATION

The IN_DB_UTIL_FN_GET_DURATION function will calculate and display the elapsed duration of time based on the provided number of microseconds (usec).

Parameter	Description	
in_elapsed_usec	Type:	Input
	Datatype:	Numeric
	Default value:	Null
	Description:	Number of microseconds (usec).
v_elapsed_msg	Type:	Return
	Datatype:	Citext
	Default value:	Null
	Description:	Message containing the parsed, readable output displaying the total duration.

Examples

```
select in_db_util_fn_get_duration(elapsed_usec);

select in_db_util_fn_get_duration(7200010003);

select in_db_util_fn_get_duration
(
  in_db_util_fn_get_usec('2024-02-01 12:22:37.824032-06', '2024-02-01 14:22:37.834035-06')
);
```

IN_DB_UTIL_FN_GET_RPM

The IN_DB_UTIL_FN_GET_RPM function will calculate the Rows Per Minute (RPM) when providing the number of affected rows along with the total duration in microseconds (usec). This can be used to help quantify the throughput of an operation when measuring the duration and number of rows affected.

Parameter	Description	
in_rows	Type:	Input
	Datatype:	Numeric
	Default value:	Null
	Description:	Number of rows to use for calculating the RPM.
in_usec	Type:	Input
	Datatype:	Numeric
	Default value:	Null
	Description:	Number of microseconds (usec) to use for calculating the RPM.
v_rows_per_minute	Type:	Return
	Datatype:	Bigint
	Default value:	Null
	Description:	The calculated rows per minute based on the provided number of rows and microseconds.

Examples

```
SELECT * FROM in_db_util_fn_get_rpm(50000,60000000);

DO $$
DECLARE
  v_rows numeric := 50000;
  v_usec numeric := 60000000;
  v_rpm bigint;
BEGIN
  v_rpm := in_db_util_fn_get_rpm(v_rows, v_usec);
  RAISE INFO 'Rows Per Minute = %', LTRIM(TO_CHAR(v_rpm, '999,999,999,999'));
END;
$$;
```

IN_DB_UTIL_FN_GET_USEC

The IN_DB_UTIL_FN_GET_USEC function will calculate total duration in microseconds (usec) between two timestamps. The output of this function can be passed into the IN_DB_UTIL_FN_GET_DURATION function or the IN_DB_UTIL_FN_GET_RPM function, along with the total number of rows, to quantify the throughput of an operation.

Parameter	Description	
in_start_timestamp	Type:	Input
	Datatype:	Timestamp
	Default value:	Null
	Description:	The starting timestamp
in_end_timestamp	Type:	Input
	Datatype:	Timestamp
	Default value:	Null
	Description:	The ending timestamp
v_elapsed_usec	Type:	Return
	Datatype:	Numeric
	Default value:	Null
	Description:	The total number of microseconds between the provided timestamps.

Example

```
select in_db_util_fn_get_usec('2024-02-01 12:22:37.824032-06', '2024-02-01
14:22:37.834035-06');
```

IN_DB_UTIL database cleanup stored procedures

The following stored procedures can be used to cleanup old audit and archived workflow data from the Perceptive Content database.

IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA

The IN_DB_UTIL_SP_CLEANUP_AUDIT_DATA stored procedure can be used to cleanup audit data that is stored in the database. This procedure will delete data from the IN_AUDIT_DETAIL, and IN_AUDIT_OBJ, and IN_AUDIT tables based on the value of the **in_keep_days** parameter or the **in_begin_timestamp** and **in_end_timestamp** parameters.

Be aware that the execution of this procedure could create contention on the various audit tables and result in temporary degradation of performance. It is advised to only execute this procedure after hours or on weekends while monitoring for database contention to ensure minimal or no impact to the Perceptive Content application. If necessary, utilize smaller batch sizes, using the **in_batch_size** parameter, to help keep transaction sizes smaller and reduce the scope and duration of database locks.

Note You can use the **in_mode** = 'Count' option to get a row count of qualifying rows based on the supplied values for the **in_keep_days** or **in_begin_timestamp** and **in_end_timestamp** parameters to determine the scope of the cleanup.

Note To delete all audit records execute the procedure using **in_mode** = 'Truncate'. Alternatively, you can also use **in_mode** = 'Delete' along with **in_keep_days** = 0. The truncate mode is the fastest option for deleting all audit records.

It is recommended that you run the VACUUM ANALYZE command after deleting large numbers of rows from the audit tables.

The following parameters are required.

Parameter	Description	
in_mode	Type:	Input
	Datatype:	citext_max_9
	Default value:	Help
	Description:	Specifies which mode to use when running the stored procedure.
	Options:	Help = Display Usage Information Count = Count the number of rows that would qualify for deletion Delete = Delete the rows that qualify for deletion Truncate = Delete all rows by truncating the tables
in_keep_days	Type:	Input
	Datatype:	Integer
	Default value:	Null
	Description:	Specifies the number of days of audit data that you want to keep
	Options:	0 = Keep 0 days and delete all rows Any number of days greater than 0
in_begin_timestamp	Type:	Input
	Datatype:	Timestamp
	Default value:	Null
	Description:	Specifies the start date[time] when deleting audit data based on a date range. Examples of allowable formats: 'April 01, 2024 [8:00 AM]' '2024-04-01 [08:00:00]'
in_end_timestamp	Type:	Input
	Datatype:	Timestamp

Parameter	Description	
	Default value:	Null
	Description:	Specifies the end date[time] when deleting audit data based on a date range. Examples of allowable formats: 'April 01, 2024 [5:00 PM]' '2024-04-01 [17:00:00]'
in_convert_to_utc	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to convert the in_begin_timestamp and in_end_timestamp parameters to UTC so they align with the creation_time column of the in_audit table, which stores the creation_time of the audit records in UTC. If you are already providing the date[time] in UTC then use False to prevent unnecessary conversion.
	Options:	True = Convert to UTC False = Do not convert to UTC
in_use_time	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to include the time parts for the date ranges.
	Options:	False = 12AM True = Use the current time if specifying the in_keep_days parameter or use the time provided with the in_begin_timestamp and in_end_timestamp parameters.
in_batch_size	Type:	Input
	Datatype:	Integer
	Default value:	10000
	Description:	Specifies the batch size if breaking up deletion into smaller transactions.
	Options:	0 = Single transaction Any number greater than or equal to 50
in_max_minutes	Type:	Input

Parameter	Description	
	Datatype:	Integer
	Default value:	0
	Description:	Specifies a time limit to determine whether to start deleting the next batch of rows. This only applies if using in_batch_size greater than 0. Batches will run through completion, but a new batch will not start if time has expired.
	Options:	0 = No Time Limit Any number of minutes greater than 0
in_display_batches	Type:	Input
	Datatype:	Boolean
	Default value:	True
	Description:	Specifies whether to display the batch results and batch summary after each batch has completed.
	Options:	True = Display the batch results after each batch. False = Do not display the batch results after each batch.
in_include_summary	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to fetch and display the row counts of all qualifying rows for each table. Fetching summary data during execution could result in a much longer execution time as all qualifying rows are counted before any rows are deleted.
	Options:	False = Do not fetch and display the number of qualifying rows. True = Fetch and display the number of qualifying rows for each table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the SQL being executed during the script. Will also display background operations including session setup and state changes and call stack details.

Parameter	Description	
	Options:	No = Do not display additional debugging information. Yes = Display additional debugging information.
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Recommended index:

The following index is recommended to facilitate efficient searches using the CREATION_TIME column.

```
CREATE INDEX AUDIT_IDX_01 ON INUSER.IN_AUDIT
(
    CREATION_TIME ASC,
    AUDIT_ID ASC
) TABLESPACE "INOW_Idx";
```

Examples

Display Usage Information Only

```
CALL in_db_util_sp_cleanup_audit_data('HELP', null, null, null, null, null, null,
null, null, null, null, null)
```

KEEP_DAYS Examples:

The following will get a count of all the rows in each of the three audit tables.

```
CALL in_db_util_sp_cleanup_audit_data('COUNT', 0, null, null, null, 'yes', null, null,
null, null, null, null)
```

The following will delete all audit data that is older than 365 days using batch sizes of 10,000 rows.

Note From a transaction standpoint the number of rows is based on qualifying rows in the IN_AUDIT table but the transaction also includes the deletion of rows from the IN_AUDIT_OBJ and IN_AUDIT_DETAIL tables which could have more or less than 10,000 each that correspond to the rows in the IN_AUDIT table. That means that each transaction could affect more than three times the number of rows as specified in the **in_batch_size** parameter and a batch size of 10,000 could result in a transaction size of roughly 30,000 or more rows. If necessary, use a smaller batch size to reduce the scope and duration of each transaction.

```
CALL in_db_util_sp_cleanup_audit_data
(
    'DELETE', 365, NULL, NULL, FALSE, FALSE, 10000, 0, TRUE, FALSE, 'NO', NULL
);

DO $$
DECLARE
    v_mode          citext_max_6 := 'DELETE';
    v_keep_days     integer      := 365;
    v_begin_timestamp timestamp  := NULL;
    v_end_timestamp timestamp    := NULL;
    v_convert_to_utc boolean     := FALSE;
```

```

v_use_time          boolean          := FALSE;
v_batch_size        integer           := 10000;
v_max_minutes        integer           := 0;
v_display_batches   boolean           := TRUE;
v_include_summary   boolean           := FALSE;
v_debug             citext_max_6      := 'NO';
v_call_results       boolean;
BEGIN
  CALL in_db_util_sp_cleanup_audit_data
  (
    v_mode, v_keep_days, v_begin_timestamp, v_end_timestamp,
    v_convert_to_utc, v_use_time, v_batch_size, v_max_minutes,
    v_display_batches, v_include_summary, v_debug, v_call_results
  );
  RAISE INFO '%', v_call_results;
END;
$$;

```

DATE RANGE Example:

The following example will delete all audit data that was created on April 1, 2024 between 8am and 5pm using a batch size 10,000 rows.

Note To preserve the time part of the timestamps you must set `in_use_time = true`.

Note Setting `in_convert_to_utc = true` means the provided timestamps will be converted to UTC so that it aligns with the timestamps in the database. Do not set `in_convert_to_utc = true` if you are already providing the UTC timestamp as this will result in an unnecessary conversion and the date/time will be off by the time zone differential that is currently set for the database server.

```

CALL in_db_util_sp_cleanup_audit_data
(
  'DELETE', NULL, '2024-04-01 08:00:00', '2024-04-01 17:00:00', TRUE, TRUE, 10000, 0,
  TRUE, TRUE, 'NO', NULL
);

DO $$
DECLARE
  v_mode          citext_max_6 := 'DELETE';
  v_keep_days      integer      := NULL;
  v_begin_timestamp timestamp   := 'April 01 2024 8:00 AM';
  v_end_timestamp  timestamp   := 'April 01 2024 5:00 PM';
  v_convert_to_utc boolean      := TRUE;
  v_use_time        boolean      := TRUE;
  v_batch_size     integer       := 10000;
  v_max_minutes    integer       := 0;
  v_display_batches boolean     := TRUE;
  v_include_summary boolean     := TRUE;
  v_debug          citext_max_6 := 'NO';
  v_call_results   boolean;
BEGIN
  CALL inuser.in_db_util_sp_cleanup_audit_data
  (
    v_mode, v_keep_days, v_begin_timestamp, v_end_timestamp,
    v_convert_to_utc, v_use_time, v_batch_size, v_max_minutes,
    v_display_batches, v_include_summary, v_debug, v_call_results
  );
  RAISE INFO '%', v_call_results;
END;
$$;

```

TRUNCATE Example:

The following will display a summary of the current row counts then truncate all three of the audit tables. To truncate the three audit tables without counting and displaying the summary data then specify false for the **in_include_summary** parameter. This is the fastest way to remove all audit data from the three audit tables.

```
CALL in_db_util_sp_cleanup_audit_data
(
  'TRUNCATE', null, null, null, null, null, null, null, null, TRUE, null, null
);
```

IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA

The IN_DB_UTIL_SP_CLEANUP_WF_ARCH_DATA stored procedure can be used to cleanup archived workflow data. This procedure will delete data from the IN_WF_ITEM_ARCH and IN_WF_ITEM_HIST_ARCH and IN_WF_ITEM_QUEUE_HIST_ARCH tables based on the value of the **in_keep_days** parameter or the **in_begin_timestamp** and **in_end_timestamp** parameters.

Be aware that the execution of this procedure could create contention on the various workflow archive tables and result in temporary degradation of performance. It is advised to only execute this procedure after hours or on weekends while monitoring for database contention to ensure minimal or no impact to the Perceptive Content application. If necessary, utilize smaller batch sizes, using the **in_batch_size** parameter, to help keep transaction sizes smaller and reduce the scope and duration of database locks.

When running this procedure you have the option to specify whether or not to delete archived workflow data for content that still exists (Folders and Documents). By default, this option (**in_has_content**) is set to false which restricts the procedure from deleting archived workflow data for folders and documents that still exist. To override this behavior, you can execute the procedure with **in_has_content** = true to indicate that you want to delete all qualifying archived workflow items regardless of whether there is referencing content that may still exist.

Note If you use the **in_has_content** = true option to remove archived workflow content for documents and folders that still exist then be aware that it's possible, depending on the specified time frame of the cleanup, that only part of the archived workflow history will be deleted for any given document or folder. This would result in an incomplete workflow archive history when viewing the properties for that document or folder. However, the remaining archived workflow history will be deleted the next time the procedure is executed if the specified time frame of the cleanup includes the remaining archived workflow history for the document or folder.

Note You can use the **in_mode** = 'Count' option to get a row count of qualifying rows based on the supplied values for the **in_keep_days** or **in_begin_timestamp** and **in_end_timestamp** parameters to determine the scope of the cleanup.

Note To delete all archived workflow records execute the procedure using **in_mode** = 'Truncate'. Alternatively, you can also use **in_mode** = 'Delete' along with **in_keep_days** = 0 and **in_has_content** = True. The truncate mode is the fastest option for deleting all archived workflow records.

It is recommended that you run the VACUUM ANALYZE command after deleting large numbers of rows from the workflow archive tables.

The following parameters are required.

Parameter	Description	
in_mode	Type:	Input
	Datatype:	citext_max_9
	Default value:	Help
	Description:	Specifies which mode to use when running the stored procedure.

Parameter	Description	
	Options:	Help = Display Usage Information Count = Count the number of rows that would qualify for deletion Delete = Delete the rows that qualify for deletion Truncate = Delete all rows by truncating the tables
in_has_content	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to remove archived workflow items from the workflow archive tables that still has existing content in the IN_DOC table (documents) and the IN_PROJ table (folders).
	Options:	False = Only delete archived workflow items for which the referencing content no longer exists. True = Delete all qualifying archived workflow items regardless of whether there is referencing content that may still exist.
in_keep_days	Type:	Input
	Datatype:	Integer
	Default value:	Null
	Description:	Specifies the number of days of archived workflow data that you want to keep.
	Options:	0 = Delete All Rows Any number of days greater than 0
in_begin_timestamp	Type:	Input
	Datatype:	Timestamp
	Default value:	Null
	Description:	Specifies the start date[time] when deleting archived workflow data based on a date range. Examples of allowable formats: 'April 01, 2024 [8:00 AM]' '2024-04-01 [8:00:00]'
in_end_timestamp	Type:	Input
	Datatype:	Timestamp
	Default value:	Null

Parameter	Description	
	Description:	Specifies the end date[time] when deleting archived workflow data based on a date range. Examples of allowable formats: 'April 01, 2024 [5:00 PM]' '2024-04-01 [17:00:00]'
in_convert_to_utc	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to convert the in_begin_timestamp and in_end_timestamp parameters to UTC so they align with the creation_time column of the in_wf_item_arch table, which stores the creation_time of the archived workflow item record in UTC.
	Options:	True = Convert to UTC False = Do not convert to UTC
in_use_time	Type:	Input
	Datatype:	Integer
	Default value:	False
	Description:	False = 12AM True = Current UTC time if specifying the in_keep_days parameter or uses the time provided with the in_begin_timestamp and in_end_timestamp parameters
	Options:	False = 12AM True = Current UTC time if specifying the in_keep_days parameter or uses the time provided with the in_begin_timestamp and in_end_timestamp parameters
in_batch_size	Type:	Input
	Datatype:	Integer
	Default value:	10000
	Description:	Specifies the batch size if breaking up deletion into smaller transactions.
	Options:	0 = Single transaction Any number greater than or equal to 50
in_max_minutes	Type:	Input
	Datatype:	Integer

Parameter	Description	
	Default value:	0
	Description:	Specifies a time limit to determine whether to start deleting the next batch of rows. This only applies if using in_batch_size greater than 0. Batches will run through completion, but a new batch will not start if time has expired.
	Options:	0 = No Time Limit Any number of minutes greater than 0
in_display_batches	Type:	Input
	Datatype:	Boolean
	Default value:	True
	Description:	Specifies whether to display the batch results and batch summary after each batch is completed.
	Options:	True = Display the batch results after each batch. False = Do not display the batch results after each batch.
in_include_summary	Type:	Input
	Datatype:	Boolean
	Default value:	False
	Description:	Specifies whether to fetch and display the row counts of all qualifying rows for each table. Fetching summary data during execution could result in longer execution times as all qualifying rows are counted before any rows are deleted.
	Options:	False = Do not fetch and display the number of qualifying rows. True = Fetch and display the number of qualifying rows for each table.
in_debug	Type:	Input
	Datatype:	citext_max_6
	Default value:	No
	Description:	Specifies whether to display the SQL being executed during the script. Will also display background operations including session setup and state changes and call stack details.
	Options:	NO = Do not display additional debugging information. YES = Display additional debugging information.

Parameter	Description	
out_return_val	Type:	Output
	Datatype:	Boolean
	Default value:	Null
	Description:	Specifies whether the execution of the procedure was successful or not. True = Success False = Error

Index Consideration:

The following index change can be considered to facilitate improved index usage when using the default option of **in_has_content** = false.

```
CREATE INDEX WF_ARCH_IDX_CLEANUP ON inuser.IN_WF_ITEM_ARCH
(
  ARCHIVE_TIME ASC,
  OBJ_ID ASC,
  ITEM_ID ASC
)
TABLESPACE "INOW_Indx";
```

Examples

Display Usage Information Only

```
CALL in_db_util_sp_cleanup_wf_arch_data
(
  'HELP', null, null, null, null, null, null, null, null, null, null, null
);
```

KEEP_DAYS Example:

The following will delete all archived workflow data that is older than 365 days using batch sizes of 1,000 rows.

Note that from a transaction standpoint the number of rows is based on qualifying rows in the IN_WF_ITEM_ARCH table but the transaction also includes the deletion of rows from the IN_WF_ITEM_HIST_ARCH and IN_WF_ITEM_QUEUE_HIST_ARCH tables which will likely have more than 10,000 rows that correspond to the rows in the IN_WF_ITEM_ARCH table. That means that each transaction will affect significantly more rows than specified in the **in_batch_size** parameter. If necessary, use a smaller batch size to reduce the scope and duration of each transaction.

```
CALL in_db_util_sp_cleanup_wf_arch_data
(
  'DELETE', FALSE, 365, NULL, NULL, FALSE, FALSE, 10000, 0, TRUE, FALSE, 'NO', NULL
);

DO $$
DECLARE
  v_mode          citext_max_9 := 'DELETE';
  v_has_content   boolean      := FALSE;
  v_keep_days     integer      := 365;
  v_begin_timestamp timestamp   := NULL;
  v_end_timestamp timestamp    := NULL;
  v_convert_to_utc boolean     := FALSE;
  v_use_time      boolean     := FALSE;
```

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

```
v_batch_size      integer      := 10000;
v_max_minutes     integer      := 0;
v_display_batches boolean      := TRUE;
v_include_summary boolean     := FALSE;
v_debug           citext_max_6 := 'NO';
v_call_results    boolean;
BEGIN
CALL in_db_util_sp_cleanup_wf_arch_data
(
  v_mode, v_has_content, v_keep_days, v_begin_timestamp, v_end_timestamp,
  v_convert_to_utc, v_use_time, v_batch_size, v_max_minutes,
  v_display_batches, v_include_summary, v_debug, v_call_results
);
RAISE INFO '%', v_call_results;
END;
$$;
```

DATE RANGE Example:

The following example will delete archived workflow data that does not still reference existing content that was created on April 1, 2024 between 8am and 5pm using a batch size of 10,000 rows.

Note To preserve the time part of the timestamps you must set `in_use_time = true`.

Note Setting `in_convert_to_utc = true` means the provided timestamps will be converted to UTC so that it aligns with the timestamps in the database. Do not set `in_convert_to_utc = true` if you are already providing the UTC timestamp as this will result in an unnecessary conversion and the date/time will be off by the time zone differential that is currently set for the database server.

```
CALL in_db_util_sp_cleanup_wf_arch_data
(
  'DELETE', FALSE, NULL, '2024-04-01 08:00:00', '2024-04-01
17:00:00', TRUE, TRUE, 10000, 0, TRUE, TRUE, 'NO', NULL
);

DO $$
DECLARE
  v_mode           citext_max_9 := 'DELETE';
  v_has_content    boolean      := FALSE;
  v_keep_days      integer      := NULL;
  v_begin_timestamp timestamp    := 'April 01 2024 8:00 AM';
  v_end_timestamp  timestamp    := 'April 01 2024 5:00 PM';
  v_convert_to_utc boolean      := TRUE;
  v_use_time       boolean      := TRUE;
  v_batch_size     integer      := 10000;
  v_max_minutes    integer      := 0;
  v_display_batches boolean     := TRUE;
  v_include_summary boolean     := TRUE;
  v_debug          citext_max_6 := 'NO';
  v_call_results   boolean;
BEGIN
CALL in_db_util_sp_cleanup_wf_arch_data
(
  v_mode, v_has_content, v_keep_days, v_begin_timestamp, v_end_timestamp,
  v_convert_to_utc, v_use_time, v_batch_size, v_max_minutes,
  v_display_batches, v_include_summary, v_debug, v_call_results
);
RAISE INFO '%', v_call_results;
END;
$$;
```

TRUNCATE Example:

The following will display a summary of the current row counts then truncate all three of the workflow archive tables. To truncate the three audit tables without counting and displaying the summary data then

specify false for the `in_include_summary` parameter. This is the fastest way to remove all archived workflow data from the three tables.

```
CALL inuser.in_db_util_sp_cleanup_wf_arch_data
(
  'TRUNCATE', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, NULL
);
```

Troubleshooting commands

- To query the log table to view the history or errors, execute the following query.

```
SELECT * FROM inuser.in_db_util_log ORDER BY log_id;
```

- To view active session details for the SYNC and UPGRADE identifiers, execute the following command.

```
CALL inuser.in_db_util_sp_sessions_display('ALL');
```

- To view active session details for the IN_DB_UTIL identifier, execute the following command.

```
CALL inuser.in_db_util_sp_sessions_display('IN_DB_UTIL');
```

- To view call stack details, execute the following command.

```
CALL inuser.in_db_util_sp_callstack_print('ALL');
```

- To view the contents of the table containing active session details and attributes, execute the following query. This table is queried by the [IN_DB_UTIL_SP_SESSIONS_DISPLAY](#) procedure.

```
SELECT * FROM inuser.in_db_util_session_attributes ORDER BY identifier;
```

- To view the contents of the table containing call stack depth between stored procedures execute the following query. This table is queried by the [IN_DB_UTIL_SP_CALLSTACK_PRINT](#) procedure.

```
SELECT * FROM inuser.in_db_util_callstack ORDER BY identifier, depth;
```

- To view the postgres advisory locks for the stored procedures and identifiers, that have been allocated and are being held by each session, execute the following query:

```
SELECT * FROM pg_locks WHERE locktype = 'advisory';
```

Cleanup after a disconnected or killed session

You can use the following procedures to clear the IN_DB_UTIL framework (session and callstack and advisory locks) for the procedures and identifiers or to manually clear the call stack and session attribute tables as needed due to an unhandled exception or after manually cancelling an operation before completion.

If you are cleaning up after a different session (not the session you are running the following commands from) be aware that you will not be able to unlock/release the the advisory locks for that session and you will need to disconnect that session if it still has a connection to the database and still holding advisory locks.

This should not be necessary in most cases as the [IN_DB_UTIL_SP_SESSION_SETUP](#) and [IN_DB_UTIL_SP_CLEAR_IDENTIFIER](#) procedures will manage these tables during execution.

Important Do not execute any of the following commands if any operations are still in progress and using the IN_DB_UTIL framework as it will clear the call-stack and session attributes which will lead to undesired results and unhandled exceptions.

- Clear session and call stack for all identifiers (SYNC and UPGRADE)

```
CALL inuser.in_db_util_sp_clear_identifier('ALL', null, null, 'YES', null);
```

- Clear session and call stack for only the SYNC identifier

```
CALL inuser.in_db_util_sp_clear_identifier('SYNC', null, null, 'YES', null);
```

- Clear session and call stack for only the UPGRADE identifier

```
CALL inuser.in_db_util_sp_clear_identifier('UPGRADE', null, null, 'YES', null);
```

- Clear session and call stack for only the IN_DB_UTIL identifier (Include PID if different than yours)

```
CALL in_db_util_sp_clear_identifier('in_db_util',
'in_db_util_sp_cleanup_audit_data', 22456, 'YES', null);
```

- Release all advisory locks being held by the current session

```
select pg_advisory_unlock_all();
```

Debugging

You can use the **in_debug** parameter with most procedures if additional logging is necessary.

Depending on the procedure, this can result in a very verbose execution that contains state change information and call stack details and other background call details to assist with debugging certain issues.

```
in_debug = 'YES'
```

Removing the IN_DB_UTIL package

Execute the following to remove all the components of the IN_DB_UTIL package from a database.

```
DO $$
DECLARE
  v_schema          citext := 'inuser';
  v_type            citext;
  v_name            citext;
  v_drop_procs      integer := 0;
  v_exec_sql        citext;
  v_sqlstate        citext;
  v_error           citext;
  v_context         citext;
  v_hint            citext;
  v_error_msg       citext;

BEGIN
  -- Drop tables unless it appears they are actively being used
  IF EXISTS (SELECT 1 FROM pg_tables WHERE schemaname = v_schema AND tablename =
'in_db_util_session_attributes') THEN
    IF (SELECT COUNT(*) FROM in_db_util_session_attributes WHERE status <> 'IDLE') = 0
    THEN
      BEGIN
        RAISE INFO 'DROP TABLE in_db_util_session_attributes;';
        EXECUTE 'DROP TABLE in_db_util_session_attributes;';
        v_drop_procs := v_drop_procs + 1;

        EXCEPTION
          WHEN OTHERS THEN
            RAISE WARNING E'\nNOTICE: Could not drop the IN_DB_UTIL_SESSION_ATTRIBUTES
table.\n';
      END;
    ELSE
      RAISE NOTICE E'\nNOTICE: There are active sessions currently using the
IN_DB_UTIL_SESSION_ATTRIBUTES table.\n';
    END IF;
END IF;
```

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

```

ELSE
    v_drop_procs := v_drop_procs + 1;
END IF;

IF EXISTS (SELECT 1 FROM pg_tables WHERE schemaname = v_schema AND tablename =
'in_db_util_callstack') THEN
    IF (SELECT COUNT(*) FROM in_db_util_callstack) = 0 THEN
        BEGIN
            RAISE INFO 'DROP TABLE in_db_util_callstack;';
            EXECUTE 'DROP TABLE in_db_util_callstack;';
            v_drop_procs := v_drop_procs + 1;

            EXCEPTION
                WHEN OTHERS THEN
                    RAISE WARNING E'\nNOTICE: Could not drop the IN_DB_UTIL_CALLSTACK
table.\n';
        END;
    ELSE
        RAISE NOTICE E'\nNOTICE: There are active sessions currently using the
IN_DB_UTIL_CALLSTACK table.\n';
    END IF;
ELSE
    v_drop_procs := v_drop_procs + 1;
END IF;

-- Drop programs if both tables were dropped
IF v_drop_procs = 2 THEN
    FOR v_type, v_name IN
        (
            SELECT routine_type, routine_name
            FROM information_schema.routines
            WHERE specific_schema = 'inuser' AND routine_name LIKE 'in_db_util%'
            UNION
            SELECT 'TYPE', user_defined_type_name
            FROM information_schema.user_defined_types
            WHERE user_defined_type_schema = 'inuser' AND user_defined_type_name LIKE
'in_db_util%'
            ORDER BY 1
        )
    LOOP
        BEGIN
            RAISE INFO 'DROP % %.'; v_type, v_schema, v_name;
            v_exec_sql := 'DROP ' || v_type || ' ' || v_schema || '.' || v_name || ';';
            EXECUTE v_exec_sql;

            EXCEPTION
                WHEN OTHERS THEN
                    GET STACKED DIAGNOSTICS
                        v_sqlstate = returned_sqlstate,
                        v_error = message_text,
                        v_context = pg_exception_context,
                        v_hint = pg_exception_hint;
                    v_error_msg := v_sqlstate || ' - ' || v_error;

                    -- RAISE WARNING '%', v_context;
                    RAISE WARNING '%', v_error_msg;
                    IF v_hint IS NOT NULL THEN RAISE WARNING 'hint: %', v_hint; END IF;
        END;
    END LOOP;
ELSE
    -- IN_DB_UTIL tables must be in use so do not proceed
    -- Display IN_DB_UTIL sessions for all identifiers

```

```

RAISE NOTICE 'NOTICE: Could not drop the IN_DB_UTIL_CALLSTACK or
IN_DB_UTIL_SESSION_ATTRIBUTES table.';

BEGIN
CALL in_db_util_sp_sessions_display('IN_DB_UTIL');
CALL in_db_util_sp_sessions_display('ALL');

EXCEPTION WHEN OTHERS THEN NULL;
END;
RAISE NOTICE E'\n\nNOTICE: The IN_DB_UTIL package was not updated since it appears
that it may be currently in use.\n\n';
END IF;

END $$;

```

Additional examples

The following examples help demonstrate how to leverage the IN_DB_UTIL framework as well as some of the other stored procedures and functions available within the IN_DB_UTIL package.

Calculating and displaying throughput metrics (RPM)

The following example combines the [IN_DB_UTIL_FN_GET_DURATION](#) function and the [IN_DB_UTIL_SP_GET_RPM](#) stored procedure to demonstrate how you could use them to record and or display throughput metrics of various operations.

Throughput is measured as Rows Per Minute (RPM)

```

DO $$
DECLARE

v_start_timestamp    timestamp;
v_total_rows         bigint;
v_elapsed_usec       numeric;
v_rows_per_minute    bigint;
v_elapsed_msg        citext;

BEGIN
-- Record the start time
v_start_timestamp := clock_timestamp();

-- Execute any operation that affects rows
PERFORM * FROM inuser.IN_DOC;

-- Record the number of affected rows
GET DIAGNOSTICS v_total_rows = ROW_COUNT;

-- Calculate Duration in Microseconds (usec)
v_elapsed_usec := inuser.in_db_util_fn_get_usec(v_start_timestamp,
clock_timestamp());

-- Get Rows Per Minute
v_rows_per_minute := inuser.in_db_util_fn_get_rpm(v_total_rows,v_elapsed_usec);

-- Build Throughput Message to Display or Log
v_elapsed_msg := 'Total Rows Selected was ' || LTRIM(TO_CHAR(v_total_rows,
'999,999,999'));
v_elapsed_msg := v_elapsed_msg || CHR(10) || 'Total Duration was ' ||
inuser.in_db_util_fn_get_duration(v_elapsed_usec);
v_elapsed_msg := v_elapsed_msg || CHR(10) || 'Total Rows Per Minute was ' ||
LTRIM(TO_CHAR(v_rows_per_minute, '999,999,999'));

```

```

-- Display the Throughput Metrics
RAISE INFO E'\n%', v_elapsed_msg;
END;
$$;

Results Displayed Throughput in the Messages Tab (pgAdmin 4)

INFO:
Total Rows Selected was 1,509
Total Duration was 446 Microseconds
Total Rows Per Minute was 203,004,484
DO

```

Generating schema DDL

The following example utilizes the various GET DDL procedures to generate the entire inuser and inemuser schemas.

Note that the generated DDL relies on the citext extension as well as the various user defined domains that are used for the case insensitive datatypes, which are created by the SunflowerPG.sql script.

Examples

```

DO $$
DECLARE
    v_table          citext := null;
    v_debug          citext := 'no';
    v_schema_ddl    citext;
    v_info_msg       citext;
    v_return_val     boolean;
BEGIN
    CALL in_db_util_sp_get_ddl_sequence(v_table, true, v_debug, v_schema_ddl,
v_return_val);
    v_info_msg := v_schema_ddl;
    CALL in_db_util_sp_get_ddl_table(v_table, v_debug, v_schema_ddl, v_return_val);
    v_info_msg := v_info_msg || v_schema_ddl;
    CALL in_db_util_sp_get_ddl_pk(v_table, null, v_debug, v_schema_ddl, v_return_val);
    v_info_msg := v_info_msg || v_schema_ddl;
    CALL in_db_util_sp_get_ddl_index(v_table, null, v_debug, v_schema_ddl,
v_return_val);
    v_info_msg := v_info_msg || v_schema_ddl;
    CALL in_db_util_sp_get_ddl_fk(v_table, null, 'from', v_debug, v_schema_ddl,
v_return_val);
    v_info_msg := v_info_msg || v_schema_ddl;

    RAISE INFO '%', v_info_msg;
END;
$$;

```

The following example utilizes the various GET DDL procedures to generate a portion of the inuser schema based on the query used get the table names. In this case, the audit tables.

```

DO $$
DECLARE
    v_table_name    citext;
    v_debug         citext := 'no';
    v_schema_ddl    citext;
    v_info_msg      citext := '';
    v_return_val    boolean;
    v_sql_query     citext;
    v_tab_list      refcursor;
    v_pass          smallint := 4;
BEGIN

```

Perceptive Content Database IN_DB_UTIL Package for PostgreSQL

```
v_sql_query := 'SELECT tablename FROM pg_tables WHERE schemaname = ''inuser'' AND
tablename LIKE ''in_audit%''';

WHILE v_pass > 0 LOOP
BEGIN
    IF v_pass = 4 THEN v_info_msg := v_info_msg || E'\n\n-- ##### TABLES
#####\n';          END IF;
    IF v_pass = 3 THEN v_info_msg := v_info_msg || E'\n\n-- ##### PRIMARY
KEYS #####\n';    END IF;
    IF v_pass = 2 THEN v_info_msg := v_info_msg || E'\n\n-- ##### INDEXES
#####\n';        END IF;
    IF v_pass = 1 THEN v_info_msg := v_info_msg || E'\n\n-- ##### FOREIGN
KEYS #####\n';    END IF;

    OPEN v_tab_list FOR EXECUTE v_sql_query;
    FETCH NEXT FROM v_tab_list INTO v_table_name;
    WHILE FOUND LOOP
        IF v_pass = 4 THEN CALL in_db_util_sp_get_ddl_table(v_table_name, v_debug,
v_schema_ddl, v_return_val); END IF;
        IF v_pass = 3 THEN CALL in_db_util_sp_get_ddl_pk(v_table_name, null, v_debug,
v_schema_ddl, v_return_val); END IF;
        IF v_pass = 2 THEN CALL in_db_util_sp_get_ddl_index(v_table_name, null, v_debug,
v_schema_ddl, v_return_val); END IF;
        IF v_pass = 1 THEN CALL in_db_util_sp_get_ddl_fk(v_table_name, null, 'from',
v_debug, v_schema_ddl, v_return_val); END IF;
        v_info_msg := v_info_msg || v_schema_ddl;
        FETCH NEXT FROM v_tab_list INTO v_table_name;
    END LOOP;
    CLOSE v_tab_list;

    v_pass := v_pass -1;
END;
END LOOP;
RAISE INFO '%', v_info_msg;
END;
$$;
```