

# Perceptive Content Database IN\_DB\_UTIL Package for Oracle Server

## Reference Guide

Version: Foundation 22.2

Written by: Documentation Team, R&D  
Date: June 2023

# 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.

© 2023 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

<b>Documentation Notice</b> .....	<b>2</b>
<b>Overview</b> .....	<b>6</b>
<b>Benefits</b> .....	<b>6</b>
<b>Create the IN_DB_UTIL package</b> .....	<b>6</b>
<b>Database objects</b> .....	<b>7</b>
<b>Application context</b> .....	<b>9</b>
IN_CONTEXT.....	9
<b>Record types and table types</b> .....	<b>10</b>
IN_CONTEXT_ATTR.....	10
IN_CONTEXT_ATTRIBUTES.....	13
IN_OBJECTS.....	13
IN_OBJECT_DDL.....	13
IN_PARAMETERS.....	14
IN_PARAMETER_VALUES.....	14
IN_CALLSTACK.....	14
IN_CALLSTACK_TABLE.....	14
IN_IDENTIFIER_TABLE.....	15
IN_DB_UTIL_LOG.....	15
<b>Framework stored procedures and functions</b> .....	<b>15</b>
IN_CALLSTACK_EXTEND.....	15
IN_CALLSTACK_TRIM.....	16
IN_CALLSTACK_PRINT.....	17
IN_CALLSTACK_GET_HIST.....	18
IN_CONTEXT_SET.....	20
IN_CONTEXT_CLEAR.....	21
IN_CONTEXT_GET.....	22
IN_CONTEXT_PRINT.....	22
IN_SESSION_SET.....	24
IN_SESSION_CLEAR.....	26
IN_SESSION_CLEAR_ALL.....	28
IN_LOGGER.....	32
CLEAR_VARIABLES.....	33
EXECUTE_DML.....	33

PRINT_SQL.....	35
CHECK_PARAMETER.....	36
CHECK_TIME.....	38
GET_IN_DB_UTIL_VERSION.....	39
<b>Schema metadata stored procedures and functions.....</b>	<b>40</b>
GET_IN_DB_SCHEMA_VERSION.....	40
PARSE_SCHEMA_VERSION.....	41
PRINT_SCHEMA_HIST.....	42
PRINT_INDEX_DDL.....	43
PRINT_FK.....	44
GET_FK_DDL.....	45
<b>Schema change stored procedures.....</b>	<b>46</b>
DROP_INDEX.....	47
DROP_PK.....	48
DROP_FK.....	49
DISABLE_FK.....	50
ENABLE_FK.....	52
RENAME_OBJECT.....	54
RENAME_CONSTRAINT.....	56
<b>Database information procedures and functions.....</b>	<b>58</b>
GET_DB_INFO.....	58
GET_CPU_COUNT.....	58
GET_DB_CONNECTIONS.....	58
GET_DB_CONNECTIONS_COUNT.....	59
<b>Session configuration stored procedures.....</b>	<b>59</b>
SET-NLS_SESSION_PARAMETERS.....	59
SET_DBA_ROLE.....	60
SET_PARALLEL.....	60
DISABLE_PARALLEL.....	61
TRACE_SESSION.....	61
<b>Metrics stored procedures and functions.....</b>	<b>62</b>
GET_SECONDS.....	62
GET_DURATION.....	63
GET_RPM.....	64
<b>Database maintenance stored procedures.....</b>	<b>65</b>

CLEANUP_AUDIT_DATA.....	65
REBUILD_INDEXES.....	70
<b>Exceptions.....</b>	<b>73</b>
<b>Troubleshooting commands.....</b>	<b>74</b>
<b>Cleanup after an early exit.....</b>	<b>75</b>
<b>Debugging.....</b>	<b>75</b>
<b>Removing the IN_DB_UTIL package.....</b>	<b>75</b>
<b>Additional examples.....</b>	<b>75</b>
Calculating and displaying throughput metrics (RPM).....	75
Coding with the MAX_MINUTES parameter.....	76

## Overview

The following document provides an overview and reference for the Perceptive Content IN\_DB\_UTIL Oracle database package.

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 features that are commonly used during database schema upgrades and maintenance operations.

The IN\_DB\_UTIL database package is leveraged by the Perceptive Content database IN\_DB\_UPGRADE package during database schema 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 mechanism to view specific details about session activity from any other database session to expose the actions and progress for any procedures leveraging the framework. These session attributes are stored in memory (SYS\_CONTEXT) and exposed using the IN\_CONTEXT\_PRINT stored procedure.
- 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 three defined identifiers.
- Provides a collection of procedures and functions for common database tasks which are leveraged by the IN\_DB\_UPGRADE package during database schema upgrades.
- Provides a collection of procedures and functions to assist with calculating throughput metrics to report on the performance of scripted operations.
- Provides a collection of procedures and functions to perform various database maintenance tasks such as deleting old audit data or rebuilding indexes.

## Create the IN\_DB\_UTIL package

You can create the IN\_DB\_UTIL package by executing the latest version of the Perceptive Content IN\_DB\_UTIL package script for Oracle. You can find this script on the [community.hyland.com](http://community.hyland.com) customer portal download site for Perceptive Content and is contained within the Perceptive Content Foundation DDL Release Scripts zip files.

The current IN\_DB\_UTIL version for Perceptive Content Release 22.2 is version 4.2.

Execute the following script to create the IN\_DB\_UTIL package. SQL\*Plus is the recommended tool for creating the IN\_DB\_UTIL package and for executing the various procedures and functions included in the package.

- Execute the following script.

```
PerceptiveContentDB_IN_DB_UTIL_Oracle_Package_v4.2.sql
```

## Database objects

The following table contains a list of the database objects that comprise the IN\_DB\_UTIL package which are created when the script above is executed.

Object Name	Object Type	Scope
IN_CONTEXT	Application Context	Global
IN_CONTEXT_ATTR	Record Type	Framework
IN_CONTEXT_ATTRIBUTES	Table Type	Framework
IN_OBJECTS	Record Type	Framework
IN_OBJECT_DDL	Table Type	Framework
IN_PARAMETERS	Record Type	Framework
IN_PARAMETER_VALUES	Nested Table Type	Framework
IN_CALLSTACK	Record Type	Framework
IN_CALLSTACK_TABLE	Nested Table Type	Framework
IN_IDENTIFIER_TABLE	Associative Array	Framework
IN_DB_UTIL_LOG	Table	Framework
IN_CALLSTACK_EXTEND	Stored Procedure	Framework
IN_CALLSTACK_TRIM	Stored Procedure	Framework
IN_CALLSTACK_PRINT	Stored Procedure	Framework
IN_CALLSTACK_GET_HIST	Stored Procedure	Framework
IN_CONTEXT_SET	Stored Procedure	Framework
IN_CONTEXT_CLEAR	Stored Procedure	Framework
IN_CONTEXT_GET	Function	Framework
IN_CONTEXT_PRINT	Stored Procedure	Framework
IN_SESSION_SET	Stored Procedure	Framework
IN_SESSION_CLEAR	Stored Procedure	Framework
IN_SESSION_CLEAR_ALL	Stored Procedure	Framework
IN_LOGGER	Stored Procedure	Framework

Object Name	Object Type	Scope
CLEAR_VARIABLES	Stored Procedure	Framework
EXECUTE_DML	Stored Procedure	Framework
PRINT_SQL	Stored Procedure	Framework
CHECK_PARAMETER	Stored Procedure	Framework
CHECK_TIME	Function	Framework
GET_IN_DB_UTIL_VERSION	Function	Framework Metadata
GET_IN_DB_SCHEMA_VERSION	Function	Schema Metadata
PARSE_SCHEMA_VERSION	Stored Procedure	Schema Metadata
PRINT_SCHEMA_HIST	Stored Procedure	Schema Metadata
PRINT_INDEX_DDL	Stored Procedure	Schema Metadata
PRINT_FK	Stored Procedure	Schema Metadata
GET_FK_DDL	Stored Procedure	Schema Metadata
DROP_INDEX	Stored Procedure	Schema Change
DROP_PK	Stored Procedure	Schema Change
DROP_FK	Stored Procedure	Schema Change
DISABLE_FK	Stored Procedure	Schema Change
ENABLE_FK	Stored Procedure	Schema Change
RENAME_OBJECT	Stored Procedure	Schema Change
RENAME_CONSTRAINT	Stored Procedure	Schema Change
GET_DB_INFO	Stored Procedure	Database Information
GET_CPU_COUNT	Function	Database Information
GET_DB_CONNECTIONS	Stored Procedure	Database Check
GET_DB_CONNECTIONS_COUNT	Function	Database Check
SET-NLS_SESSION_PARAMETERS	Stored Procedure	Session
SET_DBA_ROLE	Stored Procedure	Session

Object Name	Object Type	Scope
<a href="#">SET_PARALLEL</a>	Stored Procedure	Session
<a href="#">DISABLE_PARALLEL</a>	Stored Procedure	Session
<a href="#">TRACE_SESSION</a>	Stored Procedure	Session
<a href="#">GET_SECONDS</a>	Stored Procedure	Metrics
<a href="#">GET_DURATION</a>	Stored Procedure	Metrics
<a href="#">GET_RPM</a>	Stored Procedure	Metrics
<a href="#">CLEANUP_AUDIT_DATA</a>	Stored Procedure	Maintenance
<a href="#">REBUILD_INDEXES</a>	Stored Procedure	Maintenance

## Application context

The IN\_DB\_UTIL framework leverages the Oracle database global application context features to store various session attributes in memory. These session attributes are utilized and managed by the IN\_DB\_UTIL package using various Oracle DBMS\_SESSION subprograms such as SET\_IDENTIFIER, and CLEAR\_IDENTIFIER, and SET\_CONTEXT, and CLEAR\_CONTEXT.

## IN\_CONTEXT

A globally accessible context called IN\_CONTEXT is created as part of the IN\_DB\_UTIL framework. The IN\_DB\_UTIL package is used to interface with and manipulate the various attributes associated with the IN\_CONTEXT application context.

```
CREATE OR REPLACE CONTEXT IN_CONTEXT USING INUSER.IN_DB_UTIL ACCESSED GLOBALLY;
```

The IN\_DB\_UTIL package currently supports three different client identifiers depending on the scope of the procedures utilizing the IN\_DB\_UTIL framework. The three client identifiers are IN\_DB\_UTIL, SYNC, and UPGRADE. The IN\_DB\_UTIL framework will set the IN\_CONTEXT namespace identifier, that is specified during framework initialization, to properly scope the session attributes when setting up a session. This includes extending the call stack for the appropriate identifier as well as setting the various context attributes for the identifier within the IN\_CONTEXT namespace.

The list of supported attributes is defined in the [IN\\_CONTEXT\\_ATTR](#) record type. The IN\_CONTEXT application context attributes are visible using the [IN\\_CONTEXT\\_PRINT](#) procedure or the [IN\\_CONTEXT\\_GET](#) function or by querying the database directly using the SYS\_CONTEXT function while specifying the IN\_CONTEXT namespace along with any of the supported attributes.

### Examples

```
EXEC DBMS_SESSION.SET_IDENTIFIER(V_IDENTIFIER);
SELECT SYS_CONTEXT('IN_CONTEXT', 'STATUS') AS STATUS FROM DUAL;
```

### Example Output

```
SQL> EXEC DBMS_SESSION.SET_IDENTIFIER('IN_DB_UTIL');

PL/SQL procedure successfully completed.
```

```
SQL> SELECT SYS_CONTEXT('IN_CONTEXT', 'STATUS') AS STATUS FROM DUAL;
```

```
STATUS
-----
IDLE
```

A session that is using the IN\_DB\_UTIL package, will have the IN\_CONTEXT namespace identifier that it is currently using, associated with it. This will be visible in the CLIENT\_IDENTIFIER column of the V\$SESSION view or when querying the CLIENT\_IDENTIFIER attribute from SYS\_CONTEXT for the USERENV namespace.

### Examples

```
SELECT CLIENT_IDENTIFIER FROM V$SESSION WHERE SID = SYS_CONTEXT('USERENV', 'SID');
SELECT SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') AS CLIENT_IDENTIFIER FROM DUAL;
```

### Example Output

```
SQL> SELECT CLIENT_IDENTIFIER FROM V$SESSION WHERE SID = SYS_CONTEXT('USERENV', 'SID');
```

```
CLIENT_IDENTIFIER
-----
IN_DB_UTIL
```

```
SQL> SELECT SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') AS CLIENT_IDENTIFIER FROM DUAL;
```

```
CLIENT_IDENTIFIER
-----
IN_DB_UTIL
```

## Record types and table types

The IN\_DB\_UTIL package uses several record types and table types to pass a collection of parameters between the various subprograms of the package.

### IN\_CONTEXT\_ATTR

The IN\_CONTEXT\_ATTR type is a record type that contains the following list of fields which are used by the IN\_CONTEXT\_ATTRIBUTES table type to store and pass the values of the various IN\_CONTEXT namespace attributes between subprograms within the IN\_DB\_UTIL package. The values for these fields are visible using the IN\_CONTEXT\_PRINT procedure or fetched using the IN\_CONTEXT\_GET function or by querying the database directly using the SYS\_CONTEXT function using the IN\_CONTEXT namespace along with the specified attribute.

```
TYPE IN_CONTEXT_ATTR IS RECORD
(
  TAG                VARCHAR2(10),
  STATUS             VARCHAR2(128),
  OPERATION          VARCHAR2(128),
  CALLED_BY          VARCHAR2(128),
  START_TIME         VARCHAR2(128),
  MAX_TIME           VARCHAR2(128),
  MAX_MINUTES        VARCHAR2(128),
  OBJECT             VARCHAR2(257),
  SUBOBJECT          VARCHAR2(257),
  TABLESPACE        VARCHAR2(128),
  BATCH_SIZE         VARCHAR2(128),
  BULK_LIMIT         VARCHAR2(128),
  PARALLEL           VARCHAR2(128),
  LOGGING            VARCHAR2(9),
```

```

SCHEMA_TARGET  VARCHAR2(40),
SCHEMA_BASE    VARCHAR2(40),
SCHEMA_VERSION VARCHAR2(40),
IS_SYNC_UPGRADE VARCHAR2(3),
OS_USER        VARCHAR2(128),
DB_SESSION     VARCHAR2(128)
);
    
```

Field Name	Description
TAG	<p>A unique identifier that is created when a program initializes the IN_DB_UTIL framework for the specified identifier. The TAG will persist until the calling program exits cleanly and the session is cleared. The TAG is used to ensure that any attempts by another instance of the program cannot interfere with the current instance already leveraging the identifier within the IN_CONTEXT namespace.</p> <p>The default value is generated using the following command to return a random string of ten uppercase and alpha-numeric characters.</p> <p>DBMS_RANDOM.STRING('X',10)</p>
STATUS	<p>The status of the session associated with the specified identifier.</p> <p>Possible options are IDLE or EXECUTING</p> <p>The default is IDLE</p>
OPERATION	<p>The name of the procedure that is currently executing under the specified identifier. This is the highest procedure in the call stack and is also referenced as the CALLER in some cases.</p>
CALLED_BY	<p>The name of the procedure that called the procedure (OPERATION) that is currently executing under the specified identifier. This is the next highest procedure in the call stack (COUNT-1) and is also referenced as the CALLER_HIST in some cases.</p>
START_TIME	<p>The timestamp for when the session was initialized or when the current unit of work was started. The START_TIME will also get updated to SYSTIMESTAMP anytime the IN_CONTEXT_SET procedure is used to either set the STATUS to EXECUTING or to explicitly set the START_TIME to record the start time for an operation within the namespace for the identifier.</p>
MAX_TIME	<p>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.</p>
MAX_MINUTES	<p>For procedures that include the MAX_MINUTES parameter this field records the specified number of minutes which is used to define an end time (MAX_TIME) that a procedure is allowed to start new operations. If MAX_MINUTES has elapsed and the current time is greater than the calculated MAX_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.</p> <p>The default is 0 which is equal to unlimited.</p>

Field Name	Description
OBJECT	The primary or parent object name affected by the current actions executing under the specified identifier. This is usually a table name.
SUBOBJECT	The name of the object affected by the current actions executing under the specified identifier. This could be an index, or column, or constraint.
TABLESPACE	If applicable, the name of the tablespace that contains the OBJECT or SUBOBJECT affected by the current actions executing under the specified identifier.
BATCH_SIZE	<p>The batch size (number of rows) to populate work queues (cursors). Currently only used by CLEANUP_AUDIT_DATA and during IN_DB_UPGRADE SYNC operations. BATCH_SIZE is used to scope the size of transactions by providing a variable limit on the number of rows to process before committing a transaction.</p> <p>Could also be used for TOP N constructs for SELECT, INSERT, UPDATE, DELETE, and MERGE operations to iterate over the data in smaller batches to keep transaction sizes smaller.</p> <p>The default is 100,000 rows.</p>
BULK_LIMIT	<p>Used to impose a variable limit on the number of rows to fetch at one time during a bulk collect from the work queue batch cursor.</p> <p>Currently only leveraged by the IN_DB_UPGRADE SYNC operations (SYNC_TABLE_IN_DOC) during BULK COLLECT INTO with the LIMIT clause to throttle the number of rows collected and processed at one time to help reduce memory usage by the program.</p> <p>Default is 10,000 rows.</p>
PARALLEL	<p>The degree of parallelism the session will be forced to use during execution. Can be used for operations that leverage the parallel parameter such as index creation or for parallel execution of qualifying SQL statements.</p> <p>Default is 0 which is equal to CPU_COUNT minus 1 for less than 8 CPU's or minus 2 for 8 or more CPU's.</p>
LOGGING	<p>Used to define whether an operation is logged (LOGGING) or not logged (NOLOGGING) wherever possible. Commonly used during index creation.</p> <p>Default is LOGGING.</p>
SCHEMA_TARGET	The target schema version during upgrades. This parameter is used by the IN_DB_UPGRADE package and specified when running the UPGRADE_SETUP procedure during Perceptive Content database upgrades. It is used to conditionally determine which schema changes are necessary.
SCHEMA_BASE	The base version of the schema which only includes the version number without the schema type designators such as 'a' for advanced. This is used for a simplified schema check during conditional execution based on the schema version.
SCHEMA_VERSION	The current schema version. This parameter contains the full schema version number including the type designators such as 'a' for Advanced if applicable.

Field Name	Description
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
OS_USER	This is the operating system user name for the client process that initiated the database session and instantiated the framework using the IN_SESSION_SET procedure.  Default value is SYS_CONTEXT('USERENV', 'OS_USER')
DB_SESSION	This is the session identifier for the session that instantiated the framework using the IN_SESSION_SET procedure.  Default value is SID,SERIAL# of the session.

## IN\_CONTEXT\_ATTRIBUTES

The IN\_CONTEXT\_ATTRIBUTES type is a table type that is based on the [IN\\_CONTEXT\\_ATTR](#) record type. It is used to store and pass session attribute values by the [IN\\_CONTEXT\\_GET](#) function when fetching the IN\_CONTEXT namespace attributes for a specified identifier.

```
TYPE IN_CONTEXT_ATTRIBUTES IS TABLE OF IN_CONTEXT_ATTR;
```

## IN\_OBJECTS

The IN\_OBJECTS type is a record type that contains the following list of fields which are used by the [IN\\_OBJECT\\_DDL](#) table type to store a collection of objects and associated SQL commands.

```
TYPE IN_OBJECTS IS RECORD
(
  OBJECT_NAME      VARCHAR2(257),
  SUBOBJECT_NAME   VARCHAR2(128),
  OBJECT_SQL       VARCHAR2(2000)
);
```

Field Name	Description
OBJECT_NAME	The name of the object. This is usually the table name associated with the SUBOBJECT_NAME.
SUBOBJECT_NAME	The name of the sub-object. This could be an index name, a constraint name, or a column name.
OBJECT_SQL	The SQL associated with the object. DDL or DML.

## IN\_OBJECT\_DDL

The IN\_OBJECT\_DDL type is a table type that is based on the [IN\\_OBJECTS](#) record type. It is used to define variables to store a collection of SQL commands for processing or passing between procedures.

```
TYPE IN_OBJECT_DDL IS TABLE OF IN_OBJECTS INDEX BY PLS_INTEGER;
```

## IN\_PARAMETERS

The `IN_PARAMETERS` type is a record type that contains the following list of fields which are used by the `IN_PARAMETER_VALUES` table type to store a collection of parameters and their values.

```
TYPE IN_PARAMETERS IS RECORD
(
  PARAMETER_NAME  VARCHAR2(128),
  PARAMETER_VALUE VARCHAR2(128)
);
```

Field Name	Description
PARAMETER_NAME	The name of the parameter.
PARAMETER_VALUE	The parameter value associated with the parameter.

## IN\_PARAMETER\_VALUES

The `IN_PARAMETER_VALUES` type is a table type that is based on the `IN_PARAMETERS` record type. It is used to define variables to store a collection of parameters and their values for processing. This is used by the `SET-NLS_SESSION_PARAMETERS` procedure when setting the standard session parameters for the Perceptive Content database.

```
TYPE IN_PARAMETER_VALUES IS TABLE OF IN_PARAMETERS INDEX BY PLS_INTEGER;
```

## IN\_CALLSTACK

The `IN_CALLSTACK` type is a record type that contains the following list of fields which are used by the `IN_CALLSTACK_TABLE` table type to store the call stack details for each of the identifiers in the `IN_CONTEXT` namespace.

```
TYPE IN_CALLSTACK IS RECORD
(
  TAG                VARCHAR2(10),
  CONTEXT_OPERATION  VARCHAR2(128)
);
```

Field Name	Description
TAG	The unique tag associated with the session currently using the specified identifier of the <code>IN_CONTEXT</code> namespace.
CONTEXT_OPERATION	The name of the procedure or operation that is associated with the specific call stack entry.

## IN\_CALLSTACK\_TABLE

The `IN_CALLSTACK_TABLE` type is a table type that is based on the `IN_CALLSTACK` record type. It is used to define variables to store the call stack for each of the `IN_CONTEXT` namespace identifiers. It is used by the various call stack related procedures including `IN_CALLSTACK_EXTEND`, `IN_CALLSTACK_TRIM`, `IN_CALLSTACK_GET_HIST`, `IN_CALLSTACK_PRINT`.

```
TYPE IN_CALLSTACK_TABLE IS TABLE OF IN_CALLSTACK INDEX BY PLS_INTEGER;
```

## IN\_IDENTIFIER\_TABLE

The IN\_IDENTIFIER TABLE type is a string type associative array used to define variables to hold the IN\_CONTEXT namespace identifiers when looping through them for processing. It is currently only used by the IN\_CONTEXT\_PRINT stored procedure.

```
TYPE IN_IDENTIFIER_TABLE IS TABLE OF VARCHAR2(128) INDEX BY PLS_INTEGER;
```

## IN\_DB\_UTIL\_LOG

The IN\_DB\_UTIL\_LOG table is used by the IN\_LOGGER stored procedure to record calls to certain stored procedures as well as any exceptions that occur during execution of the IN\_DB\_UTIL and IN\_DB\_UPGRADE packages.

This table is created when the IN\_DB\_UTIL package is created but will also be recreated by the IN\_LOGGER stored procedure if it does not exist at the time of execution.

Most of the columns in the IN\_DB\_UTIL\_LOG table align with the IN\_CONTEXT\_ATTR record type.

## 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\_CALLSTACK\_EXTEND

The IN\_CALLSTACK\_EXTEND stored procedure is responsible for incrementing the call stack depth for the specified identifier when the IN\_SESSION\_SET procedure is executed during session initialization.

The call stack for each identifier is stored in memory using a global variable defined with the IN\_CALLSTACK\_TABLE table type. The call stack array is incremented using the CONTEXT\_OPERATION parameter value and the associated array index is the call stack depth.

The IN\_CALLSTACK\_EXTEND procedure should never need to be manually executed.

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	Specifies the identifier for which to extend the call stack.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE
CONTEXT_OPERATION	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)

Parameter	Description	
	<b>Default value:</b>	None
	<b>Description:</b>	Specifies the name of the procedure that called the IN_SESSION_SET procedure to initialize the session within the framework.

**Exceptions**

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [TAG\\_MISMATCH](#)
- [OTHERS](#)

**Example**

```
IN_CALLSTACK_EXTEND(V_IDENTIFIER, V_CONTEXT_OPERATION);
```

## IN\_CALLSTACK\_TRIM

The IN\_CALLSTACK\_TRIM stored procedure is responsible for decrementing the call stack depth for the specified identifier when the [IN\\_SESSION\\_CLEAR](#) procedure is executed during session tear down.

The call stack for each identifier is stored in memory using a global variable defined with the [IN\\_CALLSTACK\\_TABLE](#) table type.

The IN\_CALLSTACK\_TRIM procedure should never need to be manually executed.

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	Specifies the identifier for which to trim the call stack.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE

**Exceptions**

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [TAG\\_MISMATCH](#)
- [OTHERS](#)

**Example**

```
IN_CALLSTACK_TRIM(V_IDENTIFIER);
```

## IN\_CALLSTACK\_PRINT

The IN\_CALLSTACK\_PRINT stored procedure is used to fetch and print the entire call stack for the specified identifier.

The call stack for each identifier is stored in memory using a global variable defined with the [IN\\_CALLSTACK\\_TABLE](#) table type.

The entire call stack for a session is only visible to the current session, it is not visible to any other session. However, the two most recent call stack entries are recorded within the IN\_CONTEXT namespace for the identifier and is visible to all sessions by referencing the OPERATION and CALLED BY attributes using the [IN\\_CONTEXT\\_PRINT](#) procedure or the SYS\_CONTEXT function.

Parameter	Description	
IDENTIFIER	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	ALL (SYNC and UPGRADE)
	Description:	Specifies the identifier for which to print the call stack.
	Options:	IN_DB_UTIL SYNC UPGRADE ALL

### Exception

- [OTHERS](#)

### Examples

```
EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT;
EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT(V_IDENTIFIER);
```

### Example output

Example output during the execution the CLEANUP\_AUDIT\_DATA procedure.

```
***** Call Stack Begin *****

***** IN_DB_UTIL Call Stack *****

Depth  Tag          Procedure
-----
1      0527044CB3    CLEANUP_AUDIT_DATA

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

**Note** Depending on the IN\_DB\_UPGRADE procedure currently using the IN\_DB\_UTIL framework, it could be consuming either the SYNC or UPGRADE identifier or both (ALL). The call stack for each identifier is extended or trimmed separately so it's normal to see varying depths for each identifier based on the sequence of procedures being executed.

**Example** output at a specific point during the execution of the IN\_DB\_UPGRADE.UPTIME\_STEPS procedure.

```

***** Call Stack Begin *****

***** SYNC Call Stack *****

Depth  Tag          Procedure
-----  -
1      M54LEDMP8     CREATE_PK_IN_DOC

***** UPGRADE Call Stack *****

Depth  Tag          Procedure
-----  -
1      M54LEDMP8     UPTIME_STEPS
2      M54LEDMP8     SCHEMA_UPGRADE_7403_7500
3      M54LEDMP8     UPTIME_STEPS_IN_DOC
4      M54LEDMP8     CREATE_PK_IN_DOC

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

```

**Example** output at a specific point during the execution of the IN\_DB\_UPGRADE.DOWNTIME\_STEPS procedure.

```

***** Call Stack Begin *****

***** SYNC Call Stack *****

Depth  Tag          Procedure
-----  -
1      2VIDAEWULF   DOWNTIME_STEPS
2      2VIDAEWULF   DOWNTIME_STEPS_IN_DOC
3      2VIDAEWULF   CREATE_FK_IN_DOC

***** UPGRADE Call Stack *****

Depth  Tag          Procedure
-----  -
1      2VIDAEWULF   DOWNTIME_STEPS
2      2VIDAEWULF   SCHEMA_UPGRADE_7403_7500
3      2VIDAEWULF   DOWNTIME_STEPS_IN_DOC
4      2VIDAEWULF   CREATE_FK_IN_DOC

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

```

## IN\_CALLSTACK\_GET\_HIST

The IN\_CALLSTACK\_GET\_HIST stored procedure is used to get the last two programs from the call stack for populating the CALLER and CALLER\_HIST parameters after a calling procedure either initiates the session using the IN\_SESSION\_SET procedure or clears its session using the IN\_SESSION\_CLEAR procedure.

This call stack information is only accessible within the current session, it is not visible from any other session.

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	Specifies the identifier for which to get the call stack history
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE
TAG	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(10)
	<b>Default value:</b>	DBMS_RANDOM.STRING('X',10)
	<b>Description:</b>	The unique tag associated with the session currently using the specified identifier of the IN_CONTEXT namespace.
CALLER	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The procedure name that is currently executing for the specified identifier. The most recent entry in the call stack.
CALLER_HIST	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The procedure name that called the procedure that is currently executing for the specified identifier. The next most recent entry in the call stack.

**Exceptions**

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [OTHERS](#)

**Example**

```
IN_CALLSTACK_GET_HIST(V_IDENTIFIER, V_TAG, V_CALLER, V_CALLER_HIST);
```

## IN\_CONTEXT\_SET

The IN\_CONTEXT\_SET stored procedure is used to set the value for any of the supported attributes for the specified identifier of the IN\_CONTEXT namespace. The list of supported attributes is aligned with the fields of the [IN\\_CONTEXT\\_ATTR](#) record type.

Parameter	Description	
ATTRIBUTE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the attribute being set for the identifier.
	<b>Options:</b>	Reference the IN_CONTEXT_ATTR record type for options.
ATTRIBUTE_VALUE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(257)
	<b>Default value:</b>	None
	<b>Description:</b>	The value of the attribute being set for the identifier.
	<b>Options:</b>	Reference the IN_CONTEXT_ATTR record type for options.
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	The identifier of the IN_CONTEXT namespace.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE

### Exceptions

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- INVALID\_NUMBER – Non-Numeric value specified for MAX\_MINUTES.
- [OTHERS](#)

### Examples

```
IN_DB_UTIL.IN_CONTEXT_SET(ATTRIBUTE_NAME, ATTRIBUTE_VALUE, IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('STATUS', 'EXECUTING', IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('START_TIME', SYSTIMESTAMP, IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('MAX_MINUTES', 60, IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('OPERATION', 'CREATE_INDEXES', IDENTIFIER);
```

```
IN_DB_UTIL.IN_CONTEXT_SET('OBJECT', 'IN_DOC', IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('SUBOBJECT', 'DOC_IDX2', IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_SET('TABLESPACE', 'INDX', IDENTIFIER);
```

## IN\_CONTEXT\_CLEAR

The IN\_CONTEXT\_CLEAR stored procedure is used to clear the value for any of the supported attributes for the specified IN\_CONTEXT namespace identifier. The list of supported attributes is aligned with the fields of the IN\_CONTEXT\_ATTR record type.

Parameter	Description	
ATTRIBUTE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the attribute being cleared for the identifier.
	<b>Options:</b>	Reference the IN_CONTEXT_ATTR record type for options.
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	The identifier of the IN_CONTEXT namespace.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE ALL

### Exceptions

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [OTHERS](#)

### Examples

```
IN_DB_UTIL.IN_CONTEXT_CLEAR(ATTRIBUTE_NAME, IDENTIFIER);
IN_DB_UTIL.IN_CONTEXT_CLEAR('STATUS', IDENTIFIER );
IN_DB_UTIL.IN_CONTEXT_CLEAR('OPERATION', IDENTIFIER );
IN_DB_UTIL.IN_CONTEXT_CLEAR('OBJECT', IDENTIFIER );
IN_DB_UTIL.IN_CONTEXT_CLEAR('SUBOBJECT', IDENTIFIER );
IN_DB_UTIL.IN_CONTEXT_CLEAR('ALL', IDENTIFIER );
```

## IN\_CONTEXT\_GET

The IN\_CONTEXT\_GET function is used to fetch the current attribute values for the specified identifier of the IN\_CONTEXT namespace. The list of supported attributes align with the fields of the IN\_CONTEXT\_ATTR record type.

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	IN_DB_UTIL
	<b>Description:</b>	The identifier of the IN_CONTEXT namespace.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE

### Exceptions

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [OTHERS](#)

### Examples

```
SELECT * FROM TABLE(IN_DB_UTIL.IN_CONTEXT_GET('SYNC'));
SELECT * FROM TABLE(IN_DB_UTIL.IN_CONTEXT_GET('UPGRADE'));
SELECT * FROM TABLE(IN_DB_UTIL.IN_CONTEXT_GET('IN_DB_UTIL'));
SELECT STATUS, MAX_TIME FROM TABLE(IN_DB_UTIL.IN_CONTEXT_GET('IN_DB_UTIL'));
SELECT DB_SESSION FROM TABLE(IN_DB_UTIL.IN_CONTEXT_GET('IN_DB_UTIL'));
```

## IN\_CONTEXT\_PRINT

The IN\_CONTEXT\_PRINT stored procedure is used to interface with the IN\_CONTEXT\_GET function to get and print the attribute values for the specified identifier of the IN\_CONTEXT context namespace.

The output of IN\_CONTEXT\_PRINT includes the last two calls in the call stack for the specified identifier. The OPERATION attribute, which is also referred to as CALLER in some cases, is the current operation in the call stack. The CALLED\_BY attribute is the previous operation (COUNT-1), which is also referred to as CALLER\_HIST in some cases.

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	ALL
	<b>Description:</b>	The identifier of the IN_CONTEXT namespace.

Parameter	Description	
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE ALL (SYNC and UPGRADE)

### Examples

```
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT (IDENTIFIER);
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT;
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT ('ALL');
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT ('SYNC');
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT ('UPGRADE');
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT ('IN_DB_UTIL');
```

### Example output

Example from the IN\_DB\_UTIL.CLEANUP\_AUDIT\_DATA stored procedure.

```
Current DB Time: 2022-07-29 13:36:39.776000
Current UTC Time: 2022-07-29 18:36:39.776000

Current Activity for the IN_DB_UTIL identifier:

IN_DB_UTIL TAG:                7B37GKNLZ7
IN_DB_UTIL STATUS:             EXECUTING
IN_DB_UTIL OPERATION:          CLEANUP_AUDIT_DATA
IN_DB_UTIL CALLED BY:          NULL
IN_DB_UTIL START TIME:         2022-07-29 13:36:39.774000
IN_DB_UTIL MAX TIME:           2022-07-29 14:36:39.774000
IN_DB_UTIL MAX MINUTES:        60
IN_DB_UTIL OBJECT:             IN_AUDIT
IN_DB_UTIL SUBOBJECT:          NULL
IN_DB_UTIL TABLESPACE:        NULL
IN_DB_UTIL BATCH SIZE:         10000
IN_DB_UTIL BULK LIMIT:         NULL
IN_DB_UTIL PARALLEL:           1
IN_DB_UTIL LOGGING:            NULL
IN_DB_UTIL OS USER:            LOCAL_DOMAIN\USER_NAME
IN_DB_UTIL DB_SESSION:         742,52540
```

Example from the IN\_DB\_UTIL.UPGRADE.UPTIME\_STEPS stored procedure.

```
Current DB Time: 2022-08-01 09:22:37.849000
Current UTC Time: 2022-08-01 14:22:37.849000

Current Activity for the SYNC identifier:

SYNC TAG:                      NULL
SYNC STATUS:                   IDLE
SYNC OPERATION:                 NULL
SYNC CALLED BY:                 NULL
SYNC START TIME:               NULL
SYNC MAX TIME:                  NULL
SYNC MAX MINUTES:              NULL
SYNC OBJECT:                    NULL
SYNC SUBOBJECT:                 NULL
SYNC TABLESPACE:              NULL
SYNC BATCH SIZE:               NULL
SYNC BULK LIMIT:               NULL
SYNC PARALLEL:                  NULL
SYNC LOGGING:                   NULL
SYNC OS USER:                   NULL
```

```

SYNC DB_SESSION:          NULL

Current Activity for the UPGRADE identifier:

UPGRADE TAG:              WNCBNP25DV
UPGRADE STATUS:           EXECUTING
UPGRADE OPERATION:        CREATE_INDEXES_IN_DOC
UPGRADE CALLED BY:        UPTIME_STEPS_IN_DOC
UPGRADE START TIME:       2022-08-01 09:22:37.849000
UPGRADE MAX TIME:         2022-08-01 10:22:35.891000
UPGRADE MAX MINUTES:      60
UPGRADE OBJECT:           NEW_DOC
UPGRADE SUBOBJECT:        NEW_DOC_IDX2
UPGRADE TABLESPACE:      INDX
UPGRADE PARALLEL:         10
UPGRADE LOGGING:          LOGGING
UPGRADE SCHEMA TARGET:    7.7.0.0
UPGRADE SCHEMA VERSION:   7.1.3.3
UPGRADE IS SYNC UPGRADE:  YES
UPGRADE OS USER:          LOCAL_DOMAIN\USER_NAME
UPGRADE DB_SESSION:       742,52540

PL/SQL procedure successfully completed.

```

## IN\_SESSION\_SET

The `IN_SESSION_SET` stored procedure is used to initialize the `IN_DB_UTIL` framework for a single identifier. It is responsible for claiming initial control or extending the call stack for the specified identifier and adjusting the context attributes to reflect the changes to the call stack.

This procedure should be executed whenever a procedure is running within the `IN_DB_UTIL` framework.

This procedure ensures the specified identifier is not already claimed by another session based on a comparison of the specified TAG.

This procedure calls the `IN_CALLSTACK_EXTEND` procedure to extend the call stack as well as set initial values for some of the `IN_CONTEXT` attributes.

If a procedure is setting up the framework and is scoped to ALL identifiers (SYNC and UPGRADE) then it must be called separately for both identifiers using the same TAG value.

If a TAG is not specified (NULL) then this procedure will generate a random string using the following command: `DBMS_RANDOM.STRING('X',10);`

The `IN_SESSION_SET` procedure is responsible for the following `IN_DB_UTIL` framework initialization steps:

- Ensure there is not already different session executing under the specified identifier. If there is then the `IS_EXECUTING` exception will be raised.
- Extend the framework call stack under the specified identifier.
  - `IN_CALLSTACK_EXTEND`
- Set session level attributes of the `IN_CONTEXT` namespace for the specified identifier.

Any procedure leveraging the `IN_DB_UTIL` framework should have the following exceptions defined to properly handle the exceptions that could be raised by this procedure.

- **FAILED\_CHECK** – Invalid IDENTIFIER specified. Checked using the `CHECK_PARAMETER` procedure.

- **IS\_EXECUTING** – Returned if a procedure attempts to claim control or extend the call stack of an identifier that is already claimed by another session using a different TAG value.
- **OTHERS** – Catch all other undefined exceptions.

Parameter	Description	
IDENTIFIER	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	IN_DB_UTIL
	Description:	The identifier for the session attempting to claim control of the IN_CONTEXT namespace identifier.
	Options:	IN_DB_UTIL SYNC UPGRADE
TAG	Type:	INPUT
	Datatype:	VARCHAR2(10)
	Default value:	DBMS_RANDOM.STRING('X',10)
	Description:	The unique tag associated with the session.
CONTEXT_OPERATION	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	IN_DB_UTIL
	Description:	Specifies the name of the procedure attempting to claim the IN_CONTEXT namespace to initialize the session within the framework, which includes extending the call stack for the specified identifier.

#### Exceptions

- [FAILED\\_CHECK](#) – [INVALID\\_IDENTIFIER](#)
- [IS\\_EXECUTING](#)
- [OTHERS](#)

#### Examples

```
IN_DB_UTIL.IN_SESSION_SET(V_IDENTIFIER, V_TAG, V_CONTEXT_OPERATION);
IN_DB_UTIL.IN_SESSION_SET('IN_DB_UTIL', V_TAG, 'AUDIT_DATA_CLEANUP');
```

## IN\_SESSION\_CLEAR

The IN\_SESSION\_CLEAR stored procedure is used for clearing the IN\_DB\_UTIL framework for a single identifier of the IN\_CONTEXT namespace. It is responsible for decrementing the call stack and adjusting the context attributes to reflect the changes to the call stack.

This procedure should be executed whenever a procedure, that has also initialized the session with [IN\\_SESSION\\_SET](#), has finished its unit of work and is tearing down the session.

The TAG provided when executing the procedure must match the existing TAG associated with the specified identifier or an error will be returned. This is to help prevent any other sessions, that are initialized with a different TAG, from inadvertently making any changes to the specified identifier.

If the procedure is the last entry in the call stack then the call stack will be decremented and the associated IN\_CONTEXT namespace attributes will be reset to IDLE for the specified identifier, freeing up the identifier for other sessions to use.

If the procedure is not the last entry in the call stack then the call stack will be decremented and the IN\_CONTEXT namespace attributes will be updated to reflect the current state of the call stack for the specified identifier without freeing up the identifier for other sessions to use.

The IN\_SESSION\_CLEAR procedure is responsible for the following framework de-initialization steps.

- Decrement the call stack for the specified identifier.
  - [IN\\_CALLSTACK\\_TRIM](#)
- Clear the session attributes for the specified identifier.
  - [IN\\_CONTEXT\\_SET](#)
  - [IN\\_CONTEXT\\_CLEAR](#)

Any procedure leveraging the IN\_DB\_UTIL framework should have the following exceptions defined to properly handle the exceptions that could be raised by this procedure.

- **FAILED\_CHECK** – Invalid IDENTIFIER specified. Checked using the [CHECK\\_PARAMETER](#) procedure.
- **TAG\_MISMATCH** – Returned if a procedure attempts to clear a session but the specified TAG value does not match the TAG value that is currently occupying the specified identifier.
- **OTHERS** – Catch all other undefined exceptions.

Under normal circumstances, IN\_SESSION\_CLEAR procedure should never need to be manually executed. However, it can be used to manually clear the call stack and session attributes as needed due to an unhandled exception or after manually cancelling an operation before completion.

**Important** Do not manually execute this procedure 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	
IDENTIFIER	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	IN_DB_UTIL

Parameter	Description	
	<b>Description:</b>	The identifier for which to clear the session attributes.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE
TAG	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(10)
	<b>Default value:</b>	None
	<b>Description:</b>	The unique tag associated with the session currently occupying the specified identifier of the IN_CONTEXT namespace.

### Exceptions

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [TAG\\_MISMATCH](#)
- [OTHERS](#)

### Examples

The following example demonstrates how this procedure is called from within another procedure after it has completed its work and is cleanup up.

```
IN_DB_UTIL.IN_SESSION_CLEAR(V_IDENTIFIER, V_TAG);
```

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.

Note that the TAG must match the tag currently associated with the specified identifier.

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

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR('IN_DB_UTIL', 'WNQJAP4QWJ');
```

### Example output

The following example demonstrates the clearing of the IN\_DB\_UTIL identifier by executing the IN\_SESSION\_CLEAR procedure to clear the call stack and session attributes after cancelling the IN\_CLEANUP\_AUDIT\_DATA procedure prior to completion.

```
SQL> EXEC IN_DB_UTIL.IN_CONTEXT_PRINT('IN_DB_UTIL');
```

```
Current DB Time: 2022-08-04 15:17:34.391000
Current UTC Time: 2022-08-04 20:17:34.391000
```

```
Current Activity for the IN_DB_UTIL identifier:
```

```
IN_DB_UTIL TAG:           FUF9OPDVK8
IN_DB_UTIL STATUS:       EXECUTING
IN_DB_UTIL OPERATION:    CLEANUP_AUDIT_DATA
IN_DB_UTIL CALLED BY:    NULL
```

```

IN_DB_UTIL START TIME:      2022-08-04 14:05:33.312000
IN_DB_UTIL MAX TIME:       2022-08-04 15:07:41.157000
IN_DB_UTIL MAX MINUTES:    60
IN_DB_UTIL OBJECT:        IN_AUDIT
IN_DB_UTIL SUBOBJECT:      NULL
IN_DB_UTIL TABLESPACE:    INDX
IN_DB_UTIL BATCH SIZE:     100000
IN_DB_UTIL BULK LIMIT:     NULL
IN_DB_UTIL PARALLEL:       1
IN_DB_UTIL LOGGING:        NULL
IN_DB_UTIL OS USER:        LOCAL_DOMAIN\USER_NAME
IN_DB_UTIL DB_SESSION:     742,52540

```

PL/SQL procedure successfully completed.

```
SQL> EXEC IN_DB_UTIL.IN_SESSION_CLEAR('IN_DB_UTIL','FUF9OPDVK8');
```

PL/SQL procedure successfully completed.

```
SQL> EXEC IN_DB_UTIL.IN_CONTEXT_PRINT('IN_DB_UTIL');
```

```

Current DB Time: 2022-08-04 15:21:53.541000
Current UTC Time: 2022-08-04 20:21:53.541000

```

Current Activity for the IN\_DB\_UTIL identifier:

```

IN_DB_UTIL TAG:            NULL
IN_DB_UTIL STATUS:        IDLE
IN_DB_UTIL OPERATION:     NULL
IN_DB_UTIL CALLED BY:     NULL
IN_DB_UTIL START TIME:    NULL
IN_DB_UTIL MAX TIME:      NULL
IN_DB_UTIL MAX MINUTES:   NULL
IN_DB_UTIL OBJECT:        NULL
IN_DB_UTIL SUBOBJECT:     NULL
IN_DB_UTIL TABLESPACE:   NULL
IN_DB_UTIL BATCH SIZE:    NULL
IN_DB_UTIL BULK LIMIT:    NULL
IN_DB_UTIL PARALLEL:      NULL
IN_DB_UTIL LOGGING:       NULL
IN_DB_UTIL OS USER:       NULL
IN_DB_UTIL DB_SESSION:    NULL

```

PL/SQL procedure successfully completed.

## IN\_SESSION\_CLEAR\_ALL

The `IN_SESSION_CLEAR_ALL` stored procedure can be used to unconditionally reset the entire call stack and session attributes for the specified identifier(s) of the `IN_CONTEXT` namespace.

Under normal circumstances, the `IN_SESSION_CLEAR_ALL` procedure should never need to be executed. However, it can be used to manually clear the entire call stack and session attributes for the specified identifier if necessary due to an unhandled exception or after manually cancelling an operation before completion.

**Important** Do not manually execute this procedure 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	
IDENTIFIER	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	ALL
	Description:	The identifier for which to clear the all session attributes.
	Options:	ALL (SYNC and UPGRADE) SYNC UPGRADE IN_DB_UTIL

**Exceptions**

- [FAILED\\_CHECK – INVALID\\_IDENTIFIER](#)
- [OTHERS](#)

**Examples**

The following example demonstrates how to use this procedure to manually clear the entire call stack and session attributes for both the SYNC and UPGRADE identifiers of the IN\_CONTEXT namespace.

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

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL('ALL');
```

**Example output**

The following example demonstrates the clearing of the UPGRADE identifier by executing the IN\_SESSION\_CLEAR\_ALL procedure to clear the call stack and session attributes after cancelling the IN\_DB\_UPGRADE.UPTIME\_STEPS procedure prior to completion.

```
SQL> EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT;

***** Call Stack Begin *****

The SYNC CALLSTACK is empty.

***** UPGRADE Call Stack *****

Depth  Tag          Procedure
-----  -
1      72S0FD50LL  UPTIME_STEPS
2      72S0FD50LL  SCHEMA_UPGRADE_7403_7500
3      72S0FD50LL  UPTIME_STEPS_IN_DOC
4      72S0FD50LL  CREATE_INDEXES_IN_DOC

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

PL/SQL procedure successfully completed.
```

```

SQL> EXEC IN_DB_UTIL.IN_CONTEXT_PRINT;

Current DB Time: 2022-08-05 08:02:43.386000
Current UTC Time: 2022-08-05 13:02:43.386000

Current Activity for the SYNC identifier:

SYNC TAG:                NULL
SYNC STATUS:             IDLE
SYNC OPERATION:          NULL
SYNC CALLED BY:          NULL
SYNC START TIME:         NULL
SYNC MAX TIME:           NULL
SYNC MAX MINUTES:        NULL
SYNC OBJECT:             NULL
SYNC SUBOBJECT:          NULL
SYNC TABLESPACE:        NULL
SYNC BATCH SIZE:         NULL
SYNC BULK LIMIT:         NULL
SYNC PARALLEL:           NULL
SYNC LOGGING:            NULL
SYNC OS USER:            NULL
SYNC DB_SESSION:         NULL

Current Activity for the UPGRADE identifier:

UPGRADE TAG:             72S0FD50LL
UPGRADE STATUS:          EXECUTING
UPGRADE OPERATION:       CREATE_INDEXES_IN_DOC
UPGRADE CALLED BY:       UPTIME_STEPS_IN_DOC
UPGRADE START TIME:      2022-08-05 08:02:43.386000
UPGRADE MAX TIME:        2022-08-05 09:02:42.975000
UPGRADE MAX MINUTES:     60
UPGRADE OBJECT:          NEW_DOC
UPGRADE SUBOBJECT:       NEW_DOC_IDX83
UPGRADE TABLESPACE:     INDX
UPGRADE PARALLEL:        4
UPGRADE LOGGING:         LOGGING
UPGRADE SCHEMA TARGET:   7.7.0.0
UPGRADE SCHEMA VERSION: 7.1.3.3
UPGRADE IS SYNC UPGRADE: YES
UPGRADE OS USER:         LOCAL_DOMAIN\USER_NAME
UPGRADE DB_SESSION:      255

PL/SQL procedure successfully completed.

SQL> EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL('UPGRADE');

PL/SQL procedure successfully completed.

SQL> EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT;

***** Call Stack Begin *****

The SYNC CALLSTACK is empty.

```

The UPGRADE CALLSTACK is empty.

\*\*\*\*\* Call Stack End \*\*\*\*\*

PL/SQL procedure successfully completed.

SQL> EXEC IN\_DB\_UTIL.IN\_CONTEXT\_PRINT;

Current DB Time: 2022-08-05 08:07:46.745000  
Current UTC Time: 2022-08-05 13:07:46.745000

Current Activity for the SYNC identifier:

SYNC TAG:	NULL
SYNC STATUS:	IDLE
SYNC OPERATION:	NULL
SYNC CALLED BY:	NULL
SYNC START TIME:	NULL
SYNC MAX TIME:	NULL
SYNC MAX MINUTES:	NULL
SYNC OBJECT:	NULL
SYNC SUBOBJECT:	NULL
SYNC TABLESPACE:	NULL
SYNC BATCH SIZE:	NULL
SYNC BULK LIMIT:	NULL
SYNC PARALLEL:	NULL
SYNC LOGGING:	NULL
SYNC OS USER:	NULL
SYNC DB_SESSION:	NULL

Current Activity for the UPGRADE identifier:

UPGRADE TAG:	NULL
UPGRADE STATUS:	IDLE
UPGRADE OPERATION:	NULL
UPGRADE CALLED BY:	NULL
UPGRADE START TIME:	NULL
UPGRADE MAX TIME:	NULL
UPGRADE MAX MINUTES:	NULL
UPGRADE OBJECT:	NULL
UPGRADE SUBOBJECT:	NULL
UPGRADE TABLESPACE:	NULL
UPGRADE PARALLEL:	NULL
UPGRADE LOGGING:	NULL
UPGRADE SCHEMA TARGET:	NULL
UPGRADE SCHEMA VERSION:	7.1.3.3
UPGRADE IS SYNC UPGRADE:	NULL
UPGRADE OS USER:	NULL
UPGRADE DB_SESSION:	NULL

PL/SQL procedure successfully completed.

## IN\_LOGGER

The IN\_LOGGER stored procedure is used to log certain actions to the [IN\\_DB\\_UTIL\\_LOG](#) table.

If the IN\_DB\_UTIL\_LOG table does not exist then the IN\_LOGGER stored procedure will attempt to create it.

Logged actions include recording calls to the following stored procedures:

- IN\_SESSION\_SET
- IN\_SESSION\_CLEAR
- IN\_SESSION\_CLEAR\_ALL
- CLEANUP\_AUDIT\_DATA
- REBUILD\_INDEXES

Logged actions also include recording any exceptions (errors) that occur within the IN\_DB\_UTIL and IN\_DB\_UPGRADE packages.

Parameter	Description	
PROC_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the procedure or function that is logging the action.
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default Value:</b>	NULL
	<b>Description:</b>	The identifier of the procedure or function that is logging the action.
ERROR_MSG	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(4000)
	<b>Default Value:</b>	NULL
SQL_CMD	<b>Description:</b>	The error message that is being logged.
	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(4000)
	<b>Default Value:</b>	NULL

Parameter	Description	
	<b>Description:</b>	The SQL statement being logged.

**Exceptions**

- [FAILED\\_CHECK](#)
- [OTHERS](#)

**Example**

```
IN_DB_UTIL.IN_LOGGER(V_PROC_NAME, V_IDENTIFIER, V_ERROR_MSG, V_EXEC_SQL);
```

## CLEAR\_VARIABLES

The CLEAR\_VARIABLES stored procedure is used to clear the values of certain package level variables. This procedure is called by the [IN\\_SESSION\\_SET](#) and [IN\\_SESSION\\_CLEAR](#) and [IN\\_SESSION\\_CLEAR\\_ALL](#) stored procedures during session initialization and teardown.

## EXECUTE\_DML

The EXECUTE\_DML stored procedure can be used to pass any valid, simple DML statement for immediate execution and committal with optional feedback on the affected row count and display of the SQL statement being executed.

When the verbose option is enabled (VERBOSE=YES) then the [PRINT\\_SQL](#) procedure is used to display the SQL (DML\_COMMAND) being executed.

Parameter	Description	
DML_SCHEMA	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The name of the schema containing the table to execute the DML against.
DML_TABLE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the table to execute the DML against.
	<b>Options:</b>	Any table for which the user has insert, update, and delete privileges.
DML_COMMAND	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(4000)

Parameter	Description	
	<b>Default value:</b>	None
	<b>Description:</b>	The DML statement to execute.
	<b>Options:</b>	Must be a basic INSERT, UPDATE, DELETE statement.
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	ALL
	<b>Description:</b>	The identifier associated with the IN_CONTEXT context namespace.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE ALL
VERBOSE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	YES
	<b>Description:</b>	Indicates if the SQL statement and number of affected rows should be displayed.
	<b>Options:</b>	YES NO

### Exceptions

- [FAILED\\_CHECK](#)
- [OTHERS](#)

### Example

```

SET SERVEROUTPUT ON;

DECLARE
  V_IDENTIFIER  VARCHAR2(128);
  V_EXEC_SQL    VARCHAR2(4000);
BEGIN
  V_IDENTIFIER := 'UPGRADE';
  V_EXEC_SQL   := 'INSERT INTO INUSER.IN_PRODUCT_MOD_HIST VALUES (''2100988013_1728017900'',
''7.9.0.0'', SYS_EXTRACT_UTC(SYSTIMESTAMP), ''UTC'')';
  IN_DB_UTIL.EXECUTE_DML('INUSER', 'IN_PRODUCT_MOD_HIST', V_EXEC_SQL, V_IDENTIFIER, 'YES');
END;
/

INSERT INTO INUSER.IN_PRODUCT_MOD_HIST VALUES ('2100988013_1728017900', '7.9.0.0',
SYS_EXTRACT_UTC(SYSTIMESTAMP), 'UTC');

```

```
Inserted 1 row into the INUSER.IN_PRODUCT_MOD_HIST table.
```

```
PL/SQL procedure successfully completed.
```

## PRINT\_SQL

The PRINT\_SQL stored procedure is used to format SQL for being displayed. The procedure allows you to customize the string length and formatting options to help produce a more readable output for SQL statements compared to simply using DBMS\_OUTPUT.PUT\_LINE.

Parameter	Description	
V_STRING	<b>Type:</b>	INPUT
	<b>Datatype:</b>	CLOB
	<b>Default value:</b>	None
	<b>Description:</b>	The SQL to be formatted and displayed.
V_LENGTH	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	100
	<b>Description:</b>	The maximum length to allow for each string during formatting.
V_FORMAT	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(7)
	<b>Default value:</b>	SIMPLE
	<b>Description:</b>	The formatting option to use.
	<b>Options:</b>	SIMPLE EXTRA MESSAGE

### Examples

```
EXEC IN_DB_UTIL.PRINT_SQL(V_EXEC_SQL);
EXEC IN_DB_UTIL.PRINT_SQL(V_EXEC_SQL, 150);
EXEC IN_DB_UTIL.PRINT_SQL(V_EXEC_SQL, 75, 'EXTRA');
```

### Example output

```
DECLARE
  V_EXEC_SQL CLOB;
BEGIN
  V_EXEC_SQL := 'SELECT PRODUCT_MOD_ID, PRODUCT_VERSION, MOD_TIME, PRODUCT_MOD_NOTES FROM
INUSER.IN_PRODUCT_MOD_HIST ORDER BY MOD_TIME DESC';
```

```

    IN_DB_UTIL.PRINT_SQL(V_EXEC_SQL,75,'EXTRA');
END;
/

SELECT PRODUCT_MOD_ID, PRODUCT_VERSION, MOD_TIME, PRODUCT_MOD_NOTES
FROM INUSER.IN_PRODUCT_MOD_HIST
ORDER BY MOD_TIME DESC;

PL/SQL procedure successfully completed.

```

## CHECK\_PARAMETER

The CHECK\_PARAMETER stored procedure 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
  - INDEX
  - PACKAGE
  - SEQUENCE
  - TABLE
  - TRIGGER
- List of supported database objects that do not require an owner and use the format of OBJECT\_NAME
  - SCHEMA
  - TABLESPACE
- List of supported IN\_DB\_UTIL framework parameters
  - BATCH\_SIZE
  - BULK\_LIMIT
  - IDENTIFIER
  - LOGGING
  - MAX\_MINUTES
  - PARALLEL
  - SYNC\_UPGRADE
  - VALIDATE\_FK
  - VERBOSE

Parameter	Description	
CHECK_TYPE	Type:	INPUT

Parameter	Description	
	<b>Datatype:</b>	VARCHAR2(30)
	<b>Default value:</b>	NULL
	<b>Description:</b>	Specifies the parameter type being checked.
	<b>Options:</b>	See the list of supported parameters above.
CHECK_VALUE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(257)
	<b>Default value:</b>	NULL
	<b>Description:</b>	Specifies the value of the parameter type being checked.
	<b>Options:</b>	Certain parameters are restricted to specific values.
CHECK_STATUS	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	NULL
	<b>Description:</b>	Returned status regarding the parameter check results. The returned status depends on the type of object that is checked. For database objects it returns the count of objects found.
	<b>Options:</b>	0 = Object does not exist 1 = Object exists 2 = 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) Invalid conditions for any of the parameter checks -1 = Invalid check which could be any of the exceptions listed in the Exceptions section below. Refer to the CHECK_MESSAGE output for details.
CHECK_MESSAGE	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(4000)
	<b>Default value:</b>	NULL
	<b>Description:</b>	Returned message regarding the parameter check results.

## Exceptions

- INVALID\_CHECK\_TYPE – The specified CHECK\_TYPE is unsupported.
- INVALID\_OBJECT – Either an improper value or no value was specified for CHECK\_VALUE.
- INVALID\_FQN – The CHECK\_TYPE requires a fully qualified name but was not provided.
- INVALID\_NUMBER – An invalid value was specified for a parameter expecting a numeric value.
- [INVALID\\_IDENTIFIER](#)
- INVALID\_SQL\_NAME – An invalid qualified SQL name was provided for the object type.
- [OTHERS](#)

## Examples

```
IN_DB_UTIL.CHECK_PARAMETER('TABLE', 'INUSER.IN_DOC', V_CHECK_STATUS, V_CHECK_MESSAGE);
IN_DB_UTIL.CHECK_PARAMETER('CONSTRAINT', 'INUSER.PK_DOC', V_CHECK_STATUS, V_CHECK_MESSAGE);
IN_DB_UTIL.CHECK_PARAMETER('INDEX', 'INUSER.DOC_IDX2', V_CHECK_STATUS, V_CHECK_MESSAGE);
IN_DB_UTIL.CHECK_PARAMETER('TABLESPACE', 'DATA', V_CHECK_STATUS, V_CHECK_MESSAGE);
```

## Example output

```
SET SERVEROUTPUT ON;

DECLARE
V_CHECK_TYPE      VARCHAR2(30); -- Parameter Type
V_CHECK_VALUE     VARCHAR2(257); -- Parameter Value
V_CHECK_STATUS    NUMBER;
V_CHECK_MESSAGE   VARCHAR2(500);

BEGIN
  V_CHECK_TYPE := 'INDEX';
  V_CHECK_VALUE := 'inuser.DOC_IDX2';

  IN_DB_UTIL.CHECK_PARAMETER
  (
    V_CHECK_TYPE, V_CHECK_VALUE, V_CHECK_STATUS, V_CHECK_MESSAGE
  );

  DBMS_OUTPUT.PUT_LINE('CHECK_STATUS = '||V_CHECK_STATUS);
  DBMS_OUTPUT.PUT_LINE('CHECK_MESSAGE = '||V_CHECK_MESSAGE);
END;
/

CHECK_STATUS = 1
CHECK_MESSAGE = The INUSER.DOC_IDX2 index exists.
```

## CHECK\_TIME

The CHECK\_TIME function is used to check the MAX\_TIME attribute of the IN\_CONTEXT namespace for the specified identifier to determine if MAX\_MINUTES has expired when programmatically deciding if the next qualifying operation should be started or not.

This only applies if the MAX\_MINUTES session attribute has been set to a value greater than zero using the [IN\\_CONTEXT\\_SET](#) procedure.

This functionality is currently used by the following stored procedures:

- IN\_DB\_UTIL.CLEANUP\_AUDIT\_DATA

- IN\_DB\_UTIL.REBUILD\_INDEXES
- IN\_DB\_UPGRADE.UPTIME\_STEPS
- IN\_DB\_UPGRADE.SYNC\_TABLE\_IN\_DOC

Parameter	Description	
IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	Specifies the identifier for which to check the MAX_TIME session attribute.
	<b>Options:</b>	IN_DB_UTIL SYNC UPGRADE
V_RESULTS	<b>Type:</b>	RETURN
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	NULL
	<b>Description:</b>	Specifies if the MAX_TIME session parameter has elapsed.
	<b>Options:</b>	0 = MAX_TIME has not yet elapsed or is undefined. 1 = MAX_TIME has elapsed.

### Examples

```
SELECT IN_DB_UTIL.CHECK_TIME(V_IDENTIFIER) FROM DUAL;
SELECT IN_DB_UTIL.CHECK_TIME('IN_DB_UTIL') AS CHECK_TIME FROM DUAL;
SELECT IN_DB_UTIL.CHECK_TIME('SYNC') AS CHECK TIME FROM DUAL;
SELECT IN_DB_UTIL.CHECK_TIME('UPGRADE') AS CHECK_TIME FROM DUAL;
```

## GET\_IN\_DB\_UTIL\_VERSION

The GET\_IN\_DB\_UTIL\_VERSION function returns the current version number for the IN\_DB\_UTIL package.

Parameter	Description	
PKG_VERSION	<b>Type:</b>	RETURN
	<b>Datatype:</b>	VARCHAR2(16)
	<b>Default value:</b>	4.2.0
	<b>Description:</b>	Returns the current version of the IN_DB_UTIL package.

### Examples

```
SELECT IN_DB_UTIL.GET_IN_DB_UTIL_VERSION FROM DUAL;

GET_IN_DB_UTIL_VERSION
-----
4.2.0
```

```
SET SERVEROUTPUT ON;

DECLARE V_INDBUTIL_VERSION VARCHAR2(16) := IN_DB_UTIL.GET_IN_DB_UTIL_VERSION;
BEGIN
  DBMS_OUTPUT.PUT_LINE(CHR(10) || V_INDBUTIL_VERSION);
END;
/

4.2.0
```

## 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.

### GET\_IN\_DB\_SCHEMA\_VERSION

The GET\_IN\_DB\_SCHEMA\_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 SCHEMA\_BASE parameter returned by the [PARSE\\_SCHEMA\\_VERSION](#) stored procedure instead.

Parameter	Description	
SCHEMA_VERSION	<b>Type:</b>	RETURN
	<b>Datatype:</b>	VARCHAR2(40)
	<b>Default value:</b>	None
	<b>Description:</b>	Returns the full schema version.

### Examples

```
SELECT IN_DB_UTIL.GET_IN_DB_SCHEMA_VERSION FROM DUAL;

GET_IN_DB_SCHEMA_VERSION
-----
7.1.3.3
```

```
SET SERVEROUTPUT ON;

DECLARE V_INOW_VERSION VARCHAR2(16);
```

```

BEGIN
  V_INOW_VERSION := IN_DB_UTIL.GET_IN_DB_SCHEMA_VERSION;
  DBMS_OUTPUT.PUT_LINE(CHR(10) || V_INOW_VERSION);
END;
/
7.1.3.3

```

## PARSE\_SCHEMA\_VERSION

The PARSE\_SCHEMA\_VERSION stored procedure will parse the current INUSER schema version and return schema specific attributes to populate variables with for conditional processing or for dynamic SQL generation.

The PARSE\_SCHEMA\_VERSION procedure will return the current INUSER base schema version (without the type), and the schema type, and the default character datatype.

For example, if the current INUSER schema version is 7.7.0.0a then the procedure will return the following values:

- SCHEMA\_BASE will be the base schema version of '7.7.0.0'.
- SCHEMA\_TYPE will be the schema type of 'a', signifying Advanced Filegroup schema configuration.
- CHAR\_DATATYPE will be the default character datatype for the schema of 'VARCHAR2'.

Parameter	Description	
IN_SCHEMA	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(40)
	<b>Default value:</b>	GET_IN_DB_SCHEMA_VERSION
	<b>Description:</b>	The schema version number to parse. If NULL then the current schema version will be fetched using the GET_IN_DB_SCHEMA_VERSION function.
SCHEMA_BASE	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(40)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The base schema version without the schema type designation.
SCHEMA_TYPE	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(2)
	<b>Default value:</b>	NULL
	<b>Description:</b>	Indicates the schema type.

Parameter	Description	
	<b>Options:</b>	Empty string for standard, ANSI schema type a = Advanced Filegroup Configuration
CHAR_DATATYPE	<b>Type:</b>	OUTPUT
	<b>Datatype:</b>	VARCHAR2(9)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The default character datatype for the schema. Unicode is not currently supported on Oracle so this will always be VARCHAR2.

### Example

```

SET SERVEROUTPUT ON;

DECLARE
V_SCHEMA_BASE  VARCHAR2(40);
V_SCHEMA_TYPE  VARCHAR2(2);
V_CHAR_DATATYPE VARCHAR2(9);
BEGIN
  IN_DB_UTIL.PARSE_SCHEMA_VERSION(IN_DB_UTIL.GET_IN_DB_SCHEMA_VERSION, V_SCHEMA_BASE,
V_SCHEMA_TYPE, V_CHAR_DATATYPE);

  DBMS_OUTPUT.PUT_LINE('SCHEMA_BASE = '||V_SCHEMA_BASE);
  DBMS_OUTPUT.PUT_LINE('SCHEMA_TYPE = '||V_SCHEMA_TYPE);
  DBMS_OUTPUT.PUT_LINE('CHAR_DATATYPE = '||V_CHAR_DATATYPE);
END;
/
    
```

### Example Output

```

SCHEMA_BASE = 7.7.0.0
SCHEMA_TYPE =
CHAR_DATATYPE = VARCHAR2
    
```

## PRINT\_SCHEMA\_HIST

The PRINT\_SCHEMA\_HIST stored procedure will fetch and display the INUSER schema history from the IN\_PRODUCT\_MOD\_HIST table.

Parameter	Description	
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The name of the schema. This should always be INUSER.

### Exceptions

- [FAILED\\_CHECK](#)
- [OTHERS](#)

**Example**

```
SET SERVEROUTPUT ON;
EXEC IN_DB_UTIL.PRINT_SCHEMA_HIST;
```

## PRINT\_INDEX\_DDL

The PRINT\_INDEX\_DDL is used to generate the DDL for a single index or all the indexes for the specified table.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER
	Description:	The schema name that owns the table.
TABLE_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The table name for which to generate the index DDL.
INDEX_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The index name for which to generate the DDL.

**Exceptions**

- [FAILED\\_CHECK](#)
- [INVALID\\_OBJECT](#)
- [OTHERS](#)

**Examples**

```
SET SERVEROUTPUT ON;
EXEC IN_DB_UTIL.PRINT_INDEX_DDL('INUSER', 'IN_DOC');
EXEC IN_DB_UTIL.PRINT_INDEX_DDL('INUSER', 'IN_DOC', 'DOC_IDX2');
```

**Example Output**

```
CREATE UNIQUE INDEX "INUSER"."DOC_IDX2" ON "INUSER"."IN_DOC"
```

```
(
NLSSORT("DRAWER_ID", 'nls_sort=' 'BINARY_CI''),
NLSSORT("FOLDER", 'nls_sort=' 'BINARY_CI''),
NLSSORT("TAB", 'nls_sort=' 'BINARY_CI''),
NLSSORT("F3", 'nls_sort=' 'BINARY_CI''),
NLSSORT("F4", 'nls_sort=' 'BINARY_CI''),
NLSSORT("F5", 'nls_sort=' 'BINARY_CI''),
NLSSORT("DOC_TYPE_ID", 'nls_sort=' 'BINARY_CI''),
NLSSORT("LEGACY_UNIQUENESS", 'nls_sort=' 'BINARY_CI'')
)
PCTFREE          10
INITRANS         2
MAXTRANS         255
COMPUTE STATISTICS
STORAGE
(
INITIAL          65536
NEXT             65536
MINEXTENTS       1
MAXEXTENTS       2147483645
PCTINCREASE      0
FREELISTS        1
FREELIST GROUPS 1
BUFFER_POOL      DEFAULT
FLASH_CACHE      DEFAULT
CELL_FLASH_CACH DEFAULT
)
TABLESPACE "INDX";
```

## PRINT\_FK

The PRINT\_FK stored procedure is used to display all the foreign keys to and from the specified table.

Parameter	Description	
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The schema name that owns the table with the foreign keys being displayed.
TABLE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The name of the table with the foreign keys being displayed.

### Exceptions

- [FAILED\\_CHECK](#)
- [OTHERS](#)

### Example

```
SET SERVEROUTPUT ON;
```

```
EXEC IN_DB_UTIL.PRINT_FK('INUSER','IN_DOC');
```

```
-- EXAMPLE OUTPUT
```

```
Referential Integrity Constraints for the INUSER.IN_DOC table:
```

```
INUSER.IN_DOC References:
```

```
Foreign Key INUSER.FK_D_DOC_TYPE_ID References PK_DOC_TYPE is ENABLED and VALIDATED
```

```
Foreign Key INUSER.FK_DOC_DRAWER_ID References PK_DRAWER is ENABLED and VALIDATED
```

```
Foreign Key INUSER.FK_DOC_INSTANCE_ID References PK_INSTANCE is ENABLED and VALIDATED
```

```
INUSER.IN_DOC Referenced By:
```

```
Table INUSER.IN_DOC_CONTENT Foreign Key FK_DC_DOC_ID References PK_DOC is ENABLED and VALIDATED
```

```
Table INUSER.IN_DOC_KW Foreign Key FK_DK_DOC_ID References PK_DOC is ENABLED and VALIDATED
```

```
Table INUSER.IN_RC_RECORD_DOC Foreign Key FK_RCRD_D_ID References PK_DOC is ENABLED and VALIDATED
```

```
Table INUSER.IN_SIG Foreign Key FK_S_DOC_ID References PK_DOC is ENABLED and VALIDATED
```

```
Table INUSER.IN_TASK Foreign Key FK_T_DOC_ID References PK_DOC is ENABLED and VALIDATED
```

```
Table INUSER.IN_VERSION Foreign Key FK_V_DOC_ID References PK_DOC is ENABLED and VALIDATED
```

```
PL/SQL procedure successfully completed.
```

## GET\_FK\_DDL

The GET\_FK\_DDL stored procedure is used to generate the DDL for a specific foreign key constraint that references the specified table name.

Parameter	Description	
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The schema name that owns the table that is referenced by the foreign key.
FK_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the FK that references the PK or Unique index on the specified table.

Parameter	Description	
FK_DDL	Type:	RETURN.
	Datatype:	VARCHAR2(2000)
	Default value:	None
	Description:	The generated DDL for the FK(s) that reference the specified table

### Exceptions

- [FAILED\\_CHECK](#)
- [OTHERS](#)

### Example

```
SET SERVEROUTPUT ON;

DECLARE
  v_fk_ddl VARCHAR2(2000);
BEGIN
  FOR fk IN
  (
    SELECT CONSTRAINT_NAME FROM USER_CONSTRAINTS
    WHERE TABLE_NAME = 'IN_DOC' AND CONSTRAINT_TYPE = 'R'
    ORDER BY CONSTRAINT_NAME
  )
  LOOP
    IN_DB_UTIL.GET_FK_DDL('INUSER', fk.CONSTRAINT_NAME, v_fk_ddl);
    IN_DB_UTIL.PRINT_SQL(v_fk_ddl);
  END LOOP;
END;
/
```

### Example Output

```
ALTER TABLE "INUSER"."IN_DOC" ADD CONSTRAINT "FK_D_DOC_TYPE_ID" FOREIGN KEY ("DOC_TYPE_ID")
REFERENCES "INUSER"."IN_DOC_TYPE" ("DOC_TYPE_ID") ENABLE;

ALTER TABLE "INUSER"."IN_DOC" ADD CONSTRAINT "FK_DOC_DRAWER_ID" FOREIGN KEY ("DRAWER_ID")
REFERENCES "INUSER"."IN_DRAWER" ("DRAWER_ID") ENABLE;

ALTER TABLE "INUSER"."IN_DOC" ADD CONSTRAINT "FK_DOC_INSTANCE_ID" FOREIGN KEY ("INSTANCE_ID")
REFERENCES "INUSER"."IN_INSTANCE" ("INSTANCE_ID") ENABLE;

PL/SQL procedure successfully completed.
```

## Schema change stored procedures

The following collection of stored procedures are used to perform various database schema change operations. These are commonly used by the IN\_DB\_UPGRADE package during database schema upgrades.

## DROP\_INDEX

The DROP\_INDEX stored procedure is used to drop a single index or all indexes on the specified table except for the Primary Key (PK) index.

The Primary Key Case Insensitive (PKCI) function-based index will only be dropped if provided as input using the INDEX\_NAME parameter to specifically drop the PKCI index. If only the table name is specified, then all indexes on the table will be dropped except the PK and PKCI indexes.

If an index name is provided, then only the specified index will be dropped.

The DROP\_INDEX procedure will set the DDL\_LOCK\_TIMEOUT parameter to 60 seconds so will wait up to one minute for a DDL lock to drop an index. Note that this could result in longer waits when trying to drop an index that is actively being used.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER
	Description:	The schema name that owns the table with the index(es) being dropped.
TABLE_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The name of the table with the index(es) being dropped.
INDEX_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The name of the index being dropped.

### Exceptions

- [FAILED\\_CHECK](#)
- RESOURCE\_BUSY – Could not drop index due to resource lock wait. ORA-00054.
- PK\_INDEX – Cannot drop index used to enforce primary key constraint. ORA- 02429.
- [OTHERS](#)

### Example

```
EXEC IN_DB_UTIL.DROP_INDEX('INUSER', TABLE_NAME, [INDEX_NAME]);
```

### Example Output

```
SQL> EXEC IN_DB_UTIL.DROP_INDEX('INUSER', 'IN_AUDIT', 'AUDIT_IDX_01');
DROP INDEX INUSER.AUDIT_IDX_01
PL/SQL procedure successfully completed.
```

## DROP\_PK

The DROP\_PK stored procedure is used to drop the primary key from a single table.

If the primary key index is currently being referenced by any foreign key constraints, preventing the constraint from being dropped, then the PK\_REFERENCED exception will be raised and the foreign keys that reference the primary key constraint will be listed.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER
	Description:	The schema name that owns the table with the primary key being dropped.
TABLE_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The name of the table with the primary key being dropped.

### Exceptions

- [FAILED\\_CHECK](#)
- NO\_DATA\_FOUND – The specified table does not have a primary key.
- PK\_REFERENCED – The primary key is currently referenced by foreign keys.
- [OTHERS](#)

### Example

```
EXEC IN_DB_UTIL.DROP_PK('INUSER', 'IN_AUDIT');
```

### Example output

```
EXEC IN_DB_UTIL.DROP_PK('INUSER', 'IN_AUDIT');
ALTER TABLE INUSER.IN_AUDIT DROP PRIMARY KEY DROP INDEX
The specified primary key constraint is referenced by the following foreign keys:
Table: INUSER.IN_AUDIT_DETAIL - FK constraint: FK_AD_A_ID
Table: INUSER.IN_AUDIT_OBJ - FK constraint: FK_AO_A_ID
PL/SQL procedure successfully completed.
```

## DROP\_FK

The DROP\_FK stored procedure is used to drop a single foreign key or all the foreign keys to and/or from a specified table.

The SCOPE parameter is used to specify the scope for which to drop the foreign keys as related to the specified table.

- FROM includes the foreign keys on the specified table that reference any other table.
- TO includes the foreign keys from any other table that reference the specified table.
- ALL includes all foreign keys to and from the specified table.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER
	Description:	The schema name that owns the table with the foreign key(s) being dropped.
TABLE_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The name of the table with the foreign key(s) being dropped.
SCOPE	Type:	INPUT
	Datatype:	VARCHAR2(4)
	Default value:	FROM
	Description:	The scope for which to drop the foreign keys as related to the specified table.
	Options:	FROM TO ALL
FK_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	NULL

Parameter	Description	
	<b>Description:</b>	The name of the foreign key being dropped if only dropping one foreign key.

**Exceptions**

- [FAILED\\_CHECK](#)
- INVALID\_FK – The specified foreign key does not belong to the specified table.
- [OTHERS](#)

**Examples**

```
EXEC IN_DB_UTIL.DROP_FK('INUSER', TABLE_NAME, [ALL|TO|FROM], [FK_NAME]);
```

```
SET SERVEROUTPUT ON;
EXEC IN_DB_UTIL.DROP_FK('INUSER', 'IN_AUDIT', 'ALL');
```

**Example Output**

```
SQL> EXEC IN_DB_UTIL.DROP_FK('INUSER', 'IN_AUDIT', 'ALL');
ALTER TABLE INUSER.IN_AUDIT_DETAIL DROP CONSTRAINT FK_AD_A_ID
ALTER TABLE INUSER.IN_AUDIT_OBJ DROP CONSTRAINT FK_AO_A_ID
PL/SQL procedure successfully completed.
```

## DISABLE\_FK

The DISABLE\_FK stored procedure is used to disable a single foreign key or all the foreign keys to and/or from a single table.

The SCOPE parameter is used to specify the scope for which to disable the foreign keys as related to the specified table.

- FROM includes the foreign keys on the specified table that reference any other table.
- TO includes the foreign keys from any other table that reference the specified table.
- ALL includes all foreign keys to and from the specified table.

Parameter	Description	
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The schema name that owns the table with the foreign key(s) being disabled.
TABLE_NAME	<b>Type:</b>	INPUT

Parameter	Description	
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the table with the foreign key(s) being disabled.
SCOPE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(4)
	<b>Default value:</b>	ALL
	<b>Description:</b>	The scope for which to disable the foreign keys as related to the specified table.
FK_NAME	<b>Options:</b>	FROM TO ALL
	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
VERBOSE	<b>Description:</b>	The name of the foreign key being dropped if only dropping one foreign key.
	<b>Type:</b>	INUSER
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	YES
	<b>Description:</b>	Is used to indicate if you want to display the status of the foreign keys before and after disabling.

**Exceptions**

- [FAILED\\_CHECK](#)
- [OTHERS](#)

**Examples**

```

SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_DOC', [ALL|TO|FROM], [FK_NAME], [YES|NO]);
EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_DOC', 'TO', NULL, 'YES');
EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_DOC', 'FROM', NULL, 'YES');
EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_DOC', 'ALL', NULL, 'NO');
EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_DOC', NULL, 'FK_D_DOC_TYPE_ID', 'NO');
    
```

### Example Output

```
SQL> EXEC IN_DB_UTIL.DISABLE_FK('INUSER', 'IN_AUDIT', 'ALL', NULL, 'YES');

Referential Integrity Constraints for the INUSER.IN_AUDIT table:

INUSER.IN_AUDIT References:

INUSER.IN_AUDIT Referenced By:
Table INUSER.IN_AUDIT_DETAIL Foreign Key FK_AD_A_ID References PK_AUDIT is ENABLED and VALIDATED
Table INUSER.IN_AUDIT_OBJ Foreign Key FK_AO_A_ID References PK_AUDIT is ENABLED and VALIDATED

-----
Disable all qualifying foreign key constraints
-----

ALTER TABLE INUSER.IN_AUDIT_DETAIL DISABLE CONSTRAINT FK_AD_A_ID
ALTER TABLE INUSER.IN_AUDIT_OBJ DISABLE CONSTRAINT FK_AO_A_ID

-----

Referential Integrity Constraints for the INUSER.IN_AUDIT table:

INUSER.IN_AUDIT References:

INUSER.IN_AUDIT Referenced By:
Table INUSER.IN_AUDIT_DETAIL Foreign Key FK_AD_A_ID References PK_AUDIT is DISABLED and NOT VALIDATED
Table INUSER.IN_AUDIT_OBJ Foreign Key FK_AO_A_ID References PK_AUDIT is DISABLED and NOT VALIDATED

PL/SQL procedure successfully completed.
```

## ENABLE\_FK

The ENABLE\_FK stored procedure is used to enable and validate a single foreign key or all the foreign keys to and/or from a single table.

The SCOPE parameter is used to specify the scope for which to enable and validate the foreign keys as related to the specified table.

- FROM includes the foreign keys on the specified table that reference any other table.
- TO includes the foreign keys from any other table that reference the specified table.
- ALL includes all foreign keys to and from the specified table.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER

Parameter	Description	
	<b>Description:</b>	The schema name that owns the table with the foreign key(s) being enabled and validated.
TABLE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the table with the foreign key(s) being enabled validated.
SCOPE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(4)
	<b>Default value:</b>	ALL
	<b>Description:</b>	The scope for which to enable and validate the foreign keys as related to the specified table.
	<b>Options:</b>	FROM TO ALL
FK_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The name of the foreign key being enabled and validated if only enabling one foreign key.
VERBOSE	<b>Type:</b>	INUSER
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	YES
	<b>Description:</b>	Is used to indicate if you want to display the status of the foreign keys before and after being enabled and validated.
	<b>Options:</b>	YES NO

**Exceptions**

- [FAILED\\_CHECK](#)

- **OTHERS**

**Examples**

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_DOC', [ALL|TO|FROM], [FK_NAME], [YES|NO]);
EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_DOC', 'TO', NULL, 'YES');
EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_DOC', 'FROM', NULL, 'YES');
EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_DOC', 'ALL', NULL, 'NO');
EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_DOC', NULL, 'FK_D_DOC_TYPE_ID', 'NO');
```

**Example Output**

```
SQL> EXEC IN_DB_UTIL.ENABLE_FK('INUSER', 'IN_AUDIT', 'ALL', NULL, 'YES');

Referential Integrity Constraints for the INUSER.IN_AUDIT table:

INUSER.IN_AUDIT References:

INUSER.IN_AUDIT Referenced By:

Table INUSER.IN_AUDIT_DETAIL Foreign Key FK_AD_A_ID References PK_AUDIT is DISABLED and NOT VALIDATED
Table INUSER.IN_AUDIT_OBJ Foreign Key FK_AO_A_ID References PK_AUDIT is DISABLED and NOT VALIDATED

-----
Enable and validate all qualifying foreign key constraints
-----

ALTER TABLE INUSER.IN_AUDIT_DETAIL ENABLE CONSTRAINT FK_AD_A_ID
ALTER TABLE INUSER.IN_AUDIT_DETAIL MODIFY CONSTRAINT FK_AD_A_ID VALIDATE
ALTER TABLE INUSER.IN_AUDIT_OBJ ENABLE CONSTRAINT FK_AO_A_ID
ALTER TABLE INUSER.IN_AUDIT_OBJ MODIFY CONSTRAINT FK_AO_A_ID VALIDATE

-----

Referential Integrity Constraints for the INUSER.IN_AUDIT table:

INUSER.IN_AUDIT References:

INUSER.IN_AUDIT Referenced By:

Table INUSER.IN_AUDIT_DETAIL Foreign Key FK_AD_A_ID References PK_AUDIT is ENABLED and VALIDATED
Table INUSER.IN_AUDIT_OBJ Foreign Key FK_AO_A_ID References PK_AUDIT is ENABLED and VALIDATED

PL/SQL procedure successfully completed.
```

**RENAME\_OBJECT**

The RENAME\_OBJECT stored procedure can be used to rename a table or index. This procedure will verify the availability of the new object name before attempting to rename the object.

Parameter	Description	
OBJECT_TYPE	Type:	INPUT

Parameter	Description	
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The type of object being renamed.
	<b>Options:</b>	TABLE INDEX
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The schema name that owns the object being renamed.
OBJECT_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The current name of the object.
NEW_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The new name for the object.

### Exceptions

- [FAILED\\_CHECK](#)
- [INVALID\\_OBJECT\\_TYPE](#)
- [INVALID\\_OBJECT](#)
- [NAME\\_UNAVAILABLE](#)
- [OTHERS](#)

### Examples

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.RENAME_OBJECT([TABLE|INDEX], 'INUSER', [CURRENT_NAME], [NEW_NAME]);
EXEC IN_DB_UTIL.RENAME_OBJECT('INDEX', 'INUSER', 'AUDIT_IDX_01', 'NEW_INDEX_NAME');
```

```
SET SERVEROUTPUT ON;
```

```

DECLARE
  V_OBJECT_TYPE VARCHAR2(128);
  V_OBJECT_NAME VARCHAR2(128);
  V_NEW_NAME     VARCHAR2(128);
  -- Exceptions
  NAME_UNAVAILABLE      EXCEPTION; PRAGMA EXCEPTION_INIT(NAME_UNAVAILABLE, -20002);
  INVALID_OBJECT_TYPE   EXCEPTION; PRAGMA EXCEPTION_INIT(INVALID_OBJECT_TYPE,-20005);
  INVALID_OBJECT        EXCEPTION; PRAGMA EXCEPTION_INIT(INVALID_OBJECT, -20006);
BEGIN
  V_OBJECT_TYPE := 'INDEX';
  V_OBJECT_NAME := 'AUDIT_IDX_01';
  V_NEW_NAME    := 'NEW_INDEX_NAME';

  IN_DB_UTIL.RENAME_OBJECT(V_OBJECT_TYPE, 'INUSER', V_OBJECT_NAME, V_NEW_NAME);

  EXCEPTION
  WHEN INVALID_OBJECT_TYPE THEN NULL;
  WHEN INVALID_OBJECT       THEN NULL;
  WHEN NAME_UNAVAILABLE     THEN NULL;
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('There was an unhandled exception.');
```

### Example Output

```

SQL> EXEC IN_DB_UTIL.RENAME_OBJECT('INDEX','INUSER', 'AUDIT_IDX_01', 'NEW_INDEX_NAME');

ALTER INDEX INUSER.AUDIT_IDX_01 RENAME TO NEW_INDEX_NAME

The index has been renamed.

PL/SQL procedure successfully completed.
```

## RENAME\_CONSTRAINT

The RENAME\_CONSTRAINT stored procedure is used to rename a primary key or foreign key constraint.

Parameter	Description	
SCHEMA_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	INUSER
	<b>Description:</b>	The schema name that owns the table with the constraint being renamed.
TABLE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	None
	<b>Description:</b>	The name of the table with the constraint being renamed.

Parameter	Description	
CONSTRAINT_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	None
	Description:	The current name of the constraint being renamed.
NEW_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	NULL
	Description:	The new name for the constraint being renamed.

### Exceptions

- [FAILED\\_CHECK](#)
- [INVALID\\_OBJECT](#)
- [NAME\\_UNAVAILABLE](#)
- [OTHERS](#)

### Examples

```
EXEC IN_DB_UTIL.RENAME_CONSTRAINT('INUSER', [TABLE_NAME], [CURRENT_NAME], [NEW_NAME]);
```

```
SET SERVEROUTPUT ON;

DECLARE
  V_TABLE_NAME  VARCHAR2(128);
  V_CURRENT_NAME VARCHAR2(128);
  V_NEW_NAME    VARCHAR2(128);
  -- Exceptions
  NAME_UNAVAILABLE EXCEPTION; PRAGMA EXCEPTION_INIT(NAME_UNAVAILABLE, -20002);
  INVALID_OBJECT   EXCEPTION; PRAGMA EXCEPTION_INIT(INVALID_OBJECT, -20006);
BEGIN
  V_TABLE_NAME := 'IN_AUDIT_OBJ';
  V_CURRENT_NAME := 'FK_AUDITOBJECT_AUDIT_ID';
  V_NEW_NAME := 'FK_AO_A_ID';

  IN_DB_UTIL.RENAME_CONSTRAINT('INUSER', V_TABLE_NAME, V_CURRENT_NAME, V_NEW_NAME);

  EXCEPTION
  WHEN INVALID_OBJECT THEN NULL;
  WHEN NAME_UNAVAILABLE THEN NULL;
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('There was an unhandled exception.');
```

### Example Output

```
SQL> EXEC IN_DB_UTIL.RENAME_CONSTRAINT('INUSER', 'IN_AUDIT', 'PK_AUDIT', 'PK_NEW_NAME');
```

```
ALTER TABLE INUSER.IN_AUDIT RENAME CONSTRAINT PK_AUDIT TO PK_NEW_NAME

The constraint has been renamed.

PL/SQL procedure successfully completed.
```

## Database information procedures and functions

The following collection of stored procedures and functions are used to gather details about the database.

### GET\_DB\_INFO

The GET\_DB\_INFO stored procedure is used to display details about the database.

Gathered information includes the following:

- Instance Name
- Database Name
- Database Edition and Version
- Optimizer Related Parameters
- NLS\_DATABASE\_PARAMETERS
- NLS\_INSTANCE\_PARAMETERS
- NLS\_SESSION\_PARAMETERS

#### Exception

- [OTHERS](#)

#### Example

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.GET_DB_INFO;
```

### GET\_CPU\_COUNT

The GET\_CPU\_COUNT function is used to determine the maximum number of processors to use when executing an operation in parallel. The resulting count is used to set the degree of parallelism for a session when the [SET\\_PARALLEL](#) procedure is executed and 0 (maximum) is specified.

The resulting CPU\_COUNT is based on the CPU\_COUNT from V\$PARAMETER minus 1 for less than 8 CPU's or minus 2 for 8 or more CPU's.

#### Example

```
SELECT IN_DB_UTIL.GET_CPU_COUNT FROM DUAL;
```

### GET\_DB\_CONNECTIONS

The GET\_DB\_CONNECTIONS stored procedure is used to display a summary of the current database user connections.

#### Exception

- [OTHERS](#)

**Example**

```
SET SERVEROUTPUT ON;
EXEC IN_DB_UTIL.GET_DB_CONNECTIONS;
```

## GET\_DB\_CONNECTIONS\_COUNT

The GET\_DB\_CONNECTIONS\_COUNT function is used to get the current number of database user connections.

Parameter	Description	
V_CONNECTIONS	Type:	RETURN
	Datatype:	NUMBER
	Default value:	None
	Description:	The number of connections to the database excluding the current session.

**Example**

```
SELECT IN_DB_UTIL.GET_DB_CONNECTIONS_COUNT() AS NUMBER_OF_CONNECTIONS FROM DUAL;
```

## Session configuration stored procedures

The following stored procedures can be used to alter a database session.

### SET-NLS\_SESSION\_PARAMETERS

The SET-NLS\_SESSION\_PARAMETERS stored procedure is used to set the following standard Perceptive Content NLS and optimizer session parameters.

```
ALTER SESSION SET NLS_COMP = LINGUISTIC;
ALTER SESSION SET NLS_SORT = BINARY_CI;
ALTER SESSION SET NLS_TIMESTAMP_FORMAT = 'RRRR-MM-DD HH24:MI:SS.FF';
ALTER SESSION SET NLS_TIME_FORMAT = 'HH24:MI:SS.FF';
ALTER SESSION SET NLS_DATE_FORMAT = 'RRRR-MM-DD';
ALTER SESSION SET OPTIMIZER_MODE = 'ALL ROWS';
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = 'TRUSTED';
ALTER SESSION SET QUERY_REWRITE_ENABLED = 'TRUE';
ALTER SESSION SET CURSOR_SHARING = 'EXACT';
ALTER SESSION SET OPTIMIZER_INDEX_COST_ADJ = 100;
```

**Exception**

- [OTHERS](#)

**Example**

```
EXEC IN_DB_UTIL.SET-NLS_SESSION_PARAMETERS;
```

## SET\_DBA\_ROLE

The SET\_DBA\_ROLE stored procedure is used to set the DBA role for the current session if the invoker has DBA privileges already granted to them.

### Exceptions

- NOT\_DBA – Exception for ORA 1924 error.
- [OTHERS](#)

### Example

```
SET SERVEROUTPUT ON;
EXEC IN_DB_UTIL.SET_DBA_ROLE;
```

## SET\_PARALLEL

The SET\_PARALLEL stored procedure is used to force parallelism for DDL and DML and QUERY for the current session by executing the following alter session commands

```
ALTER SESSION FORCE PARALLEL QUERY PARALLEL V_PARALLEL;
ALTER SESSION FORCE PARALLEL DDL PARALLEL V_PARALLEL;
ALTER SESSION FORCE PARALLEL DML PARALLEL V_PARALLEL;
```

The default value of zero will use the [GET\\_CPU\\_COUNT](#) function to determine the maximum degree of parallelism to use. Default is CPU\_COUNT minus 1 for less than 8 CPU's or minus 2 for 8 or more CPU's.

Parameter	Description	
PARALLEL	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	0
	<b>Description:</b>	The degree of parallelism to force the session to use.
	<b>Options:</b>	0 = Maximum 1 = No Parallel (Disabled) N = Specified number of CPU's

### Exceptions

- NO\_ALTER\_PARALLEL – Exception for ORA -12841 errors.
- [OTHERS](#)

### Examples

```
EXEC IN_DB_UTIL.SET_PARALLEL(V_PARALLEL);
EXEC IN_DB_UTIL.SET_PARALLEL;
EXEC IN_DB_UTIL.SET_PARALLEL(1);
EXEC IN_DB_UTIL.SET_PARALLEL(6);
```

## DISABLE\_PARALLEL

The `DISABLE_PARALLEL` stored procedure is used to disable forced parallelism for DDL and DML and QUERY for the current session by executing the following alter session commands. This is equivalent to running `SET_PARALLEL(1)`.

```
ALTER SESSION DISABLE PARALLEL DDL;
ALTER SESSION DISABLE PARALLEL DML;
ALTER SESSION DISABLE PARALLEL QUERY;
```

### Exceptions

- `NO_ALTER_PARALLEL` – Exception for ORA 12841 error.
- `OTHERS`

### Example

```
EXEC IN_DB_UTIL.DISABLE_PARALLEL;
```

## TRACE\_SESSION

The `TRACE_SESSION` stored procedure can be used enable tracing for a database session.

Tracing for your database session can be enabled using the `ON` option and later disabled using the `OFF` option. The `REPORT` option will display commands necessary to run the Oracle `TKPROF` utility to format the trace file for readability.

**Note** The generated trace file will be created on the database server and the `tkprof` utility must be executed on that server using the command provided by the procedure when tracing was turned on.

Parameter	Description	
TRACING	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(10)
	<b>Default value:</b>	HELP
	<b>Description:</b>	The specific command to be executed with respect to the tracing of the session.
	<b>Options:</b>	HELP = Display usage details. ON = Turn on tracing. OFF = Turn off tracing. REPORT = Display commands to format the trace file into a readable format using the Oracle <code>TKPROF</code> utility.
TRACE_LEVEL	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	12
	<b>Description:</b>	The level of tracing to use for the session.

Parameter	Description	
	<b>Options:</b>	See Oracle documentation for all options.
TRACEFILE_IDENTIFIER	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	USERNAME_SID_SERIAL#
	<b>Description:</b>	The identifier to be used in labeling the trace file for easier identification in the trace directory.
VERBOSE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	YES
	<b>Description:</b>	Used to determine if commands being executed are displayed back to the screen or not.
	<b>Options:</b>	NO = Suppress Verboseness YES = Verbose

### Exceptions

- NO\_DATA\_FOUND – Unable to obtain session details.
- INSUFFICIENT\_PRIVS – ORA-01031. Grant ALTER SESSION to INUSER to resolve.
- [OTHERS](#)

### Examples

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.TRACE_SESSION('HELP');
EXEC IN_DB_UTIL.TRACE_SESSION('ON', 12, 'INUSER_TRACE', 'YES');
EXEC IN_DB_UTIL.TRACE_SESSION(TRACING=>'ON', TRACE_LEVEL=>4);
EXEC IN_DB_UTIL.TRACE_SESSION(TRACING=>'OFF');
EXEC IN_DB_UTIL.TRACE_SESSION(TRACING=>'REPORT');
```

## Metrics stored procedures and functions

The following function and stored procedure can be used to display throughput metrics of various scripted operations for informational purposes.

### GET\_SECONDS

The GET\_SECONDS stored procedure is used to calculate the number of seconds between two timestamps. This procedure is used when evaluating the duration for an operation.

Parameter	Description	
START_TIMESTAMP	Type:	INPUT
	Datatype:	TIMESTAMP
	Default value:	None
	Description:	The starting timestamp for an operation.
END_TIMESTAMP	Type:	INPUT
	Datatype:	TIMESTAMP
	Default value:	SYSTIMESTAMP
	Description:	The end timestamp for an operation.
ELAPSED_SECONDS	Type:	OUT
	Datatype:	NUMBER
	Default value:	0
	Description:	The calculated number of seconds to return to the caller.

### Examples

```
EXEC IN_DB_UTIL.GET_SECONDS(V_START_TIMESTAMP, NULL, V_TOTAL_SECS);
```

```
SET TIMING ON;
SET SERVEROUTPUT ON;

DECLARE
  V_START_TIME TIMESTAMP := SYSTIMESTAMP;
  V_TOTAL_SECS NUMBER;
BEGIN
  DBMS_SESSION.SLEEP(5);
  IN_DB_UTIL.GET_SECONDS(V_START_TIME, NULL, V_TOTAL_SECS);
  DBMS_OUTPUT.PUT_LINE(CHR(10)||'Elapsed Seconds: '||V_TOTAL_SECS);
END;
/
```

### Example Output

```
Elapsed Seconds: 4.97

PL/SQL procedure successfully completed.

Elapsed: 00:00:04.97
```

## GET\_DURATION

The GET\_DURATION stored procedure is used to parse seconds into a message for displaying the elapsed time in Days, Hours, Minutes and Seconds.

Parameter	Description	
ELAPSED_SECONDS	Type:	INPUT
	Datatype:	NUMBER
	Default value:	None
	Description:	The number of seconds to be parsed.
ELAPSED_MESSAGE	Type:	OUT
	Datatype:	VARCHAR2(200)
	Default value:	None
	Description:	The message displaying the duration to return to the caller.

### Examples

```
EXEC IN_DB_UTIL.GET_DURATION(V_TOTAL_SECS, V_ELAPSED_MESSAGE);
```

```
SET SERVEROUTPUT ON;

DECLARE
  V_ELAPSED_MESSAGE VARCHAR2(200);
BEGIN
  IN_DB_UTIL.GET_DURATION(6400, V_ELAPSED_MESSAGE);
  DBMS_OUTPUT.PUT_LINE(CHR(10)||'Duration: '||V_ELAPSED_MESSAGE);
END;
/
```

### Example Output

```
Duration: 1 Hour 46 Minutes 40 Seconds

PL/SQL procedure successfully completed.
```

## GET\_RPM

The GET\_RPM stored procedure is used to calculate throughput and return Rows Per Minute (RPM). This can be used to quantify the throughput of an operation when measuring the duration and number of rows affected.

Parameter	Description	
ROWS	Type:	INPUT
	Datatype:	NUMBER
	Default value:	None

Parameter	Description	
	<b>Description:</b>	The number of rows affected.
SECONDS	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	None
	<b>Description:</b>	Elapsed Seconds.
RPM	<b>Type:</b>	OUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	None
	<b>Description:</b>	The calculated rows per minute to represent throughput performance for measured operations.

### Examples

```
EXEC IN_DB_UTIL.GET_RPM(V_TOTAL_ROWS, V_TOTAL_SECS, V_ROWS_PER_MINUTE);

SET SERVEROUTPUT ON;

DECLARE
  V_TOTAL_ROWS NUMBER := 123456;
  V_TOTAL_SECS NUMBER := 5;
  V_RPM        NUMBER := 0;
BEGIN
  IN_DB_UTIL.GET_RPM(V_TOTAL_ROWS, V_TOTAL_SECS, V_RPM);
  DBMS_OUTPUT.PUT_LINE(CHR(10)||'Rows Per Minute: '||TRIM(TO_CHAR(V_RPM, '999,999,999')));
END;
/
```

### Example Output

```
Rows Per Minute: 1,481,472

PL/SQL procedure successfully completed.
```

## Database maintenance stored procedures

The following stored procedures have been made available to assist with common database maintenance tasks.

### CLEANUP\_AUDIT\_DATA

The 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 KEEP\_DAYS parameter or the BEGIN\_DATETIME and END\_DATETIME parameters.

It is advised to only execute this procedure after hours or on weekends to ensure minimal impact to the Perceptive Content application.

**Transactions**

From a transactional 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 100,000 each that correspond to the rows in the IN\_AUDIT table. This means that each transaction could affect roughly three times the number of rows as specified in the BATCH\_SIZE parameter. A BATCH\_SIZE of 100,000 could result in a transaction size of roughly 300,000 or more rows.

If necessary, utilize smaller batch sizes, using the BATCH\_SIZE parameter, to help keep transaction sizes smaller and reduce the scope and duration of database resources.

**Parallelism**

It is not possible to scope a transaction across multiple statements/tables when using parallelism due to the ORA-12839 error. Because of that limitation, transactions will be handled in the following manner based on the specified degree of parallelism.

- If PARALLEL > 1
  - Transactions will be scoped for each table and will commit/rollback after rows are deleted from each of the three tables.
  - If an error occurs only the records for the current table is rolled back.
- If PARALLEL = 1 (NOPARALLEL)
  - Transactions will be scoped across all three tables (entire batch) and the commit/rollback of deleted rows will affect all three tables.
  - If an error occurs then all records for the batch (all three tables) is rolled back.

Parameter	Description	
RUN_MODE	Type:	INPUT
	Datatype:	VARCHAR2(8)
	Default value:	HELP
	Description:	Indicates the mode to use during execution.
	Options:	HELP = Display usage details. COUNT = Count qualifying rows. DELETE = Delete qualifying rows. TRUNCATE = Truncate audit tables.
KEEP_DAYS	Type:	INPUT
	Datatype:	NUMBER
	Default value:	NULL

Parameter	Description	
	<b>Description:</b>	Indicates the number of days to retain.
	<b>Options:</b>	0 = Delete All Rows Any number of days greater than 0.
START_TIMESTAMP	<b>Type:</b>	INPUT
	<b>Datatype:</b>	TIMESTAMP
	<b>Default value:</b>	NULL
	<b>Description:</b>	Starting timestamp for rows to delete. Inclusive.
	<b>Options:</b>	Any timestamp less than end timestamp.
END_TIMESTAMP	<b>Type:</b>	INPUT
	<b>Datatype:</b>	TIMESTAMP
	<b>Default value:</b>	NULL
	<b>Description:</b>	Ending timestamp for rows to delete. Inclusive.
	<b>Options:</b>	Any timestamp greater than start timestamp.
CONVERT_TO_UTC	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	YES
	<b>Description:</b>	Whether to convert the input timestamps to UTC to align with the timestamps in the database tables which are stored as UTC.
	<b>Options:</b>	YES = Convert to UTC NO = Do Not Convert to UTC
USE_TIME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	NO
	<b>Description:</b>	Whether to include the time part of timestamps when evaluating which records to delete.
	<b>Options:</b>	NO = Do not consider the time part of the timestamp. YES = Consider the time part for more granular timestamps.

Parameter	Description	
MAX_MINUTES	Type:	INPUT
	Datatype:	NUMBER
	Default value:	0
	Description:	Maximum number of minutes to execute. This only applies if using BATCH_SIZE greater than 0.
	Options:	0 = No Time Limit Any number of minutes greater than 0.
BATCH_SIZE	Type:	INPUT
	Datatype:	NUMBER
	Default value:	100,000
	Description:	Indicates the batch size if breaking up deletion into smaller transactions. Note that actual transaction sizes are roughly three times larger than the specified value for BATCH_SIZE since all three audit tables are included within a single transaction.
	Options:	0 = Single transaction 1 or more rows.
INCLUDE_SUMMARY	Type:	INPUT
	Datatype:	VARCHAR2(3)
	Default value:	YES
	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:	YES = Fetch and display the number of qualifying rows for each table NO = Do not fetch and display the number of qualifying rows
PARALLEL	Type:	INPUT
	Datatype:	NUMBER
	Default value:	1

Parameter	Description	
	<b>Description:</b>	Degree of parallelism to force the session to use during audit data cleanup.
	<b>Options:</b>	1 or more. 1 = NOPARALLEL
VERBOSE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(3)
	<b>Default value:</b>	NO
	<b>Description:</b>	Specifies whether to display additional details in the log, including SQL statements.
	<b>Options:</b>	NO = No additional logging YES = Include extra details in the log

### Exceptions

- [FAILED\\_CHECK](#)
- [IS\\_EXECUTING](#)
- CLEANUP\_ERROR – An error occurred while deleting qualifying audit data.
- [OTHERS](#)

### Recommended Index

The following index is recommended to facilitate efficient searches using the CREATION\_TIME column and should be created prior to executing the CLEANUP\_AUDIT\_DATA stored procedure.

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

### Examples

- Display Usage Information Only

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.CLEANUP_AUDIT_DATA;
EXEC IN_DB_UTIL.CLEANUP_AUDIT_DATA('HELP');
```

- KEEP\_DAYS Example

The following will delete all audit data that is older than 365 days using batch sizes of 100,000 rows. Please see the information above regarding transaction sizes and parallelism for details on how the transactions are scoped during the execution of the CLEANUP\_AUDIT\_DATA procedure.

```
SET SERVEROUTPUT ON;
BEGIN
IN_DB_UTIL.CLEANUP_AUDIT_DATA
```

```
(
  RUN_MODE      => 'DELETE'
,KEEP_DAYS     => 365
,MAX_MINUTES   => 60
,BATCH_SIZE    => 100000
,PARALLEL      => 4
);
END;
/
```

• **Date Range Example**

The following example will delete all audit data that was created on December 25, 2021 between 8am and 5pm using a single transaction (no batches).

- Note that to preserve the time part of the timestamps you must set USE\_TIME = 1
- Note that CONVERT\_TO\_UTC = 1 (default) which means the provided timestamps will be adjusted to the UTC timestamp, based on the server differential, so that it aligns with the timestamps in the database.

```
SET SERVEROUTPUT ON;

BEGIN
IN_DB_UTIL.CLEANUP_AUDIT_DATA
(
  RUN_MODE      => 'DELETE'
,START_TIMESTAMP => '2021-12-25 08:00'
,END_TIMESTAMP  => '2021-12-25 17:00'
,CONVERT_TO_UTC => 'YES'
,USE_TIME       => 'YES'
,BATCH_SIZE     => 0
,VERBOSE        => 'YES'
);
END;
/
```

**Truncate Table Example**

The following example will truncate each of the three audit tables. This is the fastest way to remove all rows from each of the tables. This operation is unrecoverable.

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.CLEANUP_AUDIT_DATA (RUN_MODE=>'TRUNCATE', INCLUDE_SUMMARY=>'NO');
```

## REBUILD\_INDEXES

The REBUILD\_INDEXES stored procedure can be used to rebuild indexes in the INUSER schema. It's not very often that indexes should require a rebuild but the following procedure helps facilitate the operations should it become necessary.

Parameter	Description	
SCHEMA_NAME	Type:	INPUT
	Datatype:	VARCHAR2(128)
	Default value:	INUSER

Parameter	Description	
	<b>Description:</b>	The name of the schema owning the table and indexes.
TABLE_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The table name for which to rebuild its indexes.
INDEX_NAME	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The name of the index to rebuild.
TABLESPACE	<b>Type:</b>	INPUT
	<b>Datatype:</b>	VARCHAR2(128)
	<b>Default value:</b>	NULL
	<b>Description:</b>	The name of the tablespace if moving the indexes to a new tablespace.
MAX_MINUTES	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	0
	<b>Description:</b>	The number of minutes to allow the procedure to start new index rebuilds. 0 = No time limit
PARALLEL	<b>Type:</b>	INPUT
	<b>Datatype:</b>	NUMBER
	<b>Default value:</b>	1
	<b>Description:</b>	The degree of parallelism to use while rebuilding indexes 1 or More 1 = NOPARALLEL

Parameter	Description	
LOGGING	Type:	INPUT
	Datatype:	VARCHAR2(9)
	Default value:	LOGGING
	Description:	Specifies if the index rebuild should be done with LOGGING or NOLOGGING.

### Exceptions

- [FAILED\\_CHECK](#)
- [IS\\_EXECUTING](#)
- [OTHERS](#)

### Examples

```
SET SERVEROUTPUT ON;

-- Rebuild all indexes on the IN_WF_ITEM table with a degree of 4 for parallelism
EXEC IN_DB_UTIL.REBUILD_INDEXES ('INUSER', 'IN_WF_ITEM', NULL, NULL, 1, 4, 'LOGGING');

-- Rebuild only the WF_ITEM_IDX_05 index
EXEC IN_DB_UTIL.REBUILD_INDEXES ('INUSER', 'IN_WF_ITEM', 'WF_ITEM_IDX_05');
```

### Example Output

```
SQL> EXEC IN_DB_UTIL.REBUILD_INDEXES ('INUSER', 'IN_WF_ITEM', 'WF_ITEM_IDX_05',NULL,0,4);

Procedure Name: REBUILD_INDEXES

Current DB Time: 2022-09-02 12:16:36.780746
Current UTC Time: 2022-09-02 17:16:36.780746

Current Activity for the IN_DB_UTIL identifier:

IN_DB_UTIL TAG:                06PLM0ZOVQ
IN_DB_UTIL STATUS:             EXECUTING
IN_DB_UTIL OPERATION:          REBUILD_INDEXES
IN_DB_UTIL CALLED BY:          NULL
IN_DB_UTIL START TIME:         2022-09-02 12:16:36.731688
IN_DB_UTIL MAX TIME:           NULL
IN_DB_UTIL MAX MINUTES:        NULL
IN_DB_UTIL OBJECT:             NULL
IN_DB_UTIL SUBOBJECT:          NULL
IN_DB_UTIL TABLESPACE:        NULL
IN_DB_UTIL BATCH SIZE:         NULL
IN_DB_UTIL BULK LIMIT:         NULL
IN_DB_UTIL PARALLEL:           4
IN_DB_UTIL LOGGING:            LOGGING
IN_DB_UTIL OS USER:            OSUSER
IN_DB_UTIL DB_SESSION:         126,65152
```

```

Using the default value of 0 for MAX_MINUTES.

Indexes will be rebuilt until all qualifying indexes have been rebuilt.

*****
***      Index Rebuilds      ***
*****

----- Index Details -----

Table Name:      IN_WF_ITEM
Index Name:      WF_ITEM_IDX_05
Tablespace Name: NEWINDX
Status:         VALID
Last Analyzed:  2022-09-02 12:16:20
Sample Size:    8,568,289
Number of Rows: 8,568,289

ALTER INDEX INUSER.WF_ITEM_IDX_05 REBUILD ONLINE LOGGING;

Duration: 6.486054 Seconds

Rebuilt the INUSER.WF_ITEM_IDX_05 index.

*****
***      Index Rebuild Results      ***
*****

Successfully rebuilt 1 index

Total Duration: 6.538909 Seconds

The procedure completed successfully.

PL/SQL procedure successfully completed.

Elapsed: 00:00:06.63

```

## Exceptions

The following is a partial list of custom exceptions that are commonly used across the various procedures and functions. Please reference the IN\_DB\_UTIL creation script for a complete list of exceptions.

Custom Exception	Error	Description
INVALID_IDENTIFIER	-20001	An unsupported identifier was provided for the IN_CONTEXT namespace.
NAME_UNAVAILABLE	-20002	The new object name is already being used and is unavailable.
INVALID_CHECK_TYPE	-20004	Raised the CHECK_PARAMETER procedure when an unsupported CHECK_TYPE was used.

INVALID_OBJECT_TYPE	-20005	The specified object type is invalid.
INVALID_OBJECT	-20006	The specified object name was not found.
INVALID_FQN	-20007	Raised by the CHECK_PARAMETER procedure if the constructed object name is malformed and is not OWNER.OBJECT_NAME.
FAILED_CHECK	-20030	The parameter value, as checked by the CHECK_PARAMETER procedure, is invalid.
IS_EXECUTING	-20040	An attempt was made to initialize a session for one of IN_CONTEXT namespace identifiers but the identifier is already in use by another session.
TAG_MISMATCH	-20050	The TAG that was provided does not match the TAG associated with the process that is currently using the specified identifier.
OTHERS		All other unhandled exceptions.

## Troubleshooting commands

The [IN\\_CONTEXT\\_PRINT](#) and [IN\\_CALLSTACK\\_PRINT](#) procedures can be used to view context attribute values and call stack details if needed for troubleshooting. For more detailed troubleshooting information you can enable [DEBUG](#) mode when compiling the package.

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

```
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT;
```

- To view active session details for the IN\_DB\_UTIL identifier, execute the following command.

```
EXEC IN_DB_UTIL.IN_CONTEXT_PRINT('IN_DB_UTIL');
```

- To view call stack details for the SYNC and UPGRADE identifiers, execute the following command.

```
EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT;
```

- To view call stack details for the IN\_DB\_UTIL identifier, execute the following command.

```
EXEC IN_DB_UTIL.IN_CALLSTACK_PRINT('IN_DB_UTIL');
```

The [IN\\_DB\\_UTIL\\_LOG](#) table contains a log of any exceptions that occur during the execution of the IN\_DB\_UTIL and IN\_DB\_UPGRADE packages. It also contains a log of calls to the IN\_SESSION\_SET, and IN\_SESSION\_CLEAR, and IN\_SESSION\_CLEAR\_ALL stored procedures.

```
SELECT * FROM INUSER.IN_DB_UTIL_LOG ORDER BY LOG_TIME;
```

## Cleanup after an early exit

You can use the [IN\\_SESSION\\_CLEAR](#) and [IN\\_SESSION\\_CLEAR\\_ALL](#) procedures to manually clear the call stack and session attributes as needed due to an unhandled exception or after manually cancelling an operation before completion.

**Important** Do not execute any of the following commands 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.

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

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL ('ALL');
```

- Clear session and call stack for only the SYNC identifier

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL ('SYNC');
```

- Clear session and call stack for only the UPGRADE identifier

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL ('UPGRADE');
```

- Clear session and call stack for only the IN\_DB\_UTIL identifier

```
EXEC IN_DB_UTIL.IN_SESSION_CLEAR_ALL ('IN_DB_UTIL');
```

## Debugging

If additional logging is necessary for debugging purposes, the package can be compiled with the `DEBUG` parameter equal to 'YES'.

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.

```
-- Enable/Disable Additional Debugging Information.
-- 'YES' = Enable Debugging Output During Execution

DEBUG VARCHAR2(3) := UPPER('YES');
```

## Removing the IN\_DB\_UTIL package

Execute the following command to drop the `IN_DB_UTIL` package from an Oracle database.

```
DROP PACKAGE INUSER.IN_DB_UTIL;
```

## 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 [GET\\_SECONDS](#) function and the [GET\\_DURATION](#) and [GET\\_RPM](#) stored procedures 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)

```

SET TIMING ON;
SET SERVEROUTPUT ON;

DECLARE
V_START_TIMESTAMP    TIMESTAMP;
V_ROWS_PER_MINUTE    NUMBER;
V_TOTAL_ROWS         NUMBER;
V_ELAPSED_SECONDS    NUMBER;
V_ELAPSED_MESSAGE    VARCHAR2(200);
V_ERROR_MSG          VARCHAR2(500);

OBJECT_EXISTS EXCEPTION; PRAGMA EXCEPTION_INIT(OBJECT_EXISTS, -955);

BEGIN
  -- Record the start time
  V_START_TIMESTAMP := CURRENT_TIMESTAMP;

  -- Execute any operation that affects rows
  EXECUTE IMMEDIATE 'CREATE TABLE INUSER.IN_AUDIT_DETAIL_BAK AS SELECT * FROM INUSER.IN_AUDIT_DETAIL';

  -- Record the number of rows
  V_TOTAL_ROWS := SQL%ROWCOUNT;

  -- Calculate Duration in Seconds
  IN_DB_UTIL.GET_SECONDS(V_START_TIMESTAMP, CURRENT_TIMESTAMP, V_ELAPSED_SECONDS);

  -- Get Duration Message to Display
  IN_DB_UTIL.GET_DURATION(V_ELAPSED_SECONDS, V_ELAPSED_MESSAGE);

  -- Get Rows Per Minute
  IN_DB_UTIL.GET_RPM(V_TOTAL_ROWS, V_ELAPSED_SECONDS, V_ROWS_PER_MINUTE);

  -- Rows Affected
  DBMS_OUTPUT.PUT_LINE(CHR(10)||'Total Rows Affected was '||TRIM(TO_CHAR(V_TOTAL_ROWS, '999,999')));

  -- Display Elapsed Time Message
  DBMS_OUTPUT.PUT_LINE('Total Duration was '||V_ELAPSED_MESSAGE);

  -- Display Throughput RPM (Rows Per Minute)
  DBMS_OUTPUT.PUT_LINE('Total Rows Per Minute was '||TRIM(TO_CHAR(V_ROWS_PER_MINUTE, '999,999,999')));

  -- Drop the backup table
  EXECUTE IMMEDIATE 'DROP TABLE INUSER.IN_AUDIT_DETAIL_BAK';

EXCEPTION
  WHEN OBJECT_EXISTS THEN
    DBMS_OUTPUT.PUT_LINE(CHR(10)||'The INUSER.IN_AUDIT_DETAIL_BAK table already exists.');
```

```

  WHEN OTHERS THEN
    V_ERROR_MSG := SUBSTR(SQLERRM, 1, 500);
    DBMS_OUTPUT.PUT_LINE(CHR(10)||'ERROR - '||V_ERROR_MSG);
END;
/
```

### Example Output

```

Total Rows Affected was 6,272
Total Duration was .041 Seconds
Total Rows Per Minute was 9,178,537
```

## Coding with the MAX\_MINUTES parameter

The following example demonstrates how to use the MAX\_MINUTES parameter, of the IN\_DB\_UTIL framework, to define an end time that can be evaluated when looping through multiple operations.

Note that any operation that is started will run to completion, but new operations will not be started if the specified number of minutes has elapsed.

This should only be used for situations where it is acceptable that any remaining operations are not executed if time has expired.

While the example below is executing, you can monitor progress by running the following command, from a separate session, to view the scripted updates to the START\_TIME and SUBOBJECT attributes for the IN\_DB\_UTIL identifier. Any of the other supported attributes can also be updated as needed.

```
SET SERVEROUTPUT ON;

EXEC IN_DB_UTIL.IN_CONTEXT_PRINT('IN_DB_UTIL');
```

The following example will run for one minute (MAX\_MINUTES=1) and will update the START TIME and SUBOBJECT attributes of the IN\_DB\_UTIL identifier every ten seconds until time has expired. The output will be displayed only after MAX\_MINUTES has elapsed, and the PL/SQL has completed.

```
SET TIMING ON;
SET SERVEROUTPUT ON;

DECLARE
  V_COUNTER      NUMBER := 0;
  V_CHECK_TIME   NUMBER := 0;
  V_MAX_MINUTES  NUMBER := 1;
  V_OPERATION    VARCHAR2(128) := 'TEST';
  V_IDENTIFIER   VARCHAR2(128) := 'IN_DB_UTIL';
  V_TAG          VARCHAR2(10)  := DBMS_RANDOM.STRING('X',10);
BEGIN
  IN_DB_UTIL.IN_SESSION_SET(V_IDENTIFIER, V_TAG, V_OPERATION);
  IN_DB_UTIL.IN_CONTEXT_SET('OBJECT', V_OPERATION, V_IDENTIFIER);
  IN_DB_UTIL.IN_CONTEXT_SET('MAX_MINUTES', V_MAX_MINUTES, V_IDENTIFIER);
  IN_DB_UTIL.IN_CALLSTACK_PRINT(V_IDENTIFIER);
  IN_DB_UTIL.IN_CONTEXT_PRINT(V_IDENTIFIER);
  DBMS_OUTPUT.PUT_LINE(CHR(10));

  WHILE V_CHECK_TIME = 0
  LOOP
    V_COUNTER := V_COUNTER+1;
    IN_DB_UTIL.IN_CONTEXT_SET('SUBOBJECT', 'PASS# '||V_COUNTER, V_IDENTIFIER);
    IN_DB_UTIL.IN_CONTEXT_SET('STATUS', 'EXECUTING', V_IDENTIFIER);
    DBMS_SESSION.SLEEP(10);

    SELECT INUSER.IN_DB_UTIL.CHECK_TIME(V_IDENTIFIER) INTO V_CHECK_TIME FROM DUAL;

    IF V_CHECK_TIME = 0 THEN
      DBMS_OUTPUT.PUT_LINE('MAX_TIME has not elapsed.');
```

```
    ELSIF V_CHECK_TIME = 1 THEN
      DBMS_OUTPUT.PUT_LINE('MAX_TIME has elapsed.');
```

```
    END IF;

  END LOOP;

  IN_DB_UTIL.IN_SESSION_CLEAR(V_IDENTIFIER, V_TAG);
  IN_DB_UTIL.IN_CALLSTACK_PRINT(V_IDENTIFIER);
  IN_DB_UTIL.IN_CONTEXT_PRINT(V_IDENTIFIER);
END;
/
```

### Example Output

```
Current DB Time: 2022-08-22 17:44:23.937000
Current UTC Time: 2022-08-22 22:44:23.937000

Current Activity for the IN_DB_UTIL identifier:
```

```
IN_DB_UTIL TAG:                N1Q5DYNBN9
IN_DB_UTIL STATUS:             EXECUTING
IN_DB_UTIL OPERATION:          TEST
IN_DB_UTIL CALLED BY:          NULL
IN_DB_UTIL START TIME:         2022-08-22 17:44:23.937000
IN_DB_UTIL MAX TIME:           2022-08-22 17:45:23.937000
IN_DB_UTIL MAX MINUTES:        1
IN_DB_UTIL OBJECT:             NULL
IN_DB_UTIL SUBOBJECT:          NULL
IN_DB_UTIL TABLESPACE:        NULL
IN_DB_UTIL BATCH SIZE:         NULL
IN_DB_UTIL BULK LIMIT:         NULL
IN_DB_UTIL PARALLEL:           NULL
IN_DB_UTIL LOGGING:            NULL
IN_DB_UTIL OS USER:             LOCAL_DOMAIN\USER_NAME
IN_DB_UTIL DB_SESSION:         10,60461
```

```
MAX_TIME has not elapsed.
MAX_TIME has not elapsed.
MAX_TIME has not elapsed.
MAX_TIME has not elapsed.
MAX_TIME has not elapsed.
MAX_TIME has elapsed.
```

```
Current DB Time: 2022-08-22 17:45:23.990000
Current UTC Time: 2022-08-22 22:45:23.990000
```

Current Activity for the IN\_DB\_UTIL identifier:

```
IN_DB_UTIL TAG:                NULL
IN_DB_UTIL STATUS:             IDLE
IN_DB_UTIL OPERATION:          NULL
IN_DB_UTIL CALLED BY:          NULL
IN_DB_UTIL START TIME:         NULL
IN_DB_UTIL MAX TIME:           NULL
IN_DB_UTIL MAX MINUTES:        NULL
IN_DB_UTIL OBJECT:             NULL
IN_DB_UTIL SUBOBJECT:          NULL
IN_DB_UTIL TABLESPACE:        NULL
IN_DB_UTIL BATCH SIZE:         NULL
IN_DB_UTIL BULK LIMIT:         NULL
IN_DB_UTIL PARALLEL:           NULL
IN_DB_UTIL LOGGING:            NULL
IN_DB_UTIL OS USER:             NULL
IN_DB_UTIL DB_SESSION:         NULL
```

PL/SQL procedure successfully completed.

Elapsed: 00:01:00.08