Oracle® Database

SODA for PL/SQL Developer's Guide

Release 18c
E84719-04
February 2019

Oracle Database SODA for PL/SQL Developer's Guide, Release 18c

E84719-04

Primary Author: Drew Adams

Contributors: Douglas McMahon, Maxim Orgiyan, Srikrishnan Suresh

Contents

Index

List of Examples

List of Tables

Preface

This document describes how to use Simple Oracle Document Access (SODA) for C.

Audience

This document is intended for users of Simple Oracle Document Access (SODA) for PL/SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- https://docs.oracle.com/en/database/oracle/simple-oracle-document-access/ for complete information about SODA and its implementations
- *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for general information about SODA
- *Oracle as a Document Store* for general information about using JSON data in Oracle Database, including with SODA
- *Oracle Database JSON Developer's Guide* for information about using SQL and PL/SQL with JSON data stored in Oracle Database

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at OTN Registration.

If you already have a user name and password for OTN then you can go directly to the documentation section of the OTN Web site at OTN Documentation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

SODA for PL/SQL Prerequisites

SODA for PL/SQL is an integral part of Oracle Database, starting with Release 18c (18.1). The database is the only prerequisite for using SODA for PL/SQL, but some features are available only starting with particular database releases.

The following features were added to SODA for PL/SQL in Oracle Database Release 18.3. You need that database release or later to use them:

- Data-type `SODA_OPERATION_T`
- Indexing
- JSON data guide

2
SODA for PL/SQL Overview

SODA for PL/SQL is a PL/SQL API that implements **Simple Oracle Document Access** (SODA). You can use it with PL/SQL to perform create, read (retrieve), update, and delete (CRUD) operations on documents of any kind, and you can use it to query JSON documents.

SODA is a set of NoSQL-style APIs that let you create and store collections of documents in Oracle Database, retrieve them, and query them, without needing to know Structured Query Language (SQL) or how the data in the documents is stored in the database.

Oracle Database supports storing and querying JSON data. To access this functionality, you need structured query language (SQL) with special JSON SQL operators. SODA for PL/SQL hides the complexities of SQL/JSON programming.

The remaining topics of this document describe various features of SODA for PL/SQL.

Note:

- This book provides information about using SODA with PL/SQL applications, and it describes all SODA features currently available for use with PL/SQL. To use SODA for PL/SQL you also need to understand SODA generally. For such general information, please consult *Oracle Database Introduction to Simple Oracle Document Access (SODA)*. Some features described in that book are not yet available with SODA for PL/SQL.
- This book does not provide general information about PL/SQL, including reference information about the SODA for PL/SQL methods and constants. For such information, please consult *Oracle Database PL/SQL Language Reference*.

See Also:

Oracle Database JSON Developer's Guide for information about using SQL and PL/SQL with JSON data stored in Oracle Database

3
Using SODA for PL/SQL

How to access SODA for PL/SQL is described, as well as how to use it to perform create, read (retrieve), update, and delete (CRUD) operations on collections.

(CRUD operations are also called “read and write operations” in this document.)

- Getting Started with SODA for PL/SQL
  How to access SODA for PL/SQL is described, as well as how to use it to create a database collection, insert a document into a collection, and retrieve a document from a collection.
- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.
- Opening an Existing Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to open an existing document collection.
- Checking Whether a Given Collection Exists with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.
- Discovering Existing Collections with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.list_collection_names` to discover existing collections.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Creating Documents with SODA for PL/SQL
  You use a constructor for PL/SQL object type `SODA_DOCUMENT_T` to create SODA documents.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.
- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.
- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.
- Indexing the Documents in a Collection with SODA for PL/SQL
  You index the documents in a SODA collection with `SODA_COLLECTION_T` method `create_index()`. Its input parameter is a textual JSON index specification. This can specify support for B-tree, spatial, full-text, and ad hoc indexing, and it can specify support for a JSON data guide.
- Getting a Data Guide for a Collection with SODA for PL/SQL
  You use `SODA_COLLECTION_T` method `get_Data_Guide()` to get a data guide for a collection. A data guide is a JSON document that summarizes the structural and type information of the JSON documents in the collection. It records metadata about the fields used in those documents.
- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.

3.1 Getting Started with SODA for PL/SQL

How to access SODA for PL/SQL is described, as well as how to use it to create a database collection, insert a document into a collection, and retrieve a document from a collection.

> **Note:**
>
> Don’t worry if not everything in this topic is clear to you on first reading. The necessary concepts are developed in detail in other topics. This topic should give you an idea of what is involved overall in using SODA.

Follow these steps to get started with SODA for PL/SQL:

1. Ensure that the prerequisites have been met for using SODA for PL/SQL. See SODA for PL/SQL Prerequisites.
2. Identify the database schema (user account) used to store collections, and grant database role `SODA_APP` to that schema:

   ```
   GRANT SODA_APP TO schemaName;
   ```

3. Use PL/SQL code such as that in Example 3-1 to do the following:

a. Create and open a collection (an instance of PL/SQL object type `SODA_COLLECTION_T`), using the default collection configuration (metadata).

b. Create a document with particular JSON content, as an instance of PL/SQL object type `SODA_DOCUMENT_T`.

c. Insert the document into the collection.

d. Get the inserted document back. Its other components, besides the content, are generated automatically.

e. Print the unique document key, which is one of the components generated automatically.

f. Commit the document insertion.

g. Find the document in the collection, by providing its key.

h. Print some of the document components: key, content, creation timestamp, last-modified timestamp, and version.

4. Drop the collection, cleaning up the database table that is used to store the collection and its metadata:

```
SELECT DBMS_SODA.drop_collection('myCollectionName') AS drop_status
FROM DUAL;
```

> **Caution:**
>
> Do *not* use SQL to drop the database *table* that underlies a collection. Dropping a *collection* involves more than just dropping its database table. In addition to the documents that are stored in its table, a collection has *metadata*, which is also persisted in Oracle Database. Dropping the table underlying a collection does *not* also drop the collection metadata.

> **Note:**
>
> If a PL/SQL subprogram that you write invokes subprograms that are in package `DBMS_SODA`, and if your subprogram has definer (owner) rights, then a database administrator (DBA) must grant role `SODA_APP` to your subprogram. For example, this code grants role `SODA_APP` to procedure `my_soda_proc`, which is owned by database schema (user) `my_db_schema`:
>
> ```
> GRANT SODA_APP TO PROCEDURE my_db_schema.my_soda_proc;
> ```

> **See Also:**
>
> *Oracle Database Security Guide* for information about role `SODA_APP`

Example 3-1 Getting Started Run-Through

```
DECLARE
    collection      SODA_COLLECTION_T;
    document        SODA_DOCUMENT_T;
    foundDocument   SODA_DOCUMENT_T;
    result_document SODA_DOCUMENT_T;
    docKey          VARCHAR2(100);
    status          NUMBER;
BEGIN
    -- Create a collection.
    collection := DBMS_SODA.create_collection('myCollectionName');

    -- The default collection has BLOB content, so create a BLOB-based
document.
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" :
"Alexander"}'));

    -- Insert the document and get it back.
    result_document := collection.insert_one_and_get(document);

    -- The result document has auto-generated components, such as key and
version,
    -- in addition to the content.  Print the auto-generated document key.
    docKey := result_document.get_key;
    DBMS_OUTPUT.put_line('Auto-generated key is ' || docKey);

    -- Commit the insert
    COMMIT;

    -- Find the document in the collection by its key
    foundDocument := collection.find_one(docKey);

    -- Get and print some document components: key, content, etc.
    DBMS_OUTPUT.put_line('Document components:');
    DBMS_OUTPUT.put_line('  Key: ' || foundDocument.get_key);
    DBMS_OUTPUT.put_line('  Content: '
                         ||
utl_raw.cast_to_varchar2(foundDocument.get_blob));
    DBMS_OUTPUT.put_line('  Creation timestamp: ' ||
foundDocument.get_created_on);
    DBMS_OUTPUT.put_line('  Last-modified timestamp: '
                         || foundDocument.get_last_modified);
    DBMS_OUTPUT.put_line('  Version: ' || foundDocument.get_version);
END;
/
```

Example 3-2 Sample Output for Getting Started Run-Through

Example 3-1 results in output similar to this. The values of the auto-generated components will differ in any actual execution.

```
Auto-generated key is 96F35328CD3B4F96BF3CD01BCE9EBDF5
Document components:
```

```
Key: 96F35328CD3B4F96BF3CD01BCE9EBDF5
Content: {"name" : "Alexander"}
Creation timestamp: 2017-09-19T01:05:06.160289Z
Last-modified timestamp: 2017-09-19T01:05:06.160289Z
Version: FD69FB6ACE73FA735EC7922CA4A02DDE0690462583F9EA2AF754D7E342B3EE78
```

3.2 Creating a Document Collection with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

Example 3-3 uses PL/SQL function `DBMS_SODA.create_collection` to create a collection that has the default metadata.

The *default collection metadata* has the following characteristics.

- Each document in the collection has these document components:
  - Key
  - Content
  - Creation timestamp
  - Last-modified timestamp
  - Version
- The collection can store only JSON documents.
- Document keys are automatically generated for documents that you add to the collection.

The default collection configuration is recommended in most cases, but collections are highly configurable. When you create a collection you can specify things such as the following:

- Storage details, such as the name of the table that stores the collection and the names and data types of its columns.
- The presence or absence of columns for creation timestamp, last-modified timestamp, and version.
- Whether the collection can store only JSON documents.
- Methods of document key generation, and whether document keys are client-assigned or generated automatically.
- Methods of version generation.

This configurability also lets you map a new collection to an existing database table.

To configure a collection in a nondefault way, supply custom collection metadata, expressed in JSON, as the second argument to `DBMS_SODA.create_collection`.

If you do not care about the details of collection configuration then pass only the collection name to `DBMS_SODA.create_collection` — no second argument. That creates a collection with the default configuration.

If a collection with the same name already exists then it is simply opened and its handle is returned. If custom metadata is provided and it does not match the metadata

of the existing collection then the collection is not opened and an error is raised. (To match, all metadata fields must have the same values.)

> **Note:**
>
> Unless otherwise stated, the remainder of this documentation assumes that a collection has the default configuration.

> **See Also:**
>
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for information about the default naming of a collection table
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.create_collection`

Example 3-3 Creating a Collection That Has the Default Metadata

This example creates collection `myCollectionName` with the default metadata.

```
DECLARE
    collection  SODA_Collection_T;
BEGIN
    collection := DBMS_SODA.create_collection('myCollectionName');
END;
/
```

Related Topics

- Getting the Metadata of an Existing Collection
  You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.
- Checking Whether a Given Collection Exists with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.

3.3 Opening an Existing Document Collection with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.open_collection` to open an existing document collection.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.open_collection`

Example 3-4 Opening an Existing Document Collection

This example uses PL/SQL function `DBMS_SODA.open_collection` to open the collection named `myCollectionName` and returns a `SODA_COLLECTION_T` instance that represents this collection. If the value returned is `NULL` then there is no existing collection named `myCollectionName`.

```
DECLARE
    collection  SODA_COLLECTION_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
END;
/
```

3.4 Checking Whether a Given Collection Exists with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.open_collection`

Example 3-5 Checking for a Collection with a Given Name

This example uses `DBMS_SODA.open_collection` to try to open an existing collection named `myCollectionName`. It prints a message if no such collection exists.

```
DECLARE
    collection SODA_COLLECTION_T;
```

```
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    IF collection IS NULL THEN
        DBMS_OUTPUT.put_line('Collection does not exist');
    END IF;
END;
/
```

Related Topics

- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

3.5 Discovering Existing Collections with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.list_collection_names` to discover existing collections.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.list_collection_names`

Example 3-6 Printing the Names of All Existing Collections

This example uses PL/SQL function `DBMS_SODA.list_collection_names` to obtain a list of the collection names. It then iterates over that list, printing out the names.

```
DECLARE
    coll_list  SODA_COLLNAME_LIST_T;
BEGIN
    coll_list := DBMS_SODA.list_collection_names;
    DBMS_OUTPUT.put_line('Number of collections: ' ||
to_char(coll_list.count));
    DBMS_OUTPUT.put_line('Collection List: ');
    IF (coll_list.count > 0) THEN
        -- Loop over the collection name list
        FOR i IN
            coll_list.first .. coll_list.last
        LOOP
            DBMS_OUTPUT.put_line(coll_list(i));
        END LOOP;
    ELSE
        DBMS_OUTPUT.put_line('No collections found');
    END IF;
END;
/
```

3.6 Dropping a Document Collection with SODA for PL/SQL

You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.

> **Caution:**
>
> Do *not* use SQL to drop the database *table* that underlies a collection. Dropping a *collection* involves more than just dropping its database table. In addition to the documents that are stored in its table, a collection has *metadata*, which is also persisted in Oracle Database. Dropping the table underlying a collection does *not* also drop the collection metadata.

> **Note:**
>
> Day-to-day use of a typical application that makes use of SODA does not require that you drop and re-create collections. But if you need to do that for any reason then this guideline applies.
>
> Do *not* drop a collection and then re-create it with *different metadata* if there is any application running that uses the collection in any way. Shut down any such applications before re-creating the collection, so that all live SODA objects are released.
>
> There is no problem just dropping a collection. Any read or write operation on a dropped collection raises an error. And there is no problem dropping a collection and then re-creating it with the same metadata. But if you re-create a collection with different metadata, and if there are any live applications using SODA objects, then there is a risk that a stale collection is accessed, and *no error is raised* in this case.

> **Note:**
>
> Commit all writes to a collection *before* using `DBMS_SODA.drop_collection`. For the drop to succeed, all uncommitted writes to the collection must first be either committed or rolled back — you must explicitly use SQL `COMMIT` or `ROLLBACK`.. Otherwise, an exception is raised.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.drop_collection`

Example 3-7 Dropping a Document Collection

This example uses PL/SQL function `DBMS_SODA.drop_collection` to drop collection `myCollectionName`.

If the collection cannot be dropped because of uncommitted write operations then an exception is thrown. If the collection is dropped successfully, the returned status is 1; otherwise, the status is 0. In particular, if a collection with the specified name does not exist, the returned status is 0 — no exception is thrown.

```
DECLARE
    status  NUMBER := 0;
BEGIN
    status := DBMS_SODA.drop_collection('myCollectionName');
END;
/
```

Related Topics

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.

3.7 Creating Documents with SODA for PL/SQL

You use a constructor for PL/SQL object type `SODA_DOCUMENT_T` to create SODA documents.

SODA for PL/SQL represents a document using an instance of PL/SQL object type `SODA_DOCUMENT_T`. This object is a *carrier* of document content and other document components, such as the document key.

Here is an example of the *content* of a JSON document:

```
{ "name" :    "Alexander",
  "address" : "1234 Main Street",
  "city" :    "Anytown",
  "state" :   "CA",
  "zip" :     "12345"
}
```

A document has these **components**:

- Key
- Content
- Creation time stamp
- Last-modified time stamp
- Version
- Media type (`"application/json"` for JSON documents)

You *create a document* by invoking one of the **`SODA_DOCUMENT_T`** constructors. The constructors differ according to the content type of the documents they create: `VARCHAR2`, `CLOB`, or `BLOB`.

You can write a document of a given content type only to a collection whose content column has been defined for documents of that type. For example, you can write (insert or replace) only a document with content type `BLOB` to a collection whose `contentColumn` has a `sqlType` value of `BLOB`. (`BLOB` is the default content type for a collection.)

There are different ways to invoke a document constructor:

- You can provide the document *key*, as the first argument.

  In a collection, each document must have a key. You must provide the key when you create the document *only* if you expect to insert the document into a collection that does *not* automatically generate keys for inserted documents. By default, collections are configured to automatically generate document keys.

- You *must* provide the document *content*. If you also provide the document key then the content is the second argument to the constructor.

  If you provide only the content then you must specify both the formal and actual content parameters, separated by the association arrow (`=>`): **`v_content =>`** *`actual`*, **`c_content =>`** *`actual`*, or **`b_content =>`** *`actual`*, for content of type `VARCHAR2`, `CLOB`, or `BLOB`, respectively.

- You can provide the document media type, which defaults to `"application/json"`. Unless you provide all of the parameters (key, content, and media type) you must specify both the formal and actual media-type parameters, , separated by the association arrow (`=>`): **`media_type =>`** *`actual`*.

Parameters that you do not provide explicitly default to `NULL`.

Providing only the content parameter can be useful for creating documents that you insert into a collection that automatically generates document keys. Providing only the key and content can be useful for creating documents that you insert into a collection that has client-assigned keys. Providing (the content and) the media type can be useful for creating *non*-JSON documents (using a media type other than `"application/json"`).

However you invoke a `SODA_DOCUMENT_T` constructor, doing so sets the components that you provide (the content, possibly the key, and possibly the media type) to the values you provide for them. And it sets the values of the creation time stamp, last-modified time stamp, and version to a SQL `NULL` value.

Object type `SODA_DOCUMENT_T` provides **getter methods** (also known as *getters*), which each retrieve a particular component from a document. (Getter `get_data_type()`

actually returns information about the content component, rather than the component itself.)

Table 3-1 Getter Methods for Documents (SODA_DOCUMENT_T)

Getter Method	Description
`get_created_on()`	Get the *creation time stamp* for the document, as a `VARCHAR2` value.
`get_key()`	Get the unique *key* for the document, as a `VARCHAR2` value.
`get_last_modified()`	Get the *last-modified time stamp* for the document, as a `VARCHAR2` value.
`get_media_type()`	Get the *media type* for the document, as a `VARCHAR2` value.
`get_version()`	Get the document *version*, as a `VARCHAR2` value.
`get_blob()`	Get the document *content*, as a `BLOB` value. The document content must be `BLOB` data, or else an error is raised.
`get_clob()`	Get the document *content*, as a `CLOB` value. The document content must be `CLOB` data, or else an error is raised.
`get_varchar2()`	Get the document *content*, as a `VARCHAR2` value. The document content must be `VARCHAR2` data, or else an error is raised.
`get_data_type()`	Get the data type of the document content, as a `PLS_INTEGER` value. The value is **`DBMS_SODA.DOC_VARCHAR2`** for `VARCHAR2` content, **`DBMS_SODA.DOC_BLOB`** for `BLOB` content, and **`DBMS_SODA.DOC_CLOB`** for `CLOB` content.

Immediately after you create a document, the getter methods return these values:

- Values provided to the constructor
- `"application/json"`, for method `get_media_type()`, if the media type was not provided
- `NULL` for other components

Each content storage data type has an associated content-component getter method. You must use the getter method that is appropriate to each content storage type: `get_varchar2()` for `VARCHAR2` storage, `get_clob()` for `CLOB` storage, and `get_blob()` for `BLOB` storage. Otherwise, an error is raised.

Example 3-8 creates a `SODA_DOCUMENT_T` instance, providing only content. The media type defaults to `"application/json"`, and the other document components default to `NULL`.

Example 3-9 creates a SODA_DOCUMENT_T instance, providing the document key and content. The media type defaults to "application/json", and the other document components default to NULL.

> **See Also:**
>
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an overview of SODA documents
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for restrictions that apply for SODA documents
> - *Oracle Database PL/SQL Packages and Types Reference* for information about object type SODA_DOCUMENT_T constructors and getter methods

Example 3-8 Creating a Document with JSON Content

This example uses SODA_DOCUMENT_T constructors to create three documents, one of each content type. The example provides only the document content (which is the same for each).

The content parameter is different in each case; it specifies the SQL data type to use to store the content. The first document creation here uses content parameter **v**_content, which specifies VARCHAR2 content; the second uses parameter **c**_content, which specifies CLOB content; the third uses parameter **b**_content, which specifies BLOB content.

After creating each document, the example uses getter methods to get the document content. Note that the getter method that is appropriate for each content storage type is used: get_blob() for BLOB content, and so on.

The document with content type BLOB would be appropriate for writing to the collection created in Example 3-3, because that collection (which has the default metadata) accepts documents with (only) BLOB content. The other two documents would not be appropriate for that collection; trying to insert them would raise an error.

```
DECLARE
    v_doc  SODA_DOCUMENT_T;
    b_doc  SODA_DOCUMENT_T;
    c_doc  SODA_DOCUMENT_T;
BEGIN
    -- Create VARCHAR2 document
    v_doc := SODA_DOCUMENT_T(v_content => '{"name" : "Alexander"}');
    DBMS_OUTPUT.put_line('Varchar2 Doc content: ' || v_doc.get_varchar2);

    -- Create BLOB document
    b_doc := SODA_DOCUMENT_T(
               b_content => utl_raw.cast_to_raw('{"name" : "Alexander"}'));
    DBMS_OUTPUT.put_line('Blob Doc content: ' ||
                         utl_raw.cast_to_varchar2(b_doc.get_blob));

    -- Create CLOB document
    c_doc := SODA_DOCUMENT_T(c_content => '{"name" : "Alexander"}');
```

```
    DBMS_OUTPUT.put_line('Clob Doc content: ' || c_doc.get_clob);
END;
/
```

Example 3-9 Creating a Document with Document Key and JSON Content

This example is similar to Example 3-8, but it provides the document key (`myKey`) as well as the document content.

```
DECLARE
    v_doc  SODA_DOCUMENT_T;
    b_doc  SODA_DOCUMENT_T;
    c_doc  SODA_DOCUMENT_T;
BEGIN
    -- Create VARCHAR2 document
    v_doc := SODA_DOCUMENT_T('myKey' , v_content => '{"name" :
"Alexander"}');
    DBMS_OUTPUT.put_line('Varchar2 Doc key: ' || v_doc.get_key);
    DBMS_OUTPUT.put_line('Varchar2 Doc content: ' || v_doc.get_varchar2);

    -- Create BLOB document
    b_doc := SODA_DOCUMENT_T('myKey' ,
                             b_content => utl_raw.cast_to_raw('{"name" :
"Alexander"}'));
    DBMS_OUTPUT.put_line('Blob Doc key: ' || b_doc.get_key);
    DBMS_OUTPUT.put_line('Blob Doc content: ' ||
                         utl_raw.cast_to_varchar2(b_doc.get_blob));

    -- Create CLOB document
    c_doc := SODA_DOCUMENT_T('myKey' , c_content => '{"name" :
"Alexander"}');
    DBMS_OUTPUT.put_line('Clob Doc key: ' || c_doc.get_key);
    DBMS_OUTPUT.put_line('Clob Doc content: ' || c_doc.get_clob);
END;
/
```

Related Topics

- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to

uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.

- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

3.8 Inserting Documents into Collections with SODA for PL/SQL

To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.

Both method `insert_one()` and method `insert_one_and_get()` insert a document into a collection and automatically set the values of the creation time stamp, last-modified time stamp, and version (if the collection is configured to include these components and to generate the version automatically, as is the case by default).

When you insert a document, any document components that currently have `NULL` values (as a result of creating the document without providing those component values) are updated to have appropriate, automatically generated values. Thereafter, other SODA operations on a document can automatically update the last-modified timestamp and version components.

In addition to inserting the document, `insert_one_and_get` returns a result document, which contains the generated document components, such as the key, and which does not contain the content of the inserted document.

> **Note:**
>
> If the collection is configured with client-assigned document keys (which is not the default case), and the input document provides a key that identifies an existing document in the collection, then these methods throw an exception.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one_and_get()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods

Example 3-10 Inserting a Document into a Collection

This example creates a document and inserts it into a collection using SODA_COLLECTION_T method insert_one() .

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" :
"Alexander"}'));

    -- Insert a document
    status := collection.insert_one(document);
END;
/
```

Example 3-11 Inserting a Document into a Collection and Getting the Result Document

This example creates a document and inserts it into a collection using method insert_one_and_get(). It then gets (and prints) each of the generated components from the result document (which contains them). To obtain the components it uses SODA_DOCUMENT_T methods get_key(), get_created_on(), get_last_modified(), and get_version().

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    ins_doc     SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" :
"Alexander"}'));
    ins_doc := collection.insert_one_and_get(document);

    -- Insert the document and get its components
    IF ins_doc IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Inserted document components:');
        DBMS_OUTPUT.put_line('Key: ' || ins_doc.get_key);
        DBMS_OUTPUT.put_line('Creation timestamp: ' ||
ins_doc.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: '
                             || ins_doc.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || ins_doc.get_version);
    END IF;
END;
/
```

Related Topics

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.

3.9 SODA for PLSQL Read and Write Operations

A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

You typically use `SODA_OPERATION_T` to specify all SODA read and write operations other than inserts. You chain together `SODA_OPERATION_T` nonterminal methods to narrow the scope or otherwise condition or qualify a read or write operation.

Nonterminal methods return the same `SODA_OPERATION_T` instance on which they are invoked, which allows you to chain them together. The nonterminal methods are `key()`, `keys()`, `filter()`, `version()`, `skip()`, and `limit()`.

A `SODA_OPERATION_T` **terminal** method at the end of the chain carries out the actual read or write operation. The methods for read operations are `get_cursor()`, `get_one()`, and `count()`. The methods for write operations are `replace_one()`, `replace_one_and_get()`, and `remove()`.

Unless documentation states otherwise, you can chain together any nonterminal methods, and you can end the chain with any terminal method. However, not all combinations make sense. For example, it does not make sense to chain method `version()` together with methods that do not uniquely identify the document, such as `keys()`.

Related Topics

- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal

methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.
- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T`, including each of its methods

3.10 Finding Documents in Collections with SODA for PL/SQL

You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods
> - *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`

Example 3-12 Finding All Documents in a Collection

This example uses `SODA_COLLECTION_T` methods `find()` and `getCursor()` to obtain a cursor for a query result list that contains each document in a collection. It then uses the cursor in a `WHILE` statement to get and print the content of each document in the result list, as a string. Finally, it closes the cursor.

It uses `SODA_DOCUMENT_T` methods `get_key()`, `get_blob()`, `get_created_on()`, `get_last_modified()`, and `get_version()`, to get the document components, which it prints. It passes the document content to SQL/JSON function `json_query` to pretty-print (using keyword `PRETTY`).

> **Note:**
>
> To avoid resource leaks, *close* any cursor that you no longer need.

```
DECLARE
    collection    SODA_COLLECTION_T;
    document      SODA_DOCUMENT_T;
    cur           SODA_CURSOR_T;
    status        BOOLEAN;
BEGIN
    -- Open the collection to be queried
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Open the cursor to fetch the documents
    cur := collection.find().get_cursor();

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: ' ||
json_query(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;

    -- IMPORTANT: You must close the cursor, to release resources.
    status := cur.close;
END;
/
```

Example 3-13 Finding the Unique Document That Has a Given Document Key

This example uses `SODA_COLLECTION_T` methods `find()`, `key()`, and `get_one()` to find the unique document whose key is `"key1"`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Find a document using a key
    document := collection.find().key('key1').get_one;

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: ' || JSON_QUERY(document.get_blob,
'$' PRETTY));
        DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

Example 3-14 Finding Multiple Documents with Specified Document Keys

This example defines key list **`myKeys`**, with (string) keys `"key1"`, `"key2"`, and `"key3"`. It then finds the documents that have those keys, and it prints their components. `SODA_COLLECTION_T` method `keys()` specifies the documents with the given keys.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    status      BOOLEAN;
    myKeys      SODA_KEY_LIST_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Set the keys list
    myKeys := SODA_KEY_LIST_T('key1', 'key2', 'key3');

    -- Find documents using keys
    cur := collection.find().keys(myKeys).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
```

```
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: ' ||
json_query(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;
    status := cur.close;
END;
/
```

Example 3-15 Finding Documents with a Filter Specification

`SODA_COLLECTION_T` method `filter()` provides a powerful way to filter JSON documents in a collection. Its parameter is a JSON query-by-example (QBE, also called a filter specification).

The syntax of filter specifications is an expressive pattern-matching language for JSON documents. This example uses only a very simple QBE, just to indicate how you make use of one in SODA for PL/SQL.

This example does the following:

1. Creates a filter specification that looks for all JSON documents whose `name` field has value `"Alexander"`.
2. Uses the filter specification to find the matching documents.
3. Prints the components of each document.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    status      BOOLEAN;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter specification (QBE)
    qbe := '{"name" : "alexander"}';

    -- Open a cursor for the filtered documents
    cur := collection.find().filter(qbe).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
```

```
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: ' ||
JSON_QUERY(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;
    status := cur.close;
END;
/
```

> **See Also:**
>
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an introduction to SODA filter specifications
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for reference information about SODA filter specifications

Example 3-16 Specifying Pagination Queries with Methods skip() and limit()

This example uses `SODA_COLLECTION_T` methods `filter()`, `skip()` and `limit()` in a pagination query.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_Cursor_T;
    status      BOOLEAN;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter
    qbe := '{"name" : "Alexander"}';

    -- Find all documents that match the QBE, skip over the first 1000 of
them,
    -- limit the number of returned documents to 100
    cur := collection.find().filter(qbe).skip(1000).limit(100).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
```

```
            DBMS_OUTPUT.put_line('Content: ' ||
                                 JSON_QUERY(document.get_blob, '$' PRETTY));
            DBMS_OUTPUT.put_line('Creation timestamp: ' ||
                                 document.get_created_on);
            DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                                 document.get_last_modified);
            DBMS_OUTPUT.put_line('Version: ' || document.get_version);
        END IF;
    END LOOP;
    status := cur.close;
END;
/
```

Example 3-17 Specifying Document Version

This example uses `SODA_COLLECTION_T` method `version()` to specify the document version. This is useful for implementing optimistic locking, when used with the terminal methods for write operations.

You typically use `version()` together with method `key()`, which specifies the document. You can also use `version()` with methods `keyLike()` and `filter()`, provided they identify at most one document.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Find a particular version of the document that has a given key
    document := collection.find().key('key1').version('version1').get_one;

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: ' ||
                             JSON_QUERY(document.get_blob, '$' PRETTY));
        DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                             document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

Example 3-18 Counting the Number of Documents Found

This example uses `SODA_COLLECTION_T` method `count()` to get a count of all of the documents in the collection. It then gets a count of all of the documents that are returned by a filter specification (QBE).

```
DECLARE
    collection  SODA_COLLECTION_T;
```

```
    num_docs    NUMBER;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Get a count of all documents in the collection
    num_docs := collection.find().count;
    DBMS_OUTPUT.put_line('Count: ' || num_docs);

    -- Set the filter
    qbe := '{"name" : "Alexander"}';

    -- Get a count of all documents in the collection that match a filter
spec
    num_docs := collection.find().filter(qbe).count;
    DBMS_OUTPUT.put_line('Count: ' || num_docs);
END;
/
```

Related Topics

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

3.11 Replacing Documents in a Collection with SODA for PL/SQL

You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

In addition to replacing the content, methods `replace_one()` and `replace_one_and_get()` update the values of the last-modified timestamp and the version. Replacement does *not* change the document key or the creation timestamp.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
- *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
- *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `replace_one()`
- *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `replace_one_and_get()`
- *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods

Example 3-19 Replacing a Document in a Collection, Given Its Key, and Getting the Result Document

This example replaces a document in a collection, given its key. It then gets (and prints) the key and the generated components from the result document. To obtain the components it uses `SODA_DOCUMENT_T` methods `get_key()`, `get_created_on()`, `get_last_modified()`, and `get_version()`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    new_doc     SODA_DOCUMENT_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" : "Sriky"}'));
    new_doc := collection.find().key('key1').replace_one_and_get(document);

    IF new_doc IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || new_doc.get_key);
        DBMS_OUTPUT.put_line('Creation timestamp: ' ||
new_doc.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                             new_doc.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || new_doc.get_version);
    END IF;
END;
/
```

Example 3-20 Replacing a Particular Version of a Document

To implement optimistic locking when replacing a document, you can chain together methods `key()` and `version()`, as in this example.

```
DECLARE
    collection  SODA_COLLECTION_T;
```

```
    document    SODA_DOCUMENT_T;
    new_doc     SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Replace version 'version1' of the document that has key 'key1'
    new_doc := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" : "Sriky"}'));
    document :=
collection.find().key('key1').version('version1').replace_one_and_get(new_d
oc);

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: ' ||
                             JSON_QUERY(document.get_blob, '$' PRETTY));
        DBMS_OUTPUT.put_line('Creation timestamp: ' ||
document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                             document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

Related Topics

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

Related Topics

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.

3.12 Removing Documents from a Collection with SODA for PL/SQL

You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T` method `remove()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `remove_one()`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods

Example 3-21 Removing a Document from a Collection Using a Document Key

This example removes the document whose document key is `"key1"`. The removal status (1 if the document was removed; 0 if not) is returned and printed.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Remove document that has key 'key1'
    status := collection.find().key('key1').remove;

    -- Count is 1 if  document was found
    IF status = 1 THEN
        DBMS_OUTPUT.put_line('Document was removed!');
```

```
    END IF;
END;
/
```

Example 3-22 Removing a Particular Version of a Document

To implement optimistic locking when removing a document, you can chain together methods `key()` and `version()`, as in this example.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Remove version 'version1' of the document that has key 'key1'.
    status := collection.find().key('key1').version('version1').remove;

    -- Count is 1, if specified version of document with key 'key1' is
found
    IF status = 1 THEN
        DBMS_OUTPUT.put_line('Document was removed!');
    END IF;
END;
/
```

Example 3-23 Removing Documents from a Collection Using Document Keys

This example removes the documents whose keys are `key1`, `key2`, and `key3`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    num_docs    NUMBER;
    myKeys      SODA_KEY_LIST_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the keys list
    myKeys := SODA_KEY_LIST_T('key1','key2','key3');

    -- Remove documents using keys
    num_docs := collection.find().keys(myKeys).remove;

    DBMS_OUTPUT.put_line('Number of documents removed: ' || num_docs);
END;
/
```

Example 3-24 Removing JSON Documents from a Collection Using a Filter

This example uses a filter to remove the JSON documents whose `greeting` field has value `"hello"`. It then prints the number of documents removed.

```
DECLARE
    collection  SODA_COLLECTION_T;
    num_docs    NUMBER;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter specification
    qbe := '{ "greeting" : "hello" }';

    -- Get a count of all documents in the collection that match the QBE
    num_docs := collection.find().filter(qbe).remove;
    DBMS_OUTPUT.put_line('Number of documents removed: ' || num_docs);
END;
/
```

Related Topics

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

3.13 Indexing the Documents in a Collection with SODA for PL/SQL

You index the documents in a SODA collection with `SODA_COLLECTION_T` method `create_index()`. Its input parameter is a textual JSON index specification. This can specify support for B-tree, spatial, full-text, and ad hoc indexing, and it can specify support for a JSON data guide.

> **Note:**
>
> SODA for PL/SQL support for indexing was added in Oracle Database 18.3. You need that database release or later to use this SODA feature.

You drop an index on a SODA collection with `SODA_COLLECTION_T` method `drop_Index()`.

A JSON search index is used for full-text and ad hoc structural queries, and for persistent recording and automatic updating of JSON data-guide information.

An Oracle Spatial and Graph index is used for GeoJSON (spatial) data.

See Also:

- *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an overview of using SODA indexing
- *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for information about SODA index specifications
- *Oracle Database JSON Developer's Guide* for information about JSON search indexes
- *Oracle Database JSON Developer's Guide* for information about persistent data-guide information as part of a JSON search index
- *Oracle Database JSON Developer's Guide* for information about spatial indexing of GeoJSON data.

Example 3-25 Creating a B-Tree Index for a JSON Field with SODA for PL/SQL

This example creates a B-tree non-unique index for numeric field `address.zip` of the JSON documents in collection `myCollectionName`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    spec        VARCHAR2(700);
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the index specification
    spec := '{"name"   : "ZIPCODE_IDX",
              "fields" : [{"path"     : "address.zip",
                           "datatype" : "number",
                           "order"    : "asc"}]}';
    -- Create the index
    status := collection.create_index(spec);
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

Example 3-26 JSON Search Indexing with SODA for PL/SQL

This example indexes the documents in collection `myCollectionName` for ad hoc queries and full-text search (queries using QBE operator `$contains`), and it automatically accumulates and updates data-guide information about your JSON documents (aggregate structural and type information). The index specification has only field `name` (no field `fields`).

```
DECLARE
    collection  SODA_COLLECTION_T;
    spec        VARCHAR2(700);
    status      NUMBER;
BEGIN
```

```
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the index specification
    indexSpec := '{"name" : "SEARCH_AND_DATA_GUIDE_IDX"}';

    -- Create the index
    status := collection.create_index(indexSpec);
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

The simple index specification it uses is equivalent to this one, which makes explicit the default values:

```
{"name" : "SEARCH_AND_DATA_GUIDE_IDX",
 "dataguide" : "on",
 "search_on" : "text_value"}
```

If you instead wanted *only ad hoc* (search) indexing then you would explicitly specify a value of `"off"` for field `dataguide`. If you instead wanted *only data-guide* support then you would explicitly specify a value of `"none"` for field `search_on`.

> **Note:**
>
> To create a data guide-enabled JSON search index, or to data guide-enable an existing JSON search index, you need database privilege `CTXAPP` and Oracle Database Release 12c (12.2.0.1) or later.

Example 3-27 Dropping an Index with SODA for PL/SQL

To drop an index on a SODA collection, just pass the index name to `SODA_COLLECTION_T` method `drop_index()`. This example drops index `myIndex`.

```
DECLARE
    coll    SODA_COLLECTION_T;
    status  NUMBER;
BEGIN
    -- Open the collection
    coll := dbms_soda.open_Collection('myCollectionName');

    -- Drop the index using name
    status := coll.drop_index('myIndex');
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

3.14 Getting a Data Guide for a Collection with SODA for PL/SQL

You use `SODA_COLLECTION_T` method `get_Data_Guide()` to get a data guide for a collection. A data guide is a JSON document that summarizes the structural and type information of the JSON documents in the collection. It records metadata about the fields used in those documents.

> **Note:**
>
> SODA for PL/SQL support for JSON data guide was added in Oracle Database 18.3. You need that database release or later to use this SODA feature.

Before you can obtain a data guide for your collection you must create a data guide-enabled JSON search index on it. Example 3-26 shows how to do that.

Example 3-28 Getting a Data Guide with SODA for PL/SQL

This example gets a data guide for collection `MyCollectionName` using `SODA_COLLECTION_T` method `get_Data_Guide()`. It then uses SQL/JSON function `json_query` to pretty-print the content of the data-guide document. Finally, it frees the temporary LOB used for the data-guide document.

```
DECLARE
    collection   SODA_COLLECTION_T;
    dataguide    CLOB;
BEGIN
    -- Open the collection.
    collection := dbms_soda.open_Collection('myCollectionName');

    -- Get the data guide for the collection.
    dataguide := collection.get_Data_Guide;
    DBMS_OUTPUT.put_line(json_query(dataguide, '$' pretty));

    -- Important: Free the temporary LOB.
    IF dbms_lob.isTemporary(dataguide) = 1
    THEN
        dbms_lob.freeTemporary(dataguide);
    end if;
END;
/
```

3.15 Handling Transactions with SODA for PL/SQL

As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.

SODA operations `DBMS_SODA.create_collection` and `DBMS_SODA.drop_collection` do *not* automatically commit before or after they perform their action. This differs from the behavior of SQL DDL statements, which commit both before and after performing their action.

One consequence of this is that, before a SODA collection can be dropped, any outstanding write operations to it must be explicitly committed or rolled back — you must explicitly use SQL `COMMIT` or `ROLLBACK`. This is because `DBMS_SODA.drop_collection` does not itself issue commit before it performs its action. In this, the behavior of `DBMS_SODA.drop_collection` differs from that of a SQL `DROP TABLE` statement.

See Also:

- *Oracle Database SQL Language Reference* for information about the SQL `COMMIT` statement
- *Oracle Database SQL Language Reference* for information about the SQL `ROLLBACK` statement
- *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one()`

Example 3-29 Transaction Involving SODA Document Insertion and Replacement

This example shows the use of SQL `COMMIT` and `ROLLBACK` statements in an anonymous PL/SQL block. It opens a SODA collection, inserts a document, and then replaces its content. The combination of the document insertion and document content replacement operations is *atomic*: a single transaction.

```
DECLARE
    collection SODA_COLLECTION_T;
    status NUMBER;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    status := collection.insert_one(
                SODA_Document_T(
                  b_content =>
utl_raw.cast_to_raw('{"a":"aval","b":"bval","c":"cval"}'));
    status := collection.replace_one('key1',
SODA_DOCUMENT_T('{"x":"xval","y":"yval"}'));
    -- Commit the transaction
    COMMIT;
    DBMS_OUTPUT.put_line('Transaction is committed');
-- Catch exceptions and roll back if an error is raised
```

```
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line (SQLERRM);
    ROLLBACK;
    DBMS_OUTPUT.put_line('Transaction has been rolled back');
END;
/
```

Related Topics

- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`.

4

SODA Collection Configuration Using Custom Metadata

SODA collections are highly configurable. You can customize collection metadata, to obtain different behavior from that provided by default.

> **Note:**
>
> You can customize collection metadata to obtain different behavior from that provided by default. However, changing some components requires familiarity with Oracle Database concepts, such as SQL data types. Oracle recommends that you do not change such components unless you have a compelling reason. Because SODA collections are implemented on top of Oracle Database tables (or views), many collection configuration components are related to the underlying table configuration.
>
> For example, if you change the content column type from `BLOB` (the default value) to `VARCHAR2` then you must understand the implications (content size for `VARCHAR2` is limited to 32K bytes, character-set conversion can take place, and so on).

- Getting the Metadata of an Existing Collection
  You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.

> **See Also:**
>
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for general information about SODA document collections and their metadata
> - *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for reference information about collection metadata components

4.1 Getting the Metadata of an Existing Collection

You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `get_metadata()`
> - *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`

Example 4-1 Getting the Metadata of a Collection

This example shows the result of invoking `SODA_COLLECTION_T` method `get_metadata()` on the collection with the default configuration that was created using Example 3-3. (It also uses SQL/JSON function `json_query`, with keyword `PRETTY`, to pretty-print the JSON data obtained.)

```
DECLARE
    collection  SODA_COLLECTION_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    IF collection IS NULL THEN
        DBMS_OUTPUT.put_line('Collection does not exist');
    ELSE
        DBMS_OUTPUT.put_line('Metadata: '
                             || json_query(collection.get_metadata, '$'
PRETTY));
    END IF;
END;
/
```

Example 4-2 Default Collection Metadata

```
{
   "schemaName" : "mySchemaName",
   "tableName" : "myTableName",
   "keyColumn" :
   {
      "name" : "ID",
      "sqlType" : "VARCHAR2",
      "maxLength" : 255,
      "assignmentMethod" : "UUID"
   },
   "contentColumn" :
   {
      "name" : "JSON_DOCUMENT",
      "sqlType" : "BLOB",
```

```
      "compress" : "NONE",
      "cache" : true,
      "encrypt" : "NONE",
      "validation" : "STANDARD"
   },
   "versionColumn" :
   {
     "name" : "VERSION",
     "method" : "SHA256"
   },
   "lastModifiedColumn" :
   {
     "name" : "LAST_MODIFIED"
   },
   "creationTimeColumn" :
   {
      "name" : "CREATED_ON"
   },
   "readOnly" : false
}
```

4.2 Creating a Collection That Has Custom Metadata

To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.

The optional second argument to PL/SQL function `DBMS_SODA.create_collection` is a SODA **collection specification**. It is JSON data that specifies the metadata for the new collection.

If a collection with the same name already exists then it is simply opened and its handle is returned. If the custom metadata provided does not match the metadata of the existing collection then the collection is not opened and an error is raised. (To match, all metadata fields must have the same values.)

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.create_collection`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `get_metadata()`
> - *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`

Example 4-3 Creating a Collection That Has Custom Metadata

This example creates a collection with custom metadata that specifies two metadata columns, named `KEY` (for document keys), and `JSON` (for document content type JSON). The key assignment method is `CLIENT`, and the content-column SQL data type is `VARCHAR2`. The example uses `SODA_COLLECTION_T` method `get_metadata()` to get

the complete metadata from the newly created collection, which it passes to SQL/JSON function `json_query` to pretty-print (using keyword `PRETTY`).

```
DECLARE
    collection SODA_COLLECTION_T;
    metadata VARCHAR2(4000) :=
      '{"keyColumn" : {"name" : "KEY",  "assignmentMethod": "CLIENT" },
        "contentColumn" : { "name" : "JSON", "sqlType": "VARCHAR2" } }';
BEGIN
    collection := DBMS_SODA.create_collection('myCustomCollection',
metadata);
    DBMS_OUTPUT.put_line('Collection specification: ' ||
                         json_query(collection.get_metadata, '$' PRETTY));
END;
/
```

This is the pretty-printed output. The values of fields for `keyColumn` and `contentColumn` that are not specified in the collection specification are defaulted. The values of fields other than those provided in the collection specification (`keyColumn` and `contentColumn`) are also defaulted. The value of field `tableName` is defaulted from the collection name. The value of field `schemaName` is the database schema (user) that is current when the collection is created.

```
Collection specification: {
  "schemaName" : "mySchemaName",
  "tableName" : "myCustomCollection",
  "keyColumn" :
  {
    "name" : "KEY",
    "sqlType" : "VARCHAR2",
    "maxLength" : 255,
    "assignmentMethod" : "CLIENT"
  },

"contentColumn" :
  {
    "name" : "JSON",
    "sqlType" : "VARCHAR2",
    "maxLength" : 4000,
    "validation" : "STANDARD"
  },
  "readOnly" : false
}
```

Related Topics

- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

Index

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

W