Oracle® Database

SODA for C Developers Guide

Release 19c
E96229-02
July 2021

Oracle Database SODA for C Developers Guide, Release 19c

E96229-02

Primary Author: Drew Adams

Contributors: Vijaya Kumar Jitta, Christopher Jones, Maxim Orgiyan, Rajendra Pingte, Srikrishnan Suresh, Anthony Tuininga

Contents

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 C.

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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For more information, see these Oracle resources:

- *Oracle Call Interface Programmer's Guide* for complete information about Oracle Call Interface (OCI), including reference material

- 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 C Prerequisites

SODA for C is an integral part of Oracle Call Interface (OCI) starting with Oracle Database Release 18c (18.3).

To use SODA for C, ensure the following:

- You have Oracle Call Interface 18.3 or later.
- You have Oracle Database 18c or later. To use indexing you need release 18.3 or later.

It is not a requirement, but Oracle recommends that you use AL32UTF8, which implements Unicode UTF-8, as the database character set.

You compile programs that use SODA for C the same way you compile other OCI programs.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for information about building and configuring OCI applications

2

SODA for C Overview

SODA for C is a C API that is part of Oracle Call Interface (OCI). It implements **Simple Oracle Document Access** (SODA). You can use it to perform create, read (retrieve), update, and delete (CRUD) operations on documents of any kind, and you can use it to query JSON documents.

You compile programs that use SODA for C the same way you compile other OCI programs.

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. SODA collections are backed by ordinary Oracle Database tables and views. Because of this, you can generally take advantage of database features for use with the content of SODA documents. For example, you can apply database analytics and reporting to JSON data, and you can include JSON data in aggregation and join operations. In addition, your applications can use database transactions.

SODA interacts with the database transparently. To use SODA you generally do not need a database administrator, and you do not need to program with a database language, such as structured query language (SQL). SODA for C uses OCI and the database to carry out CRUD and query operations, after translating them to Oracle SQL with SQL/JSON operators.

The remaining topics of this document describe various features of SODA for C.

Note:

- This book provides information about using SODA with C applications, and it describes all SODA features currently available for use with C. To use SODA for C 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 C.
- This book does not provide general information about OCI, including reference information about the SODA for C functions and constants. For such information, please consult *Oracle Call Interface Programmer's Guide*.

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 C

How to access SODA for C 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 C
  How to access SODA for C 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 C
  Use OCI function `OCISodaCollCreate()` to create a collection, if you do not care about the details of its configuration. This creates a collection that has the default metadata. To create a collection that is configured in a nondefault way, use function `OCISodaCollCreateWithMetadata()` instead, passing it custom metadata, expressed in JSON.
- Opening an Existing Document Collection with SODA for C
  Use OCI function `OCISodaCollOpen()` to open an existing document collection.
- Checking Whether a Given Collection Exists with SODA for C
  To check for the existence of a collection with a given name, use OCI function `OCISodaCollOpen()`. The function returns `OCI_SUCCESS` if the collection was successfully opened, which means that it exists. If no such collection exists then the collection-handle pointer is `NULL`.
- Discovering Existing Collections with SODA for C
  To discover existing collections, use OCI functions `OCISodaCollList()` and `OCISodaCollGetNext()`.
- Dropping a Document Collection with SODA for C
  To drop a document collection, use OCI function `OCISodaCollDrop()`.
- Creating Documents with SODA for C
  Various ways to create a SODA document are described, along with the components of a document.
- Inserting Documents into Collections with SODA for C
  Various ways to insert a document into a SODA collection are described.
- SODA for C Read and Write Operations
  For all read operations, and for write operations other than insertions, you: (1) allocate an operation-options handle, (2) set some of its attributes to specify a particular operation, and (3) pass the handle to a generic function that performs the operation.
- Finding Documents in Collections with SODA for C
  To find documents in a collection use function `OCISodaFind()`, passing it an operation-options handle that specifies the particular find operation. To find the unique document that has a given key you can alternatively use OCI convenience function `OCISodaFindOneWithKey()`, which does not require an operation-options handle.
- Replacing Documents in a Collection with SODA for C
  You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace

as well as the new, replacement document. It returns that replacement document, but with all of its metadata filled in, as the result document.

- Removing Documents from a Collection with SODA for C
  To remove a document from a collection you can use function `OCISodaRemove()`, passing it an operation-options handle. If you only want to remove one document, specified by its key, then you can alternatively use convenience function `OCISodaRemoveOneWithKey()`. It does not require an operation-options handle — you pass it the key directly.
- Indexing the Documents in a Collection with SODA for C
  Indexing can improve the performance of QBEs. To index the documents in a SODA collection, use function `OCISodaIndexCreate()`, passing it 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 C
  You use function `OCISodaDataGuideGet()` or `OCISodaDataGuideGetWithOpts()` 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 C
  You can handle individual read and write operations, or groups of them, as a database transaction.

3.1 Getting Started with SODA for C

How to access SODA for C 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.

To get started with SODA for C, follow these steps:

1. Ensure that all of the prerequisites have been met for using SODA for C. See SODA for C Prerequisites.
2. Grant database role `SODA_APP` to the database schema (user account) where you intend to store SODA collections. (Replace placeholder *`user`* here by a real account name.)

   ```
   GRANT SODA_APP TO user;
   ```

3. Create a program file containing the C code in Example 3-1, but set variables *`usr`*, *`passwd`*, and *`connstr`* to values appropriate string values for your database account and instance.
4. Compile the file and build an executable program from it as you would for any OCI program.

5. Run the program.

   You can run it just by entering the program name on the command line. For example, if the name is `soda-get-started` then enter that at the command-line prompt:

   ```
   > soda-get-started
   ```

   If you want the program to drop the collection when done with it then pass the argument `drop` to it on the command line:

   ```
   > soda-get-started drop
   ```

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:

- All C code you have that uses SODA for C features *must* first initialize the environment in OCI *object mode*, passing `OCI_OBJECT` as the mode parameter to function `OCIEnvNlsCreate()` here.
- All SODA handles (document, collection, and any others) need to be explicitly freed using function `OCIHandleFree()` when your program no longer needs them. (In particular, a handle for a document with large content can be associated with a lot of memory.)

See Also:

- *Oracle Call Interface Programmer's Guide* for information about building an OCI application
- *Oracle Call Interface Programmer's Guide* for basic information about OCI programming

Example 3-1 Getting Started Run-Through

This example code does the following:

1. Creates an Oracle Call Interface (OCI) environment in object mode, allocates the error handle, and gets a session using function `OCISessionGet()`.
2. Creates and opens a SODA document collection, using the default collection configuration (metadata).

3. Creates a SODA document with some JSON content.
4. Inserts the document into the collection.
5. Gets the inserted document back. Its other components, besides the content, are generated automatically.
6. Prints the unique document key, which is one of the components generated automatically.
7. Finds the document in the collection, providing its key.
8. Prints some of the document components: key, version, last-modified time stamp, creation time stamp, media type, and content.
9. Optionally drops the collection, cleaning up the database table that is used to store the collection and its metadata.
10. Frees all allocated handles.

Whether or not the collection is dropped is decided at runtime. To drop the collection you provide the command-line argument `drop` to the executable program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static sword status;

int main(int argc, char *argv[])
{
  sword        rc = OCI_SUCCESS;
  OCIEnv      *envhp = NULL;
  OCIError    *errhp = NULL;
  OCISvcCtx   *svchp = NULL;
  OCIAuthInfo *authhp = NULL;
  OCISodaColl *collhp = NULL;
  OCISodaDoc  *dochp = NULL;
  boolean      isDropped = FALSE;
  ub4          docFlags = OCI_DEFAULT;
  OraText     *collectionName = (oratext *)"MyJSONCollection";
  OCISodaDoc  *foundDochp = NULL;
  OCISodaDoc  *origDochp = NULL;

  // Document content: JSON data
  char         documentContent[30] = "{\"name\":\"Alexander\"}";

  // Set these variables to strings with the appropriate user name and
password.
  // (Be sure to replace the placeholders user and password used here.)
  OraText      usr[30] = user;
  OraText      passwd[30] = password;

  // Set variable connstr to a string value composed of the host name,
port number, and service name
  // of your database instance.
  // (Be sure to replace placeholders host, port, and service used
here.)
```

```
OraText      connstr[50] = "host:port/service";

OraText     *key = NULL;
ub4          keyLen = 0;
OraText     *content = NULL;
ub4          contentLen = 0;
OraText     *version = NULL;
ub4          versionLen = 0;
OraText     *lastModified = NULL;
ub4          lastModifiedLen = 0;
OraText     *mediaType = NULL;
ub4          mediaTypeLen = 0;
OraText     *createdOn = NULL;
ub4          createdOnLen = 0;

// Set up environment. OCI_OBJECT is required for all SODA C code.
rc = OCIEnvNlsCreate(&envhp,
                     OCI_OBJECT,
                     NULL,
                     NULL,
                     NULL,
                     NULL,
                     0,
                     NULL,
                     0,
                     0);

if (rc != OCI_SUCCESS)
{
  printf ("OCIEnvNlsCreate failed\n");
  goto finally;
}

// Allocate error handle
rc = OCIHandleAlloc((dvoid *) envhp,
                    (dvoid **) &errhp,
                    OCI_HTYPE_ERROR,
                    (size_t) 0,
                    (dvoid **) 0);

if (rc != OCI_SUCCESS)
{
  printf ("OCIHandleAlloc: OCI_HTYPE_ERROR creation failed\n");
  goto finally;
}

// Allocate authentication-information handle
rc = OCIHandleAlloc ((dvoid *)envhp,
                     (dvoid **)&authhp,
                     (ub4)OCI_HTYPE_AUTHINFO,
                     (size_t)0,
                     (dvoid **)0);

if (rc != OCI_SUCCESS)
{
```

```
    printf ("OCIHandleAlloc: OCI_HTYPE_AUTHINFO creation failed\n");
    goto finally;
  }

  // Set variable usr to the user name
  rc = OCIAttrSet ((dvoid *)authhp,
                   (ub4)OCI_HTYPE_AUTHINFO,
                   (dvoid *)usr,
                   (ub4)strlen((char *)usr),
                   (ub4)OCI_ATTR_USERNAME,
                   (OCIError *)errhp);
  if (rc != OCI_SUCCESS)
  {
    printf ("OCIAttrSet: OCI_ATTR_USERNAME failed\n");
    goto finally;
  }

  // Set variable passwd to the password
  rc = OCIAttrSet ((dvoid *)authhp,
                   (ub4)OCI_HTYPE_AUTHINFO,
                   (dvoid *)passwd,
                   (ub4)strlen((char *)passwd),
                   (ub4)OCI_ATTR_PASSWORD,
                   (OCIError *)errhp);
  if (rc != OCI_SUCCESS)
  {
    printf ("OCIAttrSet: OCI_ATTR_PASSWORD failed\n");
    goto finally;
  }

  // Get service handle
  // This provides service and error handles we can use for service
calls
  rc = OCISessionGet ((OCIEnv *)envhp,
                      (OCIError *)errhp,
                      (OCISvcCtx **)&svchp,
                      (OCIAuthInfo *)authhp,
                      (OraText *)connstr,
                      (ub4)strlen((char *)connstr),
                      (OraText *)NULL,
                      (ub4)0,
                      (OraText **)0,
                      (ub4 *)0,
                      (boolean *)0,
                      (ub4)OCI_DEFAULT);

  if (rc != OCI_SUCCESS)
  {
    printf("OCISessionGet failed\n");
    goto finally;
  }

  // Create collection named by the value of variable collectionName,
with default metadata
  rc = OCISodaCollCreate(svchp,
```

```
                          collectionName,
                          (ub4) strlen(collectionName),
                          &collhp,
                          errhp,
                          OCI_DEFAULT);

  if (rc != OCI_SUCCESS)
  {
    printf("OCISodaCollCreate failed\n");
    goto finally;
  }

  // Create a document with content provided by variable documentContent
  rc = OCISodaDocCreate(envhp,
                        documentContent,
                        (ub4) strlen(documentContent),
                        docFlags,
                        &dochp,
                        errhp,
                        OCI_DEFAULT);

  if (rc != OCI_SUCCESS)
  {
    printf("OCISodaDocCreate failed\n");
    goto finally;
  }

  // Because OCISodaInsertAndGet returns the result document as dochp, we
first
  // save the pointer to the original input document handle, which was
returned
  // by OCISodaDocCreate, as origDochp. This lets us free the original
  // document handle later.
  origDochp = dochp;

  // Insert the document into the collection
  rc = OCISodaInsertAndGet(svchp,
                           collhp,
                           &dochp,
                           errhp,
                           OCI_SODA_ATOMIC_COMMIT);

  if (rc != OCI_SUCCESS)
  {
    printf("OCISodaInsertAndGet failed\n");
    goto finally;
  }

  // Get the auto-generated key of the inserted document
  rc = OCIAttrGet((dvoid *) dochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);
```

```
  if (rc != OCI_SUCCESS)
  {
    printf("OCIAttrGet for OCI_ATTR_SODA_KEY failed\n");
    goto finally;
  }

  // Find the document using its key
  printf("Find the document by its auto-generated key %.*s\n", keyLen,
key);
  rc = OCISodaFindOneWithKey(svchp,
                             collhp,
                             key,
                             keyLen,
                             OCI_DEFAULT,
                             &foundDochp,
                             errhp,
                             OCI_DEFAULT);

  if (rc != OCI_SUCCESS)
  {
    printf("OCISodaFindOneWithKey failed\n");
    goto finally;
  }

  // Get and print components of found document
  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);

  if (rc != OCI_SUCCESS)
  {
    printf("OCIAttrGet for OCI_ATTR_SODA_KEY failed\n");
    goto finally;
  }
  printf("Key: %.*s\n", keyLen, key);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &version,
                  &versionLen,
                  OCI_ATTR_SODA_VERSION,
                  errhp);

  if (rc != OCI_SUCCESS)
  {
    printf("OCIAttrGet for OCI_ATTR_SODA_VERSION failed\n");
    goto finally;
  }
  printf("Version: %.*s\n", versionLen, version);

  rc = OCIAttrGet((dvoid *) foundDochp,
```

```
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &lastModified,
                &lastModifiedLen,
                OCI_ATTR_SODA_LASTMOD_TIMESTAMP,
                errhp);

if (rc != OCI_SUCCESS)
{
  printf("OCIAttrGet for OCI_ATTR_SODA_LASTMOD_TIMESTAMP failed\n");
  goto finally;
}
printf("Last-modified: %.*s\n", lastModifiedLen, lastModified);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &createdOn,
                &createdOnLen,
                OCI_ATTR_SODA_CREATE_TIMESTAMP,
                errhp);

if (rc != OCI_SUCCESS)
{
  printf("OCIAttrGet for OCI_ATTR_SODA_CREATE_TIMESTAMP failed\n");
  goto finally;
}
printf("Created: %.*s\n", createdOnLen, createdOn);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &mediaType,
                &mediaTypeLen,
                OCI_ATTR_SODA_MEDIA_TYPE,
                errhp);

if (rc != OCI_SUCCESS)
{
  printf("OCIAttrGet for OCI_ATTR_SODA_MEDIA_TYPE failed\n");
  goto finally;
}
printf("Media Type: %.*s\n", mediaTypeLen, mediaType);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &content,
                &contentLen,
                OCI_ATTR_SODA_CONTENT,
                errhp);

if (rc != OCI_SUCCESS)
{
  printf("OCIAttrGet for OCI_ATTR_SODA_CONTENT failed\n");
  goto finally;
}
printf("Content: %.*s \n", contentLen, content);
```

```
  // Drop the collection if argument "drop" was provided
  if ((argc > 1) && (strcmp(argv[1], "drop") == 0))
  {
    rc = OCISodaCollDrop(svchp,
                         collhp,
                         &isDropped,
                         errhp,
                         OCI_DEFAULT);
    if (rc != OCI_SUCCESS)
    {
      printf("OCISodaCollDrop failed\n");
      goto finally;
    }
    else
    {
      printf("Collection dropped\n");
    }
  }

 finally:

  // Release the session and free all handles
  if (collhp)
    (void ) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);

  if (dochp)
    (void ) OCIHandleFree((dvoid *) dochp, OCI_HTYPE_SODA_DOCUMENT);

  if (origDochp)
    (void ) OCIHandleFree((dvoid *) origDochp, OCI_HTYPE_SODA_DOCUMENT);

  if (foundDochp)
    (void ) OCIHandleFree((dvoid *) foundDochp,
OCI_HTYPE_SODA_DOCUMENT);

  (void ) OCISessionRelease(svchp, errhp, (oratext *)0, 0, OCI_DEFAULT);

  if (authhp)
    (void ) OCIHandleFree ((dvoid *)authhp, (ub4)OCI_HTYPE_AUTHINFO);

  if (errhp)
    (void ) OCIHandleFree((dvoid *) errhp, OCI_HTYPE_ERROR);

  if (svchp)
    (void ) OCIHandleFree((dvoid *) svchp, OCI_HTYPE_SVCCTX);

  if (envhp)
    (void ) OCIHandleFree((dvoid *) envhp, OCI_HTYPE_ENV);
  return rc;
}
```

Related Topics

- Dropping a Document Collection with SODA for C
  To drop a document collection, use OCI function `OCISodaCollDrop()`.

3.2 Creating a Document Collection with SODA for C

Use OCI function `OCISodaCollCreate()` to create a collection, if you do not care about the details of its configuration. This creates a collection that has the default metadata. To create a collection that is configured in a nondefault way, use function `OCISodaCollCreateWithMetadata()` instead, passing it custom metadata, expressed in JSON.

For each of these functions, if a collection with the same name already exists then it is simply opened and its handle is returned. For function `OCISodaCollCreateWithMetadata()`, if the metadata passed to it does not match that 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.)

Example 3-2 uses function `OCISodaCollCreate()` to create a collection that has the default configuration (default metadata). It returns the collection as an `OCISodaColl` handle.

A collection that has the default collection metadata has the following characteristics:

- It can store only JSON documents.
- Each of its documents has these components: key, content, creation time stamp, last-modified time stamp.
- 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:

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

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

> **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 Introduction to Simple Oracle Document Access (SODA)* for reference information about collection metadata components
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaCollCreate()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaCollCreateWithMetadata()`

Example 3-2 Creating a Collection That Has the Default Metadata

This example creates collection `MyCollection` with the default metadata. Note that function `OCISodaCollCreate()` does not, itself, perform a database commit operation.

```
OCISodaColl *collhp = NULL;
OraText     *collectionName = (OraText *)"MyCollection";
rc = OCISodaCollCreate(svchp,
                       (const OraText *)collectionName,
                       (ub4)strlen(collectionName),
                       &collhp,
                       errhp,
                       OCI_DEFAULT);
```

Related Topics

- Getting the Metadata of an Existing Collection
  You can use OCI function `OCIAttrGet()` with attribute `OCI_ATTR_SODA_DESCRIPTOR`, to get *all* of the metadata of a collection at once, as a JSON document. You can also use `OCIAttrGet()` to get individual collection metadata attributes.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to OCI function `OCISodaCollCreateWithMetadata()`.
- Checking Whether a Given Collection Exists with SODA for C
  To check for the existence of a collection with a given name, use OCI function `OCISodaCollOpen()`. The function returns `OCI_SUCCESS` if the collection was successfully opened, which means that it exists. If no such collection exists then the collection-handle pointer is `NULL`.

3.3 Opening an Existing Document Collection with SODA for C

Use OCI function `OCISodaCollOpen()` to open an existing document collection.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaCollOpen()`

Example 3-3 Opening an Existing Document Collection

This example uses OCI function `OCISodaCollOpen()` to open the collection named `MyCollection`. It returns an `OCISodaColl` handle that represents this collection as the value of the fourth parameter (`collhp` in this example). The function return value is `OCI_SUCCESS` for success or `OCI_ERROR` for failure. If the value returned is `OCI_ERROR` then there is no existing collection named `MyCollection`.

```
OCISodaColl *collhp = NULL;
OraText     *collectionName = "MyCollection";
rc = OCISodaCollOpen(svchp,
                     collectionName,
                     (ub4) strlen(collectionName),
                     &collhp,
                     errhp,
                     OCI_DEFAULT);
if (!collhp) printf("Collection %s does not exist\n", collectionName);
```

3.4 Checking Whether a Given Collection Exists with SODA for C

To check for the existence of a collection with a given name, use OCI function `OCISodaCollOpen()`. The function returns `OCI_SUCCESS` if the collection was successfully opened, which means that it exists. If no such collection exists then the collection-handle pointer is `NULL`.

Example 3-3 illustrates this. If `MyCollection` names an existing collection then that collection is opened, and collection-handle `collhp` points to it. If `MyCollection` does not name an existing collection then after invoking function `OCISodaCollOpen()` the value of collection-handle `collhp` is still `NULL`.

Related Topics

- Creating a Document Collection with SODA for C
  Use OCI function `OCISodaCollCreate()` to create a collection, if you do not care about the details of its configuration. This creates a collection that has the default metadata. To create a collection that is configured in a nondefault way, use function `OCISodaCollCreateWithMetadata()` instead, passing it custom metadata, expressed in JSON.

3.5 Discovering Existing Collections with SODA for C

To discover existing collections, use OCI functions `OCISodaCollList()` and `OCISodaCollGetNext()`.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaCollList()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaGetNext()`

Example 3-4 Printing the Names of All Existing Collections

This example uses OCI function `OCISodaCollList()` to obtain a collection cursor (`collectionCursor`). It then iterates over the cursor, printing out each collection name.

```
OCISodaCollCursor *collectionCursor;
OCISodaColl       *collhp;
OraText           *startName = NULL;
ub4                startNameLen = 0;
OraText           *collectionName = NULL;
ub4                collectionNameLen = 0;

rc = OCISodaCollList(svchp,
                     startName,
                     (ub4) strlen(startName),
                     startNameLen,
                     &collectionCursor,
                     errhp,
                     OCI_DEFAULT);

if (rc != OCI_SUCCESS) goto finally;

do
{
  rc = OCISodaCollGetNext(svchp,
                          collectionCursor,
                          &collhp,
                          errhp,
                          OCI_DEFAULT);
  if (rc == OCI_NO_DATA || rc == OCI_INVALID_HANDLE || rc == OCI_ERROR)
goto finally;

  rc = OCIAttrGet((dvoid *) collhp,
                  OCI_HTYPE_SODA_COLLECTION,
                  (dvoid *) &collectionName,
                  &collectionNameLen,
                  OCI_ATTR_SODA_COLL_NAME,
                  errhp);
```

```
  if (rc != OCI_SUCCESS) goto finally;
  printf("%s\n", collectionName);
  if (collhp) OCIHandleFree((dvoid *) collhp, (ub4)
OCI_HTYPE_SODA_COLLECTION));
}
while(1);

finally:
if (collectionCursor) OCIHandleFree((dvoid *) collectionCursor,
(ub4)OCI_HTYPE_SODA_CURSOR);
```

In this example, `startName` is `NULL`, and `startNameLen` is `0`. As a result, the cursor iterates over *all collections in the database*.

Alternatively, you could iterate over only a subset of the existing collections. For that, you could set `startName` to an existing collection name, such as `"myCollectionB"`, and set `startNameLen` to its string length. The cursor would then iterate over only that collection and the collections whose names come after that collection name alphabetically. The collections would be iterated over in alphabetic order of their names.

For example, if the existing collections are `"myCollectionA"`, `"myCollectionB"`, and `"myCollectionC"`, and if `startName` is `"myCollectionB"`, then the cursor iterates over `"myCollectionB"` and `"myCollectionC"`, in that order.

3.6 Dropping a Document Collection with SODA for C

To drop a document collection, use OCI function `OCISodaCollDrop()`.

Unlike Oracle SQL statement `DROP TABLE`, function `OCISodaCollDrop()` does not implicitly perform a commit operation before and after it drops the collection. To complete the collection removal you must explicitly commit all uncommitted writes to the collection *before* invoking `OCISodaCollDrop()`.

Dropping a collection using a collection handle does *not* free the handle. You must use OCI function `OCIHandleFree()` to free a handle.

> **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 handles 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 handles, then there is a risk that a stale collection is accessed, and *no error is raised* in this case.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaCollDrop()`

Example 3-5 Dropping a Document Collection

This example uses OCI function `OCISodaCollDrop()` to drop a collection. (Variable `collhp` is assumed to point to an existing collection — an `OCISodaColl` instance).

If the collection cannot be dropped because of uncommitted write operations then an error is returned. If the collection is dropped successfully, the value of out parameter `dropStatus` is `TRUE`; otherwise it is `FALSE`.

If the collection-handle argument (`collhp` in this example) no longer references an existing collection then *no error* is returned, but `dropStatus` is `FALSE` after the invocation of `OCISodaCollDrop()`.

```
boolean dropStatus = FALSE;
rc = OCISodaCollDrop(svchp, collhp, &dropStatus, errhp, OCI_DEFAULT);
```

Related Topics

- Handling Transactions with SODA for C
  You can handle individual read and write operations, or groups of them, as a database transaction.
- Inserting Documents into Collections with SODA for C
  Various ways to insert a document into a SODA collection are described.
- Replacing Documents in a Collection with SODA for C
  You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace as well as the new, replacement document. It returns that

replacement document, but with all of its metadata filled in, as the result document.

3.7 Creating Documents with SODA for C

Various ways to create a SODA document are described, along with the components of a document.

SODA for C represents a document using a **`OCISodaDoc`** handle. This is a *carrier* of document content and other document components, such as the document key. Document components are handle attributes.

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 can *create a document* in these ways:

- By invoking a OCI function that is specifically designed to create a document: **`OCISodaDocCreate()`**, **`OCISodaDocCreateWithKey()`**, or **`OCISodaDocCreateWithKeyAndMType()`**.

  Example 3-6 and Example 3-7 illustrate this. They both create a document handle. In each case the media type for the created document defaults to `"application/json"`, and the other document components default to `NULL`.

- By invoking function `OCIHandleAlloc()` with handle type **`OCI_HTYPE_SODA_DOCUMENT`**, to create an empty document (handle).

  Example 3-8 illustrates this.

You can use function `OCIAttrSet()` to define (set) document components (document-handle attributes), whether or not they already have values.

If you use the second approach (`OCIHandleAlloc()`) to create a document then you must invoke function `OCIAttrSet()` to set the *content* component. If you intend the document to be written to a collection with client-assigned keys then you must also invoke it to set the *key*. If you intend the document to have non-JSON content then you must also invoke it to set the *media type*.

However you create a document, you can reuse the handle for multiple document operations. For example, you can change the content or other components, passing the same handle to different write operations.

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. Use function `OCISodaDocCreate()` if the key is to be automatically generated; otherwise, supply the key (as parameter `key`) to `OCISodaDocCreateWithKey()`, or `OCISodaDocCreateWithKeyAndMType()`.

Use function `OCISodaDocCreateWithKeyAndMType()` if you want to provide the document media type (otherwise, it defaults to `"application/json"`). This can be useful for creating *non*-JSON documents (using a media type other than `"application/json"`).

Whichever document-creation function you use, invoking it sets the document 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 `null`.

You *get* document components using OCI function `OCIAttrGet()`, which is the same way you get the value of any handle attribute. You pass the type of the component you want to get to `OCIAttrGet()` as the fifth argument.

Table 3-1 Document Handle Attributes (Document Components)

Attribute	Description
`OCI_ATTR_SODA_KEY`	The unique *key* for the document.
`OCI_ATTR_SODA_CREATE_TIMESTAMP`	The *creation time stamp* for the document.
`OCI_ATTR_SODA_LASTMOD_TIMESTAMP`	The *last-modified time stamp* for the document.
`OCI_ATTR_SODA_MEDIA_TYPE`	The *media type* for the document.
`OCI_ATTR_SODA_VERSION`	The document *version*.
`OCI_ATTR_SODA_CONTENT`	The document *content*.

Immediately after you create a document, `OCIAttrGet()` returns these values for components:

- Values explicitly provided to the document-creation function
- `"application/json"`, for `OCI_ATTR_SODA_MEDIA_TYPE`, if the media type was not provided to the creation function
- `NULL` for other components

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 Call Interface Programmer's Guide* for information about OCI function `OCISodaDocCreate()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaDocCreateWithKey()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaDocCreateWithKeyAndMType()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCIHandleAlloc()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCIAttrSet()`

Example 3-6 Creating a Document with JSON Content

This example uses `OCISodaDocCreate()` to create a document handle and fill the document with content. It then frees the document handle.[1]

```
OCISodaDoc *dochp =  NULL;
OraText    *documentContent = "{\"name\":\"Alexander\"}";
ub4         docFlags = OCI_DEFAULT;

rc = OCISodaDocCreate(envhp,
                      documentContent,
                      (ub4) strlen(documentContent),
                      docFlags,
                      &dochp,
                      errhp,
                      OCI_DEFAULT)
// Make further use of handle dochp...
if (dochp) OCIHandleFree((dvoid *) dochp, (ub4) OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-7 Creating a Document with Document Key and JSON Content

This example is similar to Example 3-6, but it uses `OCISodaDocCreateWithKey()`, providing the document key (`myKey`) as well as the document content. It then gets and prints the non-null document components that were set by `OCISodaDocCreateWithKey()`: the key, the content and the media type. It then frees the document handle.

```
OCISodaDoc *dochp =  NULL;
OraText    *documentContent = "{\"name\":\"Alexander\"}";
OraText    *key = "myKey";
```

[1] The handle is freed here immediately, just as a reminder to free it when you are done with it (the same as any other handle). In practice you would make use of the handle in some way before freeing it.

```
ub4         docFlags = OCI_DEFAULT;
sword       rc = OCI_SUCCESS;
OraText    *finalKey;
ub4         finalKeyLen = 0;
OraText    *finalContent;
ub4         finalContentLen = 0;
OraText    *media;
ub4         mediaLen = 0;

rc = OCISodaDocCreateWithKey(envhp,
                             documentContent,
                             (ub4) strlen(documentContent),
                             key,
                             (ub4) strlen(key),
                             docFlags,
                             &dochp,
                             errhp,
                             OCI_DEFAULT)

if (rc != OCI_SUCCESS) goto finally;

// Get and print the key, content and media type, which were set by
OCISodaDocCreateWithKey().
OCIAttrGet((dvoid *) dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *) &finalKey,
           &finalKeyLen,
           OCI_ATTR_SODA_KEY,
           errhp);
printf ("Key: %.*s\n", finalKeyLen, finalKey);

OCIAttrGet((dvoid *) dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *) &finalContent,
           &finalContentLen,
           OCI_ATTR_SODA_CONTENT,
           errhp);
printf ("Content: %.*s\n", finalContentLen, finalContent);

OCIAttrGet((dvoid *)dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *) &media,
           &mediaLen,
           OCI_ATTR_SODA_MEDIA_TYPE,
           errhp);
printf ("Media type: %.*s\n", mediaLen, media);

finally:
if (dochp) OCIHandleFree((dvoid *) dochp, (ub4)
OCI_HTYPE_SODA_DOCUMENT);
```

This is the printed output:

```
Key: myKey
Content: {"name" : "Alexander"}
Media type: application/json
```

Example 3-8 Creating an Empty Document and Then Defining Components

```
sword       rc = OCI_SUCCESS;
OCISodaDoc *dochp = NULL;
OraText    *documentContent= "{\"name\":\"Alexander\"}";

rc = OCIHandleAlloc((void *) envhp,
                    (void **) &dochp,
                    OCI_HTYPE_SODA_DOCUMENT,
                    (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

rc = OCIAttrSet(dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                documentContent,
                (ub4) strlen(documentContent),
                OCI_ATTR_SODA_CONTENT,
                errhp);
finally: ...
```

Related Topics

- Inserting Documents into Collections with SODA for C
  Various ways to insert a document into a SODA collection are described.
- Finding Documents in Collections with SODA for C
  To find documents in a collection use function `OCISodaFind()`, passing it an operation-options handle that specifies the particular find operation. To find the unique document that has a given key you can alternatively use OCI convenience function `OCISodaFindOneWithKey()`, which does not require an operation-options handle.
- Replacing Documents in a Collection with SODA for C
  You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace as well as the new, replacement document. It returns that replacement document, but with all of its metadata filled in, as the result document.
- Removing Documents from a Collection with SODA for C
  To remove a document from a collection you can use function `OCISodaRemove()`, passing it an operation-options handle. If you only want to remove one document, specified by its key, then you can alternatively use convenience function `OCISodaRemoveOneWithKey()`. It does not require an operation-options handle — you pass it the key directly.

3.8 Inserting Documents into Collections with SODA for C

Various ways to insert a document into a SODA collection are described.

If you have created a document handle, you can use function `OCISodaInsert()` or `OCISodaInsertAndGet()` to insert the document into a collection. These functions create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key. These functions take a document handle as one of their arguments.

For convenience, you can alternatively use function `OCISodaInsertWithCtnt()` or `OCISodaInsertAndGetWithCtnt()` to insert a document without having created a document handle. You provide only the content and (optionally) the key for the document. (The key is needed only when inserting into a collection that has client-assigned keys.)

If the target collection is configured for documents that have creation and last-modified time-stamp components then all of the document-insertion functions automatically set these components. If the collection is configured to generate document versions automatically then the insertion functions also set the version component. (The default collection configuration provides both time-stamp components and the version component.)

In addition to inserting the document, functions `OCISodaInsertAndGet()` and `OCISodaInsertAndGetWithCtnt()` return a result document. The result document contains the generated document components, such as the key, version, created-on timestamp, and last-modified timestamp. It 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 return an error.

See Also:

- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaInsert()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaInsertAndGet()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaInsertWithCtnt()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaInsertAndGetWithCtnt()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaBulkInsert()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaBulkInsertAndGet()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaBulkInsertWithCtnt()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaBulkInsertAndGetWithCtnt()`

Example 3-9 Inserting a Document into a Collection

This example creates a document and inserts it into a collection using function `OCISodaInsert()`. The use of mode parameter `OCI_SODA_ATOMIC_COMMIT` ensures that the insertion and any other outstanding operations are committed.

```
OCISodaDoc *dochp = NULL;
OraText    *documentContent = "{\"name\":\"Alexander\"}";

rc = OCISodaDocCreate(envhp,
                      documentContent,
                      (ub4) strlen(documentContent),
                      OCI_DEFAULT,
                      &dochp,
                      errhp,
                      OCI_DEFAULT);

if (rc != OCI_SUCCESS) goto finally:

rc = OCISodaInsert(svchp,
                   collhp,
                   dochp,
                   errhp,
                   OCI_SODA_ATOMIC_COMMIT);

finally: ...
```

Example 3-10 Inserting a Document into a Collection and Getting the Result Document

This example creates a document and inserts it into a collection using function `OCISodaInsertAndGet()`, which also returns the result document, after insertion. The example then gets (and prints) each of the generated components from that result document (which contains them): the creation time stamp, the last-modified time stamp, the media type, and the version. To obtain each of these components it uses function `OCIAttrGet()`, passing the type of the component: `OCI_ATTR_SODA_CREATE_TIMESTAMP`, `OCI_ATTR_SODA_LASTMOD_TIMESTAMP`, `OCI_ATTR_SODA_MEDIA_TYPE`, and `OCI_ATTR_SODA_VERSION`.

```
sword rc = OCI_SUCCESS;

OraText    *key = "myKey1";
OraText    *documentContent = "{\"name\":\"Alexander\"}";
ub4         docFlags = OCI_DEFAULT;
OCISodaDoc *dochp = NULL;
OCISodaDoc *origDochp = NULL;
OraText    *resultKey;
ub4         resultKeyLen = 0;
OraText    *resultCreatedOn;
ub4         resultCreatedOnLen = 0;
OraText    *resultLastModified;
ub4         resultLastModifiedLen = 0;
OraText    *resultVersion;
ub4         resultVersionLen = 0;
OraText    *resultMedia;
ub4         resultMediaLen = 0;

// Create a document with key "myKey1"
rc = OCISodaDocCreateWithKey(envhp,
                             documentContent,
                             (ub4) strlen(documentContent),
                             key,
                             (ub4) strlen(key),
                             docFlags,
                             &dochp,
                             errhp,
                             OCI_DEFAULT);

if (rc != OCI_SUCCESS) goto finally;

// Insert the document into a collection.

// collhp is a collection-handle pointer.  We assume the collection it
// points to was configured to use client-assigned keys.

// Because OCISodaInsertAndGet returns the result document as dochp, we
first
// save the pointer to the original input document handle, which is
returned by
// OCISodaDocCreateWithKey, as origDochp.  This lets us free the
original
// document handle later.
```

```
origDochp = dochp;

rc = OCISodaInsertAndGet(svchp,
                         collhp,
                         &dochp,
                         errhp,
                         OCI_SODA_ATOMIC_COMMIT);

if (rc != OCI_SUCCESS) goto finally;

// Print some components of the result document.  (For brevity we omit
checking
// for a return value of OCI_SUCCESS in all OCIAttrGet() calls here.)

OCIAttrGet((dvoid *)dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *)&resultCreatedOn,
           &resultCreatedOnLen,
           OCI_ATTR_SODA_CREATE_TIMESTAMP,
           errhp);
printf ("Created-on time stamp: %.*s\n", resultCreatedOnLen,
resultCreatedOn);

OCIAttrGet((dvoid *)dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *)&resultLastModified,
           &resultLastModifiedLen,
           OCI_ATTR_SODA_LASTMOD_TIMESTAMP,
           errhp);
printf ("Last-modified time stamp: %.*s\n", resultLastModifiedLen,
resultLastModified);

OCIAttrGet((dvoid *)dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *)&resultVersion,
           &resultVersionLen,
           OCI_ATTR_SODA_VERSION,
           errhp);
printf ("Version: %.*s\n", resultVersionLen, resultVersion);

OCIAttrGet((dvoid *)dochp,
           OCI_HTYPE_SODA_DOCUMENT,
           (dvoid *)&resultMedia,
           &resultMediaLen,
           OCI_ATTR_SODA_MEDIA_TYPE,
           errhp);
printf ("Media type: %.*s\n", resultMediaLen, resultMedia);

finally:

// Free the document handles
if (origDochp) OCIHandleFree((dvoid *) origDochp, (ub4)
OCI_HTYPE_SODA_DOCUMENT);
```

```
if (dochp) OCIHandleFree((dvoid *) dochp, (ub4)
OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-11 Inserting a Document into a Collection Without Providing a Handle

This example uses function `OCISodaInsertWithCtnt()` to insert a document into a collection without providing a document handle. Only the document key and content are provided as arguments.

Here we assume that we are inserting the document into a collection that is configured with client-assigned keys. If you instead insert a document into a collection configured for auto-generated keys then pass `NULL` as the key argument and `0` as the key-length argument (which immediately follows the key argument).

```
OraText *documentContent = "{\"name\":\"Hannibal\"}";
OraText *key = "myKey2";

rc = OCISodaInsertWithCtnt(svchp,
                           collhp,
                           key,
                           (ub4) strlen(key),
                           (void *)documentContent,
                           (ub4) strlen(documentContent),
                           errhp,
                           OCI_SODA_ATOMIC_COMMIT);
```

Related Topics

- Handling Transactions with SODA for C
  You can handle individual read and write operations, or groups of them, as a database transaction.
- Dropping a Document Collection with SODA for C
  To drop a document collection, use OCI function `OCISodaCollDrop()`.
- Replacing Documents in a Collection with SODA for C
  You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace as well as the new, replacement document. It returns that replacement document, but with all of its metadata filled in, as the result document.

3.9 SODA for C Read and Write Operations

For all read operations, and for write operations other than insertions, you: (1) allocate an operation-options handle, (2) set some of its attributes to specify a particular operation, and (3) pass the handle to a generic function that performs the operation.

These are the read-operation functions:

- `OCISodaFindOne()` — Find and return at most one document.
- `OCISodaFind()` — Find multiple documents and return a cursor to them.

- `OCISodaDocCount()` — Find multiple documents and return the number of documents found.

These are the write-operation functions:

- `OCISodaReplOne()` — Replace one document.
- `OCISodaReplOneAndGet()` — Replace one document and return the result document.
- `OCISodaRemove()` — Remove multiple documents.

You use function `OCIHandleAlloc()` to allocate an empty operation-options handle:

```
OCISodaOperationOptions *opthp;
// Create an empty operation options handle
rc = OCIHandleAlloc((void *) envhp,
                    (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS,
                    (size_t) 0,
                    (dvoid **) 0);
```

You use function `OCIAttrSet()` to set a single attribute of an operation-options handle. For example, this sets attribute `filter` with value `{"name:"Ruth"}`:

```
OraText * filter = "{\"name\" : \"Ruth\"}";
// Set the filter on the operation options handle
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                filter,
                strlen(filter),
                OCI_ATTR_SODA_FILTER,
                errhp);
```

There is no attribute that represents multiple document keys. For an operation that involves multiple keys you use function `OCISodaOperKeysSet()` to set them.

> **Note:**
>
> If you use function `OCIAttrSet()` to set attribute `OCI_ATTR_SODA_KEY` on an operation-options handle, and you also use function `OCISodaOperKeysSet()` to set multiple keys on the same handle, then only the latest of the two settings takes effect. The effect of the first function invoked is overridden by the effect of the second.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about the SODA attributes that can be set on an operation-options handle
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaOperKeySet()`

3.10 Finding Documents in Collections with SODA for C

To find documents in a collection use function `OCISodaFind()`, passing it an operation-options handle that specifies the particular find operation. To find the unique document that has a given key you can alternatively use OCI convenience function `OCISodaFindOneWithKey()`, which does not require an operation-options handle.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaFind()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaFindOneWithKey()`

Example 3-12 Finding All Documents in a Collection

This example first obtains 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 components of each document, as a string.

```
OraText                 *key = NULL;
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
OraText                 *version = NULL;
ub4                      versionLen = 0;
OraText                 *lastModified = NULL;
ub4                      lastModifiedLen = 0;
OraText                 *mediaType = NULL;
ub4                      mediaTypeLen = 0;
OraText                 *createdOn = NULL;
ub4                      createdOnLen = 0;
ub4                      findFlags = OCI_DEFAULT;

OCISodaDocCursor        *cursorhp = NULL;
OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
```

```
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Find all documents in the collection.
//
// Because the operation-options handle (opthp) is empty, no conditions
// are set on the find operation, so all documents are returned.
//
// collhp is an OCISodaColl pointer, representing an open collection.
//
// cursorhp is a OCISodaDocCursor pointer to a returned cursor over the
// resulting document set.
rc = OCISodaFind(svchp,
                 collhp,
                 opthp,
                 findFlags,
                 &cursorhp,
                 errhp,
                 OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

// Fetch each document from the cursor, and print all of its components.
while (OCISodaDocGetNext(svchp,
                         cursorhp,
                         &foundDochp,
                         errhp,
                         OCI_DEFAULT)
       == OCI_SUCCESS)
{
  // Get and print components of found document.
  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Key: %.*s\n", keyLen, key);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &version,
                  &versionLen,
                  OCI_ATTR_SODA_VERSION,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Version: %.*s\n", versionLen, version);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &lastModified,
                  &lastModifiedLen,
                  OCI_ATTR_SODA_LASTMOD_TIMESTAMP,
  if (rc != OCI_SUCCESS) goto finally;
```

```
  printf("Last-modified: %.*s\n", lastModifiedLen, lastModified);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &createdOn,
                  &createdOnLen,
                  OCI_ATTR_SODA_CREATE_TIMESTAMP,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Created: %.*s\n", createdOnLen, createdOn);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &mediaType,
                  &mediaTypeLen,
                  OCI_ATTR_SODA_MEDIA_TYPE,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Media Type: %.*s\n", mediaTypeLen, mediaType);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &content,
                  &contentLen,
                  OCI_ATTR_SODA_CONTENT,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Content: %.*s \n", contentLen, content);

  // Important: free document handle before fetching next document.
  // This releases memory associated with the current document.
  if (foundDochp)
    (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
}

finally:

// Free all handles.
if (cursorhp)
  (void) OCIHandleFree((dvoid *) cursorhp, OCI_HTYPE_SODA_DOC_CURSOR);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
```

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

This example sets up an operations options handle with the given UUID key (`E914016C41174F6CBF7C877C7F9EB4C2`), which it passes to function `OCISodaFindOne()` to find the document with that key.

After finding the document, the example uses function `OCIAttrGet()` to retrieve the document key and content, and then it prints them. Finally, it frees the document handles that were allocated for the collection, document, and operations options.

As an alternative to setting the key attribute on an operation-options handle and using `OCISodaFindOne()`, you can use convenience function `OCISodaFindOneWithKey`. It accepts a key argument directly, in place of an operation-options handle.

```
OraText                 *key = NULL;
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
ub4                      findFlags = OCI_DEFAULT;

OraText                 *inKey = "E914016C41174F6CBF7C877C7F9EB4C2";

OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the key of the document we want to find, on the operation-options
handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Find the document with the key, by way of the operation-options handle.
//
// collhp is an OCISodaColl pointer, representing an open collection.
rc = OCISodaFindOne(svchp,
                    collhp,
                    opthp,
                    findFlags,
                    &foundDochp,
                    errhp,
                    OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

// Get and print components of found document.
rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &key,
                &keyLen,
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
```

```
printf("Key: %.*s\n", keyLen, key);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &content,
                &contentLen,
                OCI_ATTR_SODA_CONTENT,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Content: %.*s \n", contentLen, content);

finally:

// Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

Example 3-14 Finding Multiple Documents with Specified Document Keys

This example finds three documents using their keys. The keys and their string lengths, as arrays, and the number of keys (3) are passed to function `OCISodaOperKeysSet()`, which sets up the operation-options handle appropriately. (You cannot set multiple keys and their lengths using standard function **`OCIAttrSet()`**.) The example then invokes function `OCISodaFind()`, passing it the handle.

This example uses function `OCISodaFind` to find three documents using their keys. The keys and their string lengths, as arrays, and the number of keys (3) are passed to function `OCISodaOperKeysSet()`, which sets up the operation-options handle with this information. (You cannot set multiple keys and their lengths using standard function **`OCIAttrSet()`**.)

> **Note:**
>
> If you use function `OCIAttrSet()` to set attribute `OCI_ATTR_SODA_KEY` on an operation-options handle, and you also use function `OCISodaOperKeysSet()` to set multiple keys on the same handle, then only the latest of the two settings takes effect. The effect of the first function invoked is overridden by the effect of the second.

```
OraText                *key = NULL;
ub4                     keyLen = 0;
OraText                *content = NULL;
ub4                     contentLen = 0;
ub4                     findFlags = OCI_DEFAULT;

OraText                *keys[3] = {"6B67A10BC6EB4FB7BFA1ECE7E697C507",
                                   "9195598AA9FB4F1CBFA376F35BF78588",
                                   "7FD55EED38BE4F70BF327F8132394E8B"};
```

```
ub4                      keyLengths[3];
int                      i = 0;

OCISodaDocCursor        *cursorhp = NULL;
OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Fill array of key lengths.
for(i=0; i<3; i++)
  keyLengths[i] = strlen(keys[i]);

// Set keys and their lengths on the operation-options handle.
//
// You cannot set keys and their lengths using standard function
OCIAttrSet().
// Use function OCISodaOperKeysSet().
rc = OCISodaOperKeysSet(opthp,
                        keys,
                        keyLengths,
                        3,
                        errhp,
                        OCI_DEFAULT);

if (rc != OCI_SUCCESS) goto finally;

// Find documents in collection that match the keys set in operation-options
handle.
//
// collhp is an OCISodaColl pointer, representing an open collection.
//
// cursorhp is a OCISodaDocCursor pointer to a returned cursor over the
// resulting document set.
rc = OCISodaFind(svchp,
                 collhp,
                 opthp,
                 findFlags,
                 &cursorhp,
                 errhp,
                 OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

// Fetch each document from the cursor, and its key and content.
while (OCISodaDocGetNext(svchp,
                         cursorhp,
                         &foundDochp,
                         errhp,
                         OCI_DEFAULT)
       == OCI_SUCCESS)
{
```

```
  // Get and print components of found document.
  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Key: %.*s\n", keyLen, key);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &content,
                  &contentLen,
                  OCI_ATTR_SODA_CONTENT,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Content: %.*s \n\n", contentLen, content);

  // Important: Free the document handle before fetching the next
document.
  // This releases memory associated with the current document.
  if (foundDochp)
    (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
}

finally:

// Free all handles
if (cursorhp)
  (void) OCIHandleFree((dvoid *) cursorhp, OCI_HTYPE_SODA_DOC_CURSOR );
if (opthp)
  (void) OCIHandleFree((dvoid *) cursorhp,
OCI_HTYPE_SODA_OPER_OPTIONS );
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-15 Finding Documents with a Filter Specification

Function `OCISodaFind()` provides a powerful way to filter JSON documents in a collection. To use it you pass an operation-options handle that specifies attribute `OCI_ATTR_SODA_FILTER` as 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 C.

This example sets operation-options handle attribute `OCI_ATTR_SODA_FILTER` to a filter that specifies JSON documents whose `name` field has value `"Alexander"`. It then uses

the operation-options handle to find the documents that match that filter. Finally, it prints the key and content of each found document.

```
OraText                 *key = NULL;
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
ub4                      findFlags = OCI_DEFAULT;

OraText                 *filter = "{ \"name\" : \"Alexander\"}";

OCISodaDocCursor        *cursorhp = NULL;
OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the filter (query-by-example, or QBE) on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                filter,
                strlen(filter),
                OCI_ATTR_SODA_FILTER,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Find all documents in collection that match filter set in operation-
options handle.
//
// collhp is an OCISodaColl pointer, representing an open collection.
//
// cursorhp is a OCISodaDocCursor pointer to a returned cursor over the
// resulting document set.
rc = OCISodaFind(svchp,
                       collhp,
                       opthp,
                       findFlags,
                       &cursorhp,
                       errhp,
                       OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

// Fetch each document from the cursor, and print its key and content.
while (OCISodaDocGetNext(svchp,
                         cursorhp,
                         &foundDochp,
                         errhp,
                         OCI_DEFAULT)
       == OCI_SUCCESS)
{
  // Get and print key and content of found document.
```

```
  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Key: %.*s\n", keyLen, key);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &content,
                  &contentLen,
                  OCI_ATTR_SODA_CONTENT,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Content: %.*s \n\n", contentLen, content);

  // Important: Free the document handle before fetching next document.
  // This releases memory associated with the current document.
  if (foundDochp)
    (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
}

finally:

// Free all handles
if (cursorhp)
  (void) OCIHandleFree((dvoid *) cursorhp, OCI_HTYPE_SODA_DOC_CURSOR );
if (opthp)
  (void) OCIHandleFree((dvoid *) cursorhp,
OCI_HTYPE_SODA_OPER_OPTIONS );
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-16 Finding Documents with a Filter Specification and Pagination

This example uses function `OCISodaFind()` in a pagination query. It passes an operation-options handle that specifies attribute `OCI_ATTR_SODA_FILTER` as a QBE, as well as attributes `OCI_ATTR_SODA_SKIP` (the number of documents to skip) and `OCI_ATTR_SODA_LIMIT` (the maximum number of documents to return). Except for specifying pagination (skip and limit) this example is the same as Example 3-15.

```
OraText                *key = NULL;
ub4                     keyLen = 0;
OraText                *content = NULL;
ub4                     contentLen = 0;
ub4                     findFlags = OCI_DEFAULT;

OraText                *filter = "{ \"name\" : \"Alexander\"}";
ub4                     skip = 1000;
ub4                     limit = 100;
```

```
OCISodaDocCursor        *cursorhp = NULL;
OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the filter (query-by-example, or QBE) on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                filter,
                strlen(filter),
                OCI_ATTR_SODA_FILTER,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Set the number of documents to skip on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                &skip,
                0,
                OCI_ATTR_SODA_SKIP,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Set the limit of the number of documents to return on the operation-
options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                &limit,
                0,
                OCI_ATTR_SODA_LIMIT,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Find all documents in collection that match filter set in operation-
options handle.
// Honor  skip and limit values set in the handle.
//
// collhp is an OCISodaColl pointer, representing an open collection.
//
// cursorhp is a OCISodaDocCursor pointer to a returned cursor over the
// resulting document set.
rc = OCISodaFind(svchp,
                 collhp,
                 opthp,
                 findFlags,
                 &cursorhp,
                 errhp,
                 OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;
```

```
// Fetch each document from the cursor, and print its key and content.
while (OCISodaDocGetNext(svchp,
                         cursorhp,
                         &foundDochp,
                         errhp,
                         OCI_DEFAULT)
       == OCI_SUCCESS)
{
  // Get and print components of found document.
  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &key,
                  &keyLen,
                  OCI_ATTR_SODA_KEY,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Key: %.*s\n", keyLen, key);

  rc = OCIAttrGet((dvoid *) foundDochp,
                  OCI_HTYPE_SODA_DOCUMENT,
                  (dvoid *) &content,
                  &contentLen,
                  OCI_ATTR_SODA_CONTENT,
                  errhp);
  if (rc != OCI_SUCCESS) goto finally;
  printf("Content: %.*s \n\n", contentLen, content);

  // Important: Free the document handle before fetching the next
document.
  // This releases memory associated with the current document.
  if (foundDochp)
    (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
}

finally:

// Free all handles
if (cursorhp)
  (void) OCIHandleFree((dvoid *) cursorhp, OCI_HTYPE_SODA_DOC_CURSOR );
if (opthp)
  (void) OCIHandleFree((dvoid *) cursorhp,
OCI_HTYPE_SODA_OPER_OPTIONS );
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-17 Finding a Particular Version of a Document

This example uses function `OCISodaFindOne()` with an operation-options handle that specifies the version, as well as the key, of the document to find.

When specifying the document version you typically specify the key as well. But you can specify the version along with a filter, provided the filter specifies at most one document in the collection.

```
OraText                 *key = NULL;
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
OraText                 *version = NULL;
ub4                      versionLen = 0;
ub4                      findFlags = OCI_DEFAULT;

OraText                 *inKey = "E914016C41174F6CBF7C877C7F9EB4C2";
OraText                 *inVersion =
  "7CCEF2F54035DE9A9D64653645DBEF7E61B92142F2E41B3F6144262A5F7BC054";

OCISodaDocCursor        *cursorhp = NULL;
OCISodaDoc              *foundDochp = NULL;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the key on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Set the version on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inVersion,
                strlen(inVersion),
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Find document that matches key and version set on operation-options
handle.
rc = OCISodaFindOne(svchp,
                          collhp,
                          opthp,
                          findFlags,
                          &foundDochp,
                          errhp,
                          OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;
```

```
// Get the found document and print its key, version, and content.
rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &key,
                &keyLen,
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Key: %.*s\n", keyLen, key);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &version,
                &versionLen,
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Version: %.*s\n", versionLen, version);

rc = OCIAttrGet((dvoid *) foundDochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &content,
                &contentLen,
                OCI_ATTR_SODA_CONTENT,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Content: %.*s \n", contentLen, content);

finally:

// Free all handles
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (foundDochp)
  (void) OCIHandleFree((dvoid *) foundDochp, OCI_HTYPE_SODA_DOCUMENT);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

Example 3-18 Counting the Number of Documents Found

This example uses function `OCISodaDocCount()` to get a count of all of the documents in a collection that satisfy a given filter specification.

```
OraText                 *filter = "{ \"name\" : \"Alexander\"}";
ub8                      count = 0;
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the filter (query-by-example, or QBE) on the operation-options
handle.
```

```
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                filter,
                strlen(filter),
                OCI_ATTR_SODA_FILTER,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Number of documents that match filter set on  operation-options handle is
returned as count.
rc = OCISodaDocCount(svchp,
                     collhp,
                     opthp ,
                     &count,
                     errhp,
                     OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;
printf ("Number of matching documents: %d\n", count);

finally:

// Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION );
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS );
```

Related Topics

- SODA for C Read and Write Operations
  For all read operations, and for write operations other than insertions, you: (1) allocate an operation-options handle, (2) set some of its attributes to specify a particular operation, and (3) pass the handle to a generic function that performs the operation.

3.11 Replacing Documents in a Collection with SODA for C

You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace as well as the new, replacement document. It returns that replacement document, but with all of its metadata filled in, as the result document.

Function `OCISodaReplOne()` is the same as `OCISodaReplOneAndGet()`, except that it does not return the result document with completed metadata.

These are the most generic document-replacement functions. There are also other, convenience functions for more specific use cases.

You can use these convenience functions if only the document *content* is to be replaced. Instead of passing them a replacement document, you pass just the new (JSON) content as a textual argument.

- `OCISodaReplOneAndGetWithCtnt()`
- `OCISodaReplOneWithCtnt()`

You can use these convenience functions if only the document *key* is to be specified. Instead of passing them an operation-options handle, you pass just the replacement document and the key of the document to replace. This means that you cannot specify a filter, document version, and so on.

- `OCISodaReplOneAndGetWithKey()`
- `OCISodaReplOneWithKey()`

The functions with `AndGet` in their name return the new (result) document as the value of the same parameter that was used for the input document, so you can get its components.

Whichever replacement function you use, it returns a Boolean value as output parameter `isReplaced`, indicating whether the replacement operation was successful.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOne()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOneAndGet()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOneWithCtnt()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOneAndGetWithCtnt()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOneWithKey()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaReplOneAndGetWithKey()`

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

This example creates a new document as a replacement for the document with UUID key `"3C03C00FA3904FC2BF5182C424A2C6C1"`. It uses OCI function `OCISodaReplOneAndGet()` to replace the document having that key, and it gets the result document.

It uses function `OCIAttrGet()` to retrieve various components from the result document, which it prints. The use of mode parameter `OCI_SODA_ATOMIC_COMMIT` ensures that the replacement and any other outstanding operations are committed.

```
OCISodaDoc              *dochp = NULL;
OCISodaDoc              *tempDochp = NULL;

// Document content: JSON data
char                     documentContent[30] = "{\"name\":\"LiLing\"}";
ub4                      docFlags = OCI_DEFAULT;

OraText                 *key = NULL;
```

```
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
OraText                 *version = NULL;
ub4                      versionLen = 0;
OraText                 *lastModified = NULL;
ub4                      lastModifiedLen = 0;
OraText                 *mediaType = NULL;
ub4                      mediaTypeLen = 0;
OraText                 *createdOn = NULL;
ub4                      createdOnLen = 0;
boolean                  isReplaced = FALSE;

OCISodaOperationOptions *opthp;
OraText                 *inKey = "3C03C00FA3904FC2BF5182C424A2C6C1";

// Create a temporary replacement document, which has documentContent as its
content.
rc = OCISodaDocCreate(envhp,
                      documentContent,
                      (ub4) strlen(documentContent),
                      docFlags,
                      &dochp,
                      errhp,
                      OCI_DEFAULT);
if (rc != OCI_SUCCESS)
{
  printf("OCISodaDocCreate failed\n");
  goto finally;
}

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the document-key attribute on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// OCISodaReplOneAndGet returns the result document as dochp, so
// before calling it we save a pointer, tempDochp, to the handle that
// was returned by OCISodaDocCreate.  Later we free tempDochp.
tempDochp = dochp;

// Replace the document that has the key set in the operation-options
// handle with the new, replacement document pointed to by dochp, and
// get back the result document.
//
```

```
// The result document has the content of the replacement
// document, but it also has all of the other document components,
// automatically populated by SODA when the replacement document was
inserted.
rc = OCISodaReplOneAndGet(svchp,
                          collhp,
                          opthp,
                          &dochp,
                          &isReplaced,
                          errhp,
                          OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (isReplaced) printf ("Document was replaced.\n");

// Get and print the components of the document after replacement.
rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &key,
                &keyLen,
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Key: %.*s\n", keyLen, key);

rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &version,
                &versionLen,
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Version: %.*s\n", versionLen, version);

rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &lastModified,
                &lastModifiedLen,
                OCI_ATTR_SODA_LASTMOD_TIMESTAMP,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Last-modified: %.*s\n", lastModifiedLen, lastModified);

rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &createdOn,
                &createdOnLen,
                OCI_ATTR_SODA_CREATE_TIMESTAMP,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Created: %.*s\n", createdOnLen, createdOn);

rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &mediaType,
```

```
                &mediaTypeLen,
                OCI_ATTR_SODA_MEDIA_TYPE,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Media Type: %.*s\n", mediaTypeLen, mediaType);

finally:

// Release the session and free all handles, including the handle of the
temporary document.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (dochp)
  (void) OCIHandleFree((dvoid *) dochp, OCI_HTYPE_SODA_DOCUMENT);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
if (tempDochp)
  (void) OCIHandleFree((dvoid *) tempDochp, OCI_HTYPE_SODA_DOCUMENT);
```

Example 3-20 Replacing a Particular Version of a Document

To implement optimistic locking when replacing a document, you can specify both key and version, as in this example.

```
OCISodaDoc              *dochp = NULL;
OCISodaDoc              *tempDochp = NULL;

// Document content: JSON data
char                     documentContent[30] = "{\"name\":\"Esmeralda\"}";
ub4                      docFlags = OCI_DEFAULT;

OraText                 *key = NULL;
ub4                      keyLen = 0;
OraText                 *content = NULL;
ub4                      contentLen = 0;
OraText                 *version = NULL;
ub4                      versionLen = 0;
boolean                  isReplaced = FALSE;

OCISodaOperationOptions *opthp;

OraText                 *inKey = "3C03C00FA3904FC2BF5182C424A2C6C1";
OraText                 *inVersion =
  "BD0A8E86428FFD68A00FAE7833B41404637EE0A31791B36EC4C78A5782272448";

// Create a temporary replacement document, which has documentContent as its
content.
rc = OCISodaDocCreate(envhp,
                      documentContent,
                      (ub4) strlen(documentContent),
                      docFlags,
                      &dochp,
                      errhp,
                      OCI_DEFAULT);
if (rc != OCI_SUCCESS)
```

```
{
  printf("OCISodaDocCreate failed\n");
  goto finally;
}

// Allocate an empty operation options handle
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the key of the document we want to replace on the operation
options handle
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Set the version of the document we want to replace on the operation
options handle
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inVersion,
                strlen(inVersion),
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// OCISodaReplOneAndGet returns the result document as dochp, so
// before calling it we save a pointer, tempDochp, to the handle that
// was returned by OCISodaDocCreate.  Later we free tempDochp.
tempDochp = dochp;

// Replace the document that has the key and version set in the
// operation-options handle with the new, replacement document pointed
// to by dochp, and get back the result document.
//
// The result document has the content of the replacement
// document, but it also has all of the other document components,
// automatically populated by SODA when the replacement document was
inserted.
rc = OCISodaReplOneAndGet(svchp,
                          collhp,
                          opthp,
                          &dochp,
                          &isReplaced,
                          errhp,
                          OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (isReplaced) printf ("Document was replaced.\n");
```

```
// Get and print the components of found document after replacement.
rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &key,
                &keyLen,
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Key: %.*s\n", keyLen, key);

rc = OCIAttrGet((dvoid *) dochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &version,
                &versionLen,
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;
printf("Version: %.*s\n", versionLen, version);

finally:

// Release the session and free all handles, including handle of the
temporary document.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (dochp)
  (void) OCIHandleFree((dvoid *) dochp, OCI_HTYPE_SODA_DOCUMENT);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
if (tempDochp)
  (void) OCIHandleFree((dvoid *) tempDochp, OCI_HTYPE_SODA_DOCUMENT);
```

Related Topics

- SODA for C Read and Write Operations
  For all read operations, and for write operations other than insertions, you: (1) allocate an operation-options handle, (2) set some of its attributes to specify a particular operation, and (3) pass the handle to a generic function that performs the operation.

Related Topics

- Handling Transactions with SODA for C
  You can handle individual read and write operations, or groups of them, as a database transaction.
- Dropping a Document Collection with SODA for C
  To drop a document collection, use OCI function `OCISodaCollDrop()`.
- Inserting Documents into Collections with SODA for C
  Various ways to insert a document into a SODA collection are described.

3.12 Removing Documents from a Collection with SODA for C

To remove a document from a collection you can use function `OCISodaRemove()`, passing it an operation-options handle. If you only want to remove one document, specified by its key, then

you can alternatively use convenience function `OCISodaRemoveOneWithKey()`. It does not require an operation-options handle — you pass it the key directly.

Whichever document-removal function you use, the function returns the number of documents removed as an out parameter.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaRemove()`
> - *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaRemoveOneWithKey()`

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

This example removes the document with UUID key `"E914016C41174F6CBF7C877C7F9EB4C2"`. The use of mode parameter `OCI_SODA_ATOMIC_COMMIT` ensures that the removal and any other outstanding operations are committed.

```
OraText                 *inKey = "E914016C41174F6CBF7C877C7F9EB4C2";
ub8                      removeCount = 0;

OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the document-key attribute on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Remove the document that has the key set in the operation-options
handle.
rc = =OCISodaRemove(svchp,
                    collhp,
                    opthp,
                    &removeCount,
                    errhp,
                    OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (removeCount > 0)
  printf("Successfully removed document.\n");
```

```
else
  printf("Document with specified key was not found.\n");

finally:

// Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

Example 3-22 Removing a Particular Version of a Document

This example uses function `OCISodaRemove()` with an operation-options handle that specifies the version, as well as the key, of the document to remove. This is useful for implementing optimistic locking, for write operations.

When specifying the document version you typically specify the key as well. But you can specify the version along with a filter, provided the filter specifies at most one document in the collection.

```
ub8                      removeCount = 0;

OraText                 *inKey = "0C6132FC780D4F16BF9561FC9E2B4F98";
OraText                 *inVersion =
  "7CCEF2F54035DE9A9D64653645DBEF7E61B92142F2E41B3F6144262A5F7BC054";

OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle,
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the document-key attribute on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inKey,
                strlen(inKey),
                OCI_ATTR_SODA_KEY,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Set the document-version attribute on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                inVersion,
                strlen(inVersion),
                OCI_ATTR_SODA_VERSION,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Remove document that has the key and version set in the operation-options
handle.
```

```
rc = OCISodaRemove(svchp,
                   collhp,
                   opthp,
                   &removeCount,
                   errhp,
                   OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (removeCount > 0)
  printf("Successfully removed document.\n");
else
  printf("Document with specified key was not found.\n");

finally:

// Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

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

This example uses function `OCISodaOperKeysSet()` to set operation-options handle attributes for key and key length for two documents. It then invokes function `OCISodaRemove()` to remove the documents that have those keys. Function `OCISodaOperKeysSet()` accepts an array of keys, an array of the corresponding key lengths, and the number of keys as arguments. (You cannot set multiple keys and their lengths using standard function **`OCIAttrSet()`**.)

> **Note:**
>
> If you use function `OCIAttrSet()` to set attribute `OCI_ATTR_SODA_KEY` on an operation-options handle, and you also use function `OCISodaOperKeysSet()` to set multiple keys on the same handle, then only the latest of the two settings takes effect. The effect of the first function invoked is overridden by the effect of the second.

```
OraText                *keys[2] = {"ACF8C4BDA3E44F4CBF802C9708D00C10",
                                   "787B22133B254F0CBF2DB9975E277913"};

ub4                     keyLengths[2];
ub8                     removeCount = 0;
int                     i = 0;

OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;
```

```
// Fill array of key lengths.
for(i=0; i<2; i++)
  keyLengths[i] = strlen(keys[i]);

// Set keys and key lengths on operation-options handle.
rc = OCISodaOperKeysSet(opthp,
                        keys,
                        keyLengths,
                        2,
                        errhp,
                        OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

// Remove documents matching the keys in the operation options handle.
rc = OCISodaRemove(svchp,
                   collhp,
                   opthp,
                   &removeCount,
                   errhp,
                   OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (removeCount > 0)
  printf("Successfully removed %d documents.\n", removeCount);
else
  printf("Document with specified keys were not found.\n");

finally:

// Free all handles.
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
```

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.

```
ub8                      removeCount = 0;
OraText                 *filter = "{\"greeting\" : \"hello\"}";
OCISodaOperationOptions *opthp;

// Allocate an empty operation-options handle.
rc = OCIHandleAlloc((void *) envhp, (void **)&opthp,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0);
if (rc != OCI_SUCCESS) goto finally;

// Set the filter (query-by-example, or QBE) on the operation-options handle.
rc = OCIAttrSet(opthp,
                OCI_HTYPE_SODA_OPER_OPTIONS,
                filter,
```

```
                strlen(filter),
                OCI_ATTR_SODA_FILTER,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Remove documents matching the filter (QBE) set in operation-options
handle.
rc = OCISodaRemove(svchp,
                   collhp,
                   opthp,
                   &removeCount,
                   errhp,
                   OCI_SODA_ATOMIC_COMMIT);
if (rc != OCI_SUCCESS) goto finally;

if (removeCount > 0)
  printf("Successfully removed %d documents.\n", removeCount);
else
  printf("No documents matching the filter were found.\n");

finally:

//Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (opthp)
  (void) OCIHandleFree((dvoid *) opthp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

Related Topics

- SODA for C Read and Write Operations
  For all read operations, and for write operations other than insertions, you: (1) allocate an operation-options handle, (2) set some of its attributes to specify a particular operation, and (3) pass the handle to a generic function that performs the operation.

3.13 Indexing the Documents in a Collection with SODA for C

Indexing can improve the performance of QBEs. To index the documents in a SODA collection, use function `OCISodaIndexCreate()`, passing it 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.

- A B-tree index is used to index particular scalar JSON fields.
- An Oracle Spatial and Graph index is used to index GeoJSON (spatial) data.
- A JSON search index can improve the performance of:
  - QBEs that you might not anticipate or use regularly — it is a general purpose index.
  - QBEs that use operator `$contains` — full-text search.

- A JSON search index can also provide persistent recording and automatic updating of JSON data-guide information.

If a JSON search index is defined, and if a B-tree index or a spatial index applies to a given QBE, the B-tree or spatial index is generally used for that QBE, in preference to the (more general) search index.

The invocation of function `OCISodaIndexCreate()` is the same for each kind of index you create. The only difference is the index specification that is passed to the function as an argument.

You drop an index on a SODA collection using function `OCISodaIndexDrop()`, passing it the index name.

See Also:

- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaIndexCreate()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCISodaIndexDrop()`
- *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 C

This example creates a B-tree non-unique index for numeric field `address.zip` of the JSON documents in a collection that has handle `collhp`. A B-tree index specification can be recognized by the presence of field `fields`.

```
// Index specification for B-tree index on field address.zip.
OraText *indexSpec = "{\"name\" : \"ZIPCODE_IDX\", \
                      \"fields\" : [{\"path\" : \"address.zip\", \
                                     \"datatype\" : \"number\", \
                                     \"order\" : \"asc\"}]}";

// Create the index.
rc = OCISodaIndexCreate(svchp, collhp, indexSpec, strlen(indexSpec), errhp,
OCI_DEFAULT);
```

Example 3-26 Creating a JSON Search Index with SODA for C

This example indexes the documents in a collection that has handle `collhp` for ad hoc queries and full-text search (QBEs that use 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`).

```
// Index specification for JSON search index.
OraText *indexSpec = "{\"name\" : \"SEARCH_AND_DATA_GUIDE_IDX\"}";

// Create the index.
rc = OCISodaIndexCreate(svchp, collhp, indexSpec, strlen(indexSpec),
errhp, OCI_DEFAULT);
```

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* 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 C

To drop an index on a SODA collection, just pass the index name to function `OCISodaIndexDrop()`. This example drops index `ZIPCODE_IDX`.

```
boolean isDropped = FALSE;

// Drop the index named ZIPCODE_IDX.
rc = OCISodaIndexDrop(svchp,
                      "ZIPCODE_IDX",
                      strlen("ZIPCODE_IDX"),
                      &isDropped,
                      errhp,
                      OCI_DEFAULT);

printf ("isDropped %d\n", isDropped);
```

3.14 Getting a Data Guide for a Collection with SODA for C

You use function `OCISodaDataGuideGet()` or `OCISodaDataGuideGetWithOpts()` 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 C support for JSON data guide was added in Oracle Database 18.3. You need that database release or later to use this SODA feature.

There are two alternative ways to create a data guide for a collection:

- Use function `OCISodaDataGuideGetWithOpts()` together with a query-by-example (QBE) filtering operation. This creates a data guide dynamically from scratch, for only the documents selected by your query. You can thus *limit the set of documents* on which the data guide is based. Example 3-28 illustrates this.

  (This method corresponds to using SQL function `json_dataguide`.)

- Use function `OCISodaDataGuideGet()`. This always creates a data guide based on *all* documents in the collection. Example 3-29 illustrates this.

  This method makes use of *persistent data-guide information* that is stored as part of a JSON search index, so before you can use this method you must first create a data guide-enabled JSON search index on the collection. Example 3-26 shows how to do that. The data-guide information in the index is persistent, and is updated automatically as new JSON content is added.

  (This method corresponds to using PL/SQL function `get_index_dataguide`.)

The index-based function, `OCISodaDataGuideGet()`, incurs an ongoing cost of updating relevant data persistently: document writes (creation and updating) entail index updates. But because data-guide information is readily available in the index, it need not be gathered from scratch when generating the data-guide document.

Because function `OCISodaDataGuideGetWithOpts()` starts from scratch each time, a typical use of it involves applying the method to only the documents that satisfy some filter (QBE), as shown in Example 3-28.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for information about OCI function `CISodaDataGuideGet()`

Example 3-28 Creating a Data Guide Dynamically with SODA for C

This example uses function `OCISodaDataGuideGetWithOpts()` to obtain a data guide for a filtered set of documents in collection `collhp`. Using a query-by-example (QBE) filtering

operation is a common way to limit the documents represented by a dynamically created data guide.

The example pretty-prints the content of the data-guide document in the flat format. Finally, it frees the temporary LOB used for the data-guide document.

```
OCISodaDoc                *dgdochp  = NULL;
OCISodaOperationOptions   *opthp  = NULL;
oratext                   *content;
ub4                        contentLen;
oratext                   *qbe;

rc = OCIHandleAlloc((void *) envhp, (void **)&opt,
                    OCI_HTYPE_SODA_OPER_OPTIONS, (size_t) 0,
                    (dvoid **) 0
if (rc != OCI_SUCCESS) goto finally;

qbe = (oratext *)"{\"name\" : \"alexander\"}";
qlen = (ub4) strlen(qbe);

rc = OCIAttrSet(opthp, OCI_HTYPE_SODA_OPER_OPTIONS, qbe, qlen,
                OCI_ATTR_SODA_FILTER, errhp);
if (rc != OCI_SUCCESS) goto finally;

rc = OCISodaDataGuideGetWithOpts(svchp, collhp, opthp,
                                 OCI_SODA_DG_FMT_HIERARCHICAL,
                                 OCI_SODA_DATAGUIDE_PRETTY,
                                 OCI_SODA_AS_AL32UTF8,
                                 &dgdochp, errhp, OCI_DEFAULT));
if (rc != OCI_SUCCESS) goto finally;

rc = OCIAttrGet((dvoid *)dgdochp, OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *)&content, &contentLen,
                OCI_ATTR_SODA_CONTENT, errhp)
if (rc != OCI_SUCCESS) goto finally;

printf("Dataguide: %.*s\n", contentLen, content);

finally:
  // Free all handles.
    if (dgdochp)
      (void) OCIHandleFree((dvoid *) dgdochp, OCI_HTYPE_SODA_DOCUMENT);
    if (opthp)
      (void) OCIHandleFree((dvoid *) opthp,
OCI_HTYPE_SODA_OPER_OPTIONS);
```

> **See Also:**
>
> OCISodaDataGuideGetWithOpts() in *Oracle Call Interface Programmer's Guide*

Example 3-29 Creating a Data Guide Using a JSON Search Index with SODA for C

This example gets a data guide for a collection with collection handle `collhp`, using function `OCISodaDataGuideGet()`. It then prints the content of the data-guide document.

```
OCISodaDoc  *dgdochp = NULL;
OraText     *content = NULL;
ub4          contentLen = 0;

// Get the data guide based on the JSON search index defined on the
// collection.  dgdochp is the handle for the data-guide document.
rc = OCISodaDataGuideGet(svchp,
                         collhp,
                         OCI_DEFAULT,
                         &dgdochp,
                         errhp,
                         OCI_DEFAULT);
if (rc != OCI_SUCCESS) goto finally;

rc = OCIAttrGet((dvoid *) dgdochp,
                OCI_HTYPE_SODA_DOCUMENT,
                (dvoid *) &content,
                &contentLen,
                OCI_ATTR_SODA_CONTENT,
                errhp);
if (rc != OCI_SUCCESS) goto finally;

// Print the content of the data-guide document.
printf("Data guide: %.*s \n", contentLen, content);

finally:

// Free all handles.
if (collhp)
  (void) OCIHandleFree((dvoid *) collhp, OCI_HTYPE_SODA_COLLECTION);
if (dgdochp)
  (void) OCIHandleFree((dvoid *) dgdochp, OCI_HTYPE_SODA_OPER_OPTIONS);
```

3.15 Handling Transactions with SODA for C

You can handle individual read and write operations, or groups of them, as a database transaction.

You do this in *either* of these ways:

- Use execution mode parameter **`OCI_SODA_ATOMIC_COMMIT`** when you invoke a SODA operation. If an operation is executed in this mode and it completes successfully then the current transaction is committed after completion.

  As is usual for a commit, this commits *all* outstanding changes, not just changes made by the SODA operation. However, if the operation *fails* then *only changes made by the SODA operation are rolled back*; any uncommitted changes made prior to invocation of the SODA operation are not rolled back.

- Use function `OCITransCommit()` or `OCITransRollback()`, to commit or roll back, respectively, the current transaction. These are standard Oracle Call Interface (OCI) functions; they are not SODA-specific.

SODA operations of creating and dropping a 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 committed or rolled back. This is because function `OCISodaCollDrop()` does not itself commit before it performs its action. In this, its behavior differs from that of a SQL `DROP TABLE` statement.

Related Topics

- Dropping a Document Collection with SODA for C
  To drop a document collection, use OCI function `OCISodaCollDrop()`.
- Inserting Documents into Collections with SODA for C
  Various ways to insert a document into a SODA collection are described.
- Replacing Documents in a Collection with SODA for C
  You can use function `OCISodaReplOneAndGet()` to replace a document in a collection, passing it an operation-options handle that specifies the key of the document to replace as well as the new, replacement document. It returns that replacement document, but with all of its metadata filled in, as the result document.

See Also:

-
- *Oracle Call Interface Programmer's Guide* for information about execution mode parameter `OCI_SODA_ATOMIC_COMMIT`
- *Oracle Call Interface Programmer's Guide* for information about Oracle Call Interface (OCI) support for transactions
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCITransCommit()`
- *Oracle Call Interface Programmer's Guide* for information about OCI function `OCITransRollback()`

4
Character-Set Considerations for SODA for C

Use of character sets with SODA for C is discussed. This applies only to the encoding of JSON documents. (Non-JSON documents are always stored in a SODA collection using `BLOB` content, which is treated only as a sequence of bytes, not characters.)

SODA for C and Character-Set Encodings for JSON Data: Client and Database

SODA for C involves two kinds of JSON-data character-set encodings: client-side and database.

By the standard defining JSON, JSON data is encoded with a *Unicode* character set; that is, *JSON data is Unicode data, by definition*. But on the client side SODA for C relaxes the restriction that JSON data must be Unicode; you can use data that has other encodings but otherwise has JSON syntax.

On the *client* side:

- The non-Unicode encodings that you can use with a SODA for C client are all of those allowed by Oracle Call Interface (OCI), with the exception of EBCDIC: you *cannot* use an EBCDIC character set for SODA documents.
- The Unicode encodings that you can use with a SODA for C client are UTF-8, UTF-16 LE (little-endian), and UTF-16 BE (big-endian). These correspond to Oracle Database character sets AL32UTF8, AL32UTF16, and AL32UTF16LE, respectively. You *cannot* use UTF-32 — it is not an OCI client-side encoding.

On the *database* side (that is, for the content column of a collection):

- Oracle recommends that you use AL32UTF8, which implements Unicode UTF-8, as the *database character set*.
- The encoding used for JSON data in the content column of a collection depends on the SQL type:
  - `VARCHAR2` — The documents are encoded as AL32UTF8. `VARCHAR2` data is always stored in the database character set.
  - `BLOB` — The documents are encoded as UTF-8, UTF-16 BE, or UTF-16 LE. Which of these Unicode encodings is used depends on how the input documents were encoded on the client side, as is explained in Writing JSON Documents To the Database From the Client.
  - `CLOB` — The documents are encoded as UCS-2. A `CLOB` instance is encoded as UCS-2 whenever the database character set is multibyte (as is AL32UTF8).

If client-side and database-side encodings are the same (they are both Unicode) then no conversion is needed from one to the other.

But if they differ then SODA automatically converts from one character set to the other. If a character used in a document on the client side has no corresponding Unicode character then conversion to the database character set when writing the document is *lossy*. Similarly, if a character used in a document on the database side has no corresponding character in the client-side character set then conversion when reading the document is *lossy*.

For example:

- Suppose that your client-side encoding is JA16SJIS, and the content column for your SODA collection is configured to store JSON data using SQL data type `VARCHAR2`. When you write data to your collection SODA automatically converts it from JA16SJIS to the database character set (AL32UTF8).
- Suppose that your client-side encoding is AL16UTF16LE, and your collection is configured to store JSON data using SQL data type `BLOB`. Because data type `BLOB` supports encoding AL16UTF16LE, no conversion is needed.

By default, the character set used by OCI is defined by environment variable `NLS_LANG`. You can override this for a given OCI client using OCI function `OCIEnvNlsCreate()` with parameter `charset`.

In particular, you can use `OCIEnvNlsCreate()` to create an environment handle that defines the character set used by a given client as `OCI_UTF16ID` (UTF-16), which *cannot* be set from `NLS_LANG`. Character set `OCI_UTF16ID` designates a UTF-16 encoding whose endianness (big-endian or little-endian) depends on the platform where the client is run.

When a document is written to the database from a client application, or a document is read from the database to a client application, the application tells OCI what client-side encoding to use for the document. It does this by way of parameter **`docFlags`**, which is passed to either a document-handle creation function or a convenience function for writing content into a document without providing a document handle. That is, parameter `docFlags` controls the encoding of documents on the client side.

Writing JSON Documents To the Database From the Client

SODA for C functions that create a document handle are named with prefix `OCISodaDocCreate`. They all accept parameter `docFlags`.

SODA for C also provides convenience functions for writing JSON content to the database without providing a document handle. These functions are named with suffix `WithCtnt` (standing for “with content”). They also accept parameter `docflags`.

For writing, parameter `docFlags` can have either of these values:

- **`OCI_DEFAULT`** — Use the character set defined by the environment handle, or by environment variable `NLS_LANG`, if not set for the handle.

  You *must* supply document content in the encoding that is specified by the environment handle or `NLS_LANG`. Otherwise, the result of a write operation is unpredictable.

  The character set can be any that is valid for OCI (Unicode or non-Unicode), with the exception of EBCDIC. (If it is OCI_UTF16 then you must supply the document with a UTF 16 encoding whose endianness matches the endianness of the platform where the client runs.)

  If you write a document that is not encoded as Unicode to a `BLOB` column using `OCI_DEFAULT` then SODA converts the content to UTF-8 before writing.

- **`OCI_SODA_DETECT_JSON_ENC`** — Automatically detect the encoding of the document content as UTF-8, UTF-16 LE (little-endian), or UTF-16 BE (big-endian)

  You *must* supply document content in one of those encodings. Otherwise, the result of a write operation is unpredictable.

Use cases for working with JSON data on the client side:

- To work in a *non-Unicode* encoding or in a *single Unicode* encoding, use `OCI_DEFAULT`.
- To work in a mix of Unicode encodings (UTF-8, UTF-16 LE, UTF-16 BE) in the same application, use `OCI_SODA_DETECT_JSON_ENC`. (With `OCI_DEFAULT`, documents are assumed to be in the single encoding specified by the environment handle or `NLS_LANG`.)
- To work in a UTF-16 encoding that has a different endianness from that of the client-side platform, use `OCI_SODA_DETECT_JSON_ENC`.

If the client-side character set differs from the character set of the content column in the database, SODA converts the document, when writing, to the character set of the content column. To avoid any such conversion, use `BLOB` as the content data type (`BLOB` is the default), and supply the content with encoding UTF-8 or UTF-16 (BE or LE). If you do this then it does not matter which value (`OCI_DEFAULT` or `OCI_SODA_DETECT_JSON_ENC`) you use for parameter `docFlags`.

Reading JSON Documents From the Database To the Client

SODA for C functions (such as `OCISodaFindOneWithKey()`) that read content into a client-side document also provide parameter `docFlags`, which you use to specify the client-side encoding to use for the retrieved content.

For reading, parameter `docFlags` can have any of these values:

- **`OCI_DEFAULT`** — Use the character set defined by the environment handle, or by environment variable `NLS_LANG`, if not set for the handle. (This is the same as for document writes to the database.)
- **`OCI_SODA_AS_STORED`** — Use the same encoding used to store the document in the database. This value is valid only for use with a collection that uses `BLOB` storage; otherwise, an error is raised.
- **`OCI_SODA_AS_AL32UTF8`** — Use UTF-8 as the encoding.

If the client-side character set differs from the character set of the content column in the database, SODA converts the document, when reading, to the character set specified for the client. To avoid any such conversion, use `BLOB` as the content data type (`BLOB` is the default), and use `OCI_SODA_AS_STORED` for parameter `docFlags`.

See Also:

- *Oracle Call Interface Programmer's Guide* for information about setting the OCI client character set
- *Oracle Call Interface Programmer's Guide* for information about OCI support for globalization
- *Oracle Database Globalization Support Guide* for complete information about Oracle Database support for globalization
- *Oracle Database JSON Developer’s Guide*
- Unicode.org for information about Unicode
- IETF RFC4627 and ECMA 404 for the JSON Data Interchange Format

5

Multithreading in SODA for C Applications

SODA for C is designed for lockless multithreading in applications.

To achieve multithreading, just use separate handles in each thread of your SODA application. SODA handles are not designed to be shared between threads. In particular, they are not locked with mutexes to negotiate mutual exclusion among threads.

For example, to read or write to the same collection from multiple threads, obtain a separate collection handle in each thread using `OCISodaCollOpen()`, and use each handle to perform read and write operations.

Only in the case of *document* handles can it sometimes make sense to share SODA handles among threads.

For example, one thread might create documents and put them into a queue, while worker threads dequeue the head document and insert it into a collection. Document handles could be shared among threads, here.

You don't want multiple threads working on the same document at the same time, but a single document handle can be passed from one thread to another. It is your responsibility to provide application-level synchronization so that the document handle is not simultaneously accessed from different threads.

Related Topics

- Opening an Existing Document Collection with SODA for C
  Use OCI function `OCISodaCollOpen()` to open an existing document collection.
- Creating Documents with SODA for C
  Various ways to create a SODA document are described, along with the components of a document.

6

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 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 can use OCI function `OCIAttrGet()` with attribute `OCI_ATTR_SODA_DESCRIPTOR`, to get *all* of the metadata of a collection at once, as a JSON document. You can also use `OCIAttrGet()` to get individual collection metadata attributes.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to OCI function `OCISodaCollCreateWithMetadata()`.

> **See Also:**
>
> - Overview of SODA Document Collections in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for general information about SODA document collections and their metadata
> - SODA Collection Metadata Components (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for reference information about collection metadata components

6.1 Getting the Metadata of an Existing Collection

You can use OCI function `OCIAttrGet()` with attribute `OCI_ATTR_SODA_DESCRIPTOR`, to get *all* of the metadata of a collection at once, as a JSON document. You can also use `OCIAttrGet()` to get individual collection metadata attributes.

Table 6-1 Collection Handle Attributes (Collection Metadata)

Attribute	Description
`OCI_ATTR_SODA_CRTIME_COL_NAME`	The name of the database column that stores the creation time stamp of the document.
`OCI_ATTR_SODA_CTNT_CACHE`	The SecureFiles LOB cache setting.
`OCI_ATTR_SODA_CTNT_COL_NAME`	The database column that stores the document content.
`OCI_ATTR_SODA_CTNT_COMPRESS`	The SecureFiles LOB compression setting.
`OCI_ATTR_SODA_CTNT_ENCRYPT`	The SecureFiles LOB encryption setting.
`OCI_ATTR_SODA_CTNT_MAX_LEN`	The maximum length, in bytes, of the database column that stores the document content. This attribute applies only to content of type `VARCHAR2`.
`OCI_ATTR_SODA_CTNT_SQL_TYPE`	The SQL data type of the database column that stores the document content.
`OCI_ATTR_SODA_CTNT_VALIDATION`	The syntax to which JavaScript Object Notation (JSON) content must conform — standard, strict, or lax.
`OCI_ATTR_SODA_DESCRIPTOR`	All of the metadata of the collection, in JSON format.
`OCI_ATTR_SODA_KEY_ASSIGN_METHOD`	The method used to assign keys to documents that are inserted into the collection.
`OCI_ATTR_SODA_KEY_COL_NAME`	The name of the database column that stores the document key.
`OCI_ATTR_SODA_KEY_MAX_LEN`	The maximum length, in bytes, of the database column that stores the document key. This attribute applies only to content of type `VARCHAR2`.
`OCI_ATTR_SODA_KEY_SEQ_NAME`	The name of the database sequence that generates keys for documents that are inserted into a collection if the key assignment method is `SEQUENCE`.
`OCI_ATTR_SODA_KEY_SQL_TYPE`	The SQL data type of the database column that stores the document key.
`OCI_ATTR_SODA_MODTIME_COL_NAME`	The name of the database column that stores the last-modified time stamp of the document.
`OCI_ATTR_SODA_MODTIME_INDEX`	The name of the index on the database column that stores the last-modified time stamp.
`OCI_ATTR_SODA_READONLY`	An indication of whether the collection is read-only.

Table 6-1 (Cont.) Collection Handle Attributes (Collection Metadata)

Attribute	Description
OCI_ATTR_SODA_SCHEMA	The name of the Oracle Database schema (user) that owns the table or view to which the collection is mapped.
OCI_ATTR_SODA_TABLE_NAME	The name of the database table to which the collection is mapped.
OCI_ATTR_SODA_VERSION_COL_NAME	The name of the database column that stores the document version.
OCI_ATTR_SODA_VERSION_METHOD	The method used to compute version values for documents when they are inserted into a collection or replaced.
OCI_ATTR_SODA_VIEW_NAME	The name of the database view to which the collection is mapped.

See Also:

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Introduction to Simple Oracle Document Access (SODA)*

Example 6-1 Getting All of the Metadata of a Collection

This example shows the result of invoking function `OCIAttrGet()` for collection-handle attribute `OCI_ATTR_SODA_DESCRIPTOR` on the collection with the default configuration that was created using Example 3-2. This retrieves all of the collection metadata as JSON data.

```
OraText *fetchedMetadata;
ub4      fetchedMetadataLen = 0;

rc = OCIAttrGet((dvoid *)collhp,
                OCI_HTYPE_SODA_COLLECTION,
                (dvoid *)fetchedMetadata,
                &fetchedMetadataLen,
                OCI_ATTR_SODA_DESCRIPTOR, errhp);

if (rc == OCI_SUCCESS)
  printf ("Collection specification: %.*s\n",
          fetchedMetadataLen,
          fetchedMetadata);
```

The default metadata for a collection is presented in Default Collection Metadata in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*.

Example 6-2 Getting Individual Collection Metadata Attributes

This example uses `OCIAttrGet()` to get individual collection metadata attributes. For each attribute, you pass the collection handle, the attribute, and the attribute type.

```
// String collection metadata attribute.
oratext *collAttr = NULL;

// Length of collection metadata attribute (relevant only for
// string attributes).
ub4      collAttrLen = 0;

ub1      ub1CollAttr = 0;
ub4      ub4CollAttr = 0;
boolean  boolCollAttr = FALSE;

// Get and print collection metadata components.  (For brevity we
// omit checking the return values of the OCIAttrGet calls here.)

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)collAttr,
           &collAttrLen,
           OCI_ATTR_SODA_COLL_NAME,
           errhp);
printf("Collection name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)collAttr,
           &collAttrLen,
           OCI_ATTR_SODA_TABLE_NAME,
           errhp);
printf("Table name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)collAttr,
           &collAttrLen,
           OCI_ATTR_SODA_SCHEMA,
           errhp);
printf("Schema name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)collAttr,
           &collAttrLen,
           OCI_ATTR_SODA_KEY_COL_NAME,
           errhp);
printf("Key column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)(&ub1CollAttr),
           &collAttrLen,
```

```
            OCI_ATTR_SODA_KEY_SQL_TYPE,
            errhp);
if (ub1CollAttr == SQLT_CHR)
  printf ("Key column type: VARCHAR2\n");
else if (ub1CollAttr == SQLT_BIN)
  printf ("Key column type: RAW\n");
else if (ub1CollAttr == SQLT_NUM)
  printf ("Key column type: NUMBER\n");

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)(&ub4CollAttr),
            &collAttrLen,
            OCI_ATTR_SODA_KEY_MAX_LEN,
            errhp);
printf ("Key column max length: %d\n", ub4CollAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)(&ub1CollAttr),
            &collAttrLen,
            OCI_ATTR_SODA_KEY_ASSIGN_METHOD,
            errhp);

if (ub1CollAttr == OCI_SODA_KEY_METHOD_UUID)
  printf ("Key assignment method: UUID\n");
else if (ub1CollAttr == OCI_SODA_KEY_METHOD_GUID)
  printf ("Key assignment method: GUID\n");
else if (ub1CollAttr == OCI_SODA_KEY_METHOD_SEQUENCE)
  printf ("Key assignment method: SEQUENCE\n");
else if (ub1CollAttr == OCI_SODA_KEY_METHOD_CLIENT)
  printf ("Key assignment method: CLIENT\n");

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
            OCI_ATTR_SODA_CTNT_COL_NAME,
            errhp);
printf("Content column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)(&ub1CollAttr),
            &collAttrLen,
            OCI_ATTR_SODA_CTNT_SQL_TYPE,
            errhp);
if (ub1CollAttr == SQLT_JSON)
  printf ("Content column type: JSON\n");
else if (ub1CollAttr == SQLT_CHR)
  printf ("Content column type: VARCHAR2\n");
else if (ub1CollAttr == SQLT_BLOB)
  printf ("Content column type: BLOB\n");
else if (ub1CollAttr == SQLT_CLOB)
  printf ("Content column type: CLOB\n");
```

```
OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)(&ub4CollAttr),
           &collAttrLen,
           OCI_ATTR_SODA_CTNT_MAX_LEN,
           errhp);
printf ("Content column max length: %d\n", ub4CollAttr);

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)(&ub1CollAttr),
           &collAttrLen,
           OCI_ATTR_SODA_CTNT_VALIDATION,
           errhp);
if (ub1CollAttr == OCI_SODA_JSON_VALIDATION_STRICT)
  printf ("Content column validation: STRICT\n");
else if (ub1CollAttr == OCI_SODA_JSON_VALIDATION_LAX)
  printf ("Content column validation: LAX\n");
else if (ub1CollAttr == OCI_SODA_JSON_VALIDATION_STD)
  printf ("Content column validation: STANDARD\n");

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)(&ub1CollAttr),
           &collAttrLen,
           OCI_ATTR_SODA_CTNT_COMPRESS,
           errhp);
if (ub1CollAttr == OCI_SODA_LOB_COMPRESS_NONE)
  printf ("Content column compress: NONE\n");
else if (ub1CollAttr == OCI_SODA_LOB_COMPRESS_HIGH)
  printf ("Content column compress: HIGH\n");
else if (ub1CollAttr == OCI_SODA_LOB_COMPRESS_MEDIUM)
  printf ("Content column compress:  MEDIUM\n");
else if (ub1CollAttr == OCI_SODA_LOB_COMPRESS_LOW)
  printf ("Content column compress: LOW\n");

OCIAttrGet((dvoid *)collhp,
           OCI_HTYPE_SODA_COLLECTION,
           (dvoid *)(&ub1CollAttr),
           &collAttrLen,
           OCI_ATTR_SODA_CTNT_ENCRYPT,
           errhp);
if (ub1CollAttr == OCI_SODA_LOB_ENCRYPT_NONE)
  printf ("Content column encrypt: NONE\n");
else if (ub1CollAttr == OCI_SODA_LOB_ENCRYPT_3DES168)
  printf ("Content column encrypt: 3DES168\n");
else if (ub1CollAttr == OCI_SODA_LOB_ENCRYPT_AES128)
  printf ("Content column encrypt: AES128\n");
else if (ub1CollAttr == OCI_SODA_LOB_ENCRYPT_AES192)
  printf ("Content column encrypt: AES192\n");
else if (ub1CollAttr == OCI_SODA_LOB_ENCRYPT_AES256)
  printf ("Content column encrypt: AES256\n");

OCIAttrGet((dvoid *)collhp,
```

```
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)(&boolCollAttr),
            &collAttrLen,
            OCI_ATTR_SODA_CTNT_CACHE,
            errhp);
if (boolCollAttr == TRUE)
  printf ("Content column cache: TRUE\n");
else
  printf ("Content column cache: FALSE\n");

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
            OCI_ATTR_SODA_VERSION_COL_NAME,
            errhp);
printf("Version column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)(&ub1CollAttr),
            &collAttrLen,
            OCI_ATTR_SODA_VERSION_METHOD,
            errhp);
if (ub1CollAttr == OCI_SODA_VERSION_NONE)
  printf ("Version method: NONE\n");
else if (ub1CollAttr == OCI_SODA_VERSION_TIMESTAMP)
  printf ("Version method: TIMESTAMP\n");
else if (ub1CollAttr == OCI_SODA_VERSION_MD5)
  printf ("Version method: MD5\n");
else if (ub1CollAttr == OCI_SODA_VERSION_SHA256)
  printf ("Version method: SHA256\n");
else if (ub1CollAttr == OCI_SODA_VERSION_SEQUENTIAL)
  printf ("Version method: SEQUENTIAL\n");

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
            OCI_ATTR_SODA_MODTIME_COL_NAME,
            errhp);
printf("Last-modified column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
            OCI_ATTR_SODA_MODTIME_INDEX,
            errhp);
printf("Last-modified index name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
```

```
            OCI_ATTR_SODA_CRTIME_COL_NAME,
            errhp);
printf("Created-on column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)collAttr,
            &collAttrLen,
            OCI_ATTR_SODA_MTYPE_COL_NAME,
            errhp);
printf("Media type column name: %.*s\n", collAttrLen, collAttr);

OCIAttrGet((dvoid *)collhp,
            OCI_HTYPE_SODA_COLLECTION,
            (dvoid *)&boolCollAttr,
            &collAttrLen,
            OCI_ATTR_SODA_READONLY,
            errhp);

if (boolCollAttr == TRUE)
  printf("Collection is read-only");
else
  printf("Collection is not read-only");
```

Related Topics

- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to OCI function `OCISodaCollCreateWithMetadata()`.

6.2 Creating a Collection That Has Custom Metadata

To create a document collection that has custom metadata, you pass its metadata, as JSON data, to OCI function `OCISodaCollCreateWithMetadata()`.

The optional metadata argument to OCI function `OCISodaCollCreateWithMetadata()` 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 metadata passed to `OCISodaCollCreateWithMetadata()` does not match that 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 Call Interface Programmer's Guide* for information about OCI function `OCISodaCollCreateWithMetadata()`
- *Oracle Call Interface Programmer's Guide* for information about collection-handle attribute `OCI_ATTR_SODA_DESCRIPTOR`

Example 6-3 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 collection-handle attribute `OCI_ATTR_SODA_DESCRIPTOR` to get the complete metadata from the newly created collection.

```
sword        rc = OCI_SUCCESS;
OCISodaColl *collhp = NULL;
OraText     *metadata ="{\"keyColumn\" : \
{\"name\" : \"KEY\", \"assignmentMethod\": \"CLIENT\" }, \
\"contentColumn\" : { \"name\" : \"JSON\", \"sqlType\": \"VARCHAR2\" } }";
OraText     *collName = "myCustomCollection";
OraText     *fetchedMetadata = NULL;
ub4          fetchedMetadataLen = 0;

rc = OCISodaCollCreateWithMetadata(svchp,
                                   collName,
                                   (ub4) strlen(collName),
                                   metadata,
                                   (ub4) strlen(metadata),
                                   &collhp,
                                   errhp,
                                   OCI_DEFAULT));
if (rc != OCI_SUCCESS)
{
  printf(OCISodaCollCreateWithMetadata failed\n");
  goto finally;
}

rc = OCIAttrGet((dvoid *)collhp,
                OCI_HTYPE_SODA_COLLECTION,
                (dvoid *)fetchedMetadata,
                &fetchedMetadataLen,
                OCI_ATTR_SODA_DESCRIPTOR,
                errhp);

if (rc == OCI_SUCCESS)
{
  printf ("Collection specification: %.*s\n", fetchedMetadataLen,
fetchedMetadata);
}

finally: ...
```

Here is the output, formatted for readability. 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 C
  Use OCI function `OCISodaCollCreate()` to create a collection, if you do not care about the details of its configuration. This creates a collection that has the default metadata. To create a collection that is configured in a nondefault way, use function `OCISodaCollCreateWithMetadata()` instead, passing it custom metadata, expressed in JSON.

Index

G

H

I

J

L

M

N

O

P

R

S

T

W