Oracle® Key Vault
Developer's Guide

Release 21.1
E61666-06
January 2021

Oracle Key Vault Developer's Guide, Release 21.1

E61666-06

Primary Author: Mark Doran

Contributors: Rahil Mir, Min-Hank Ho, Swapna Jawarikapisha , Shirley Kumamoto, Michael Leong, Sunil Pulla, Sindhu Ravichandran, Saikat Saha, Vipin Samar

Contents

11 Oracle Key Vault Client SDK KMIP Attributes and Custom Attributes APIs

Preface

Welcome to *Oracle Key Vault Developer's Guide*. This guide explains how to use the Oracle Key Vault client SDK to integrate Oracle and non-Oracle products directly with Oracle Key Vault.

This preface contains:

Audience

This document is intended for application developers using the C and Java programming languages to manage Oracle and non-Oracle heterogenous solutions for use with Oracle Key Vault.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents:

- Oracle Key Vault Administrator's Guide
- Oracle Key Vault Release Notes

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

`http://www.oracle.com/technetwork/index.html`

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN website at

`http://www.oracle.com/technetwork/documentation/index.html`

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.

Part I

Introduction to the Oracle Key Vault Client SDK

Part I provides an overview and explains who should use this guide and the advantages of the Oracle Key Vault client SDK.

- Getting Started with the Oracle Key Vault Client SDK
  The Oracle Key Vault client SDK is designed for C and Java programmers who understand Oracle Key Vault.
- KMIP Features of the Oracle Key Vault Client SDK
  The communication exchange between the Oracle Key Vault client SDK and the Oracle Key Vault server will make use of the KMIP protocol.
- Setting Up the Oracle Key Vault SDK
  The client SDK is available in both C and Java.
- Oracle Key Vault Client SDK Program Structure
  The Oracle Key Vault client SDK program structure covers areas such as the program flow, types, environment, connection, and session.

1

Getting Started with the Oracle Key Vault Client SDK

The Oracle Key Vault client SDK is designed for C and Java programmers who understand Oracle Key Vault.

1.1 About Getting Started with the Oracle Key Vault Client SDK

The Oracle Key Vault Client SDK provides C and Java APIs to create custom applications that enable Oracle and non-Oracle products to integrate directly with Oracle Key Vault. However, it is not designed to manage endpoints or to function as an encryption library.

The Oracle Key Vault Client SDK addresses product-specific key management issues.

The following are the features of the Oracle Key Vault Client SDK:

- Enables an endpoint program to access the Oracle Key Vault server and execute multiple KMIP operations on the Key Vault server objects.
- Available for C and Java platforms.
- Is designed to enable Oracle and non-Oracle products to manage keys, credentials, symmetric keys, and other secrets. Enables users to manage heterogeneous solutions. Users can create, register, retrieve, and delete objects, as well as add, delete, and modify attributes of objects.
- Supports authentication with the Oracle Key Vault server and also can use the Oracle Key Vault configuration files. Enables endpoints to use their own

connection management. The client SDK can communicate with the Key Vault server by using a mutually authenticated secure connection (TLS).

- Enables endpoints to make use of their own memory management.

1.2 Who Should Use This Guide

This guide is intended for proficient C and Java programmers who are adept Oracle Key Vault and Oracle Database administrative users.

1.3 Platforms Supported

Oracle Key Vault Software Development Kit is supported on various platforms depending on the programming language.

C

- Linux
- Solaris SPARC64
- Solaris x64
- AIX
- HP-UX

Java

- Platform Neutral

1.4 Advantages of Using the Oracle Key Vault Client SDK

Oracle Key Vault client SDK will allow an endpoint program to access the Oracle Key Vault server and be able to perform multiple KMIP operations on the objects stored in the Oracle Key Vault server.

The key advantages of using the Oracle Key Vault Client SDK are:

- Externalize Key Management to Oracle Key Vault.
- Support KMIP operation and objects.
- Simplified connection setup.
- Tight integration with endpoint enrollment.
- Easy to embed the SDK in an existing C or Java program.
- Easy to update existing code that interfaces with another key management provider, providing the full power of KMIP key management.
- Simple and intuitive to use.
- Complies with various regulations and mandates that cover physical separation of encryption keys and encrypted data. Externalizing key management provides this separation, hence security of the overall environment is enhanced.

2

KMIP Features of the Oracle Key Vault Client SDK

The communication exchange between the Oracle Key Vault client SDK and the Oracle Key Vault server will make use of the KMIP protocol.

The Key Vault Client SDK simplifies the KMIP exposure to the endpoint and supports additional functionality that makes it easier for the endpoints to communicate with the Oracle Key Vault server.

- KMIP Version
  The Oracle Key Vault client SDK supports Version 1.1 of the KMIP specification, limited to those objects and operations required by supported profiles.
- KMIP Profile Support
  The Oracle Key Vault client SDK supports four KMIP profiles.
- KMIP Managed Objects
  The Oracle Key Vault client SDK supports four KMIP managed objects.
- KMIP Operations
  The Oracle Key Vault client SDK supports 14 KMIP operations.

2.1 KMIP Version

The Oracle Key Vault client SDK supports Version 1.1 of the KMIP specification, limited to those objects and operations required by supported profiles.

2.2 KMIP Profile Support

The Oracle Key Vault client SDK supports four KMIP profiles.

The supported profiles are as follows:

- Basic Asymmetric Key and Certificate Store
- Basic Symmetric Key Foundry and Server
- Basic Symmetric Key Store and Server
- Secret Data

2.3 KMIP Managed Objects

The Oracle Key Vault client SDK supports four KMIP managed objects.

These managed objects are as follows:

- Opaque object
- Secret data

- Symmetric key
- Template

2.4 KMIP Operations

The Oracle Key Vault client SDK supports 14 KMIP operations.

These KMIP operations are as follows:

- `Create`
- `Register` (of keys, secrets, opaque objects and templates)
- `Rekey`
- `Locate`
- `Get` (of keys, secrets, opaque objects, and templates)
- `Get Attribute`
- `Get Attribute List`
- `Add Attribute`
- `Modify Attribute`
- `Delete Attribute`
- `Activate`
- `Revoke`
- `Destroy`
- `Query`

3

Setting Up the Oracle Key Vault SDK

The client SDK is available in both C and Java.

3.1 Enrolling an Endpoint

An Endpoint must be registered and enrolled to the Oracle Key Vault server before downloading the SDK content to that endpoint.

If the endpoint is not already registered and enrolled before downloading the SDK, please enroll the endpoint by following the instructions from Enrolling Endpoints for Oracle Key Vault.

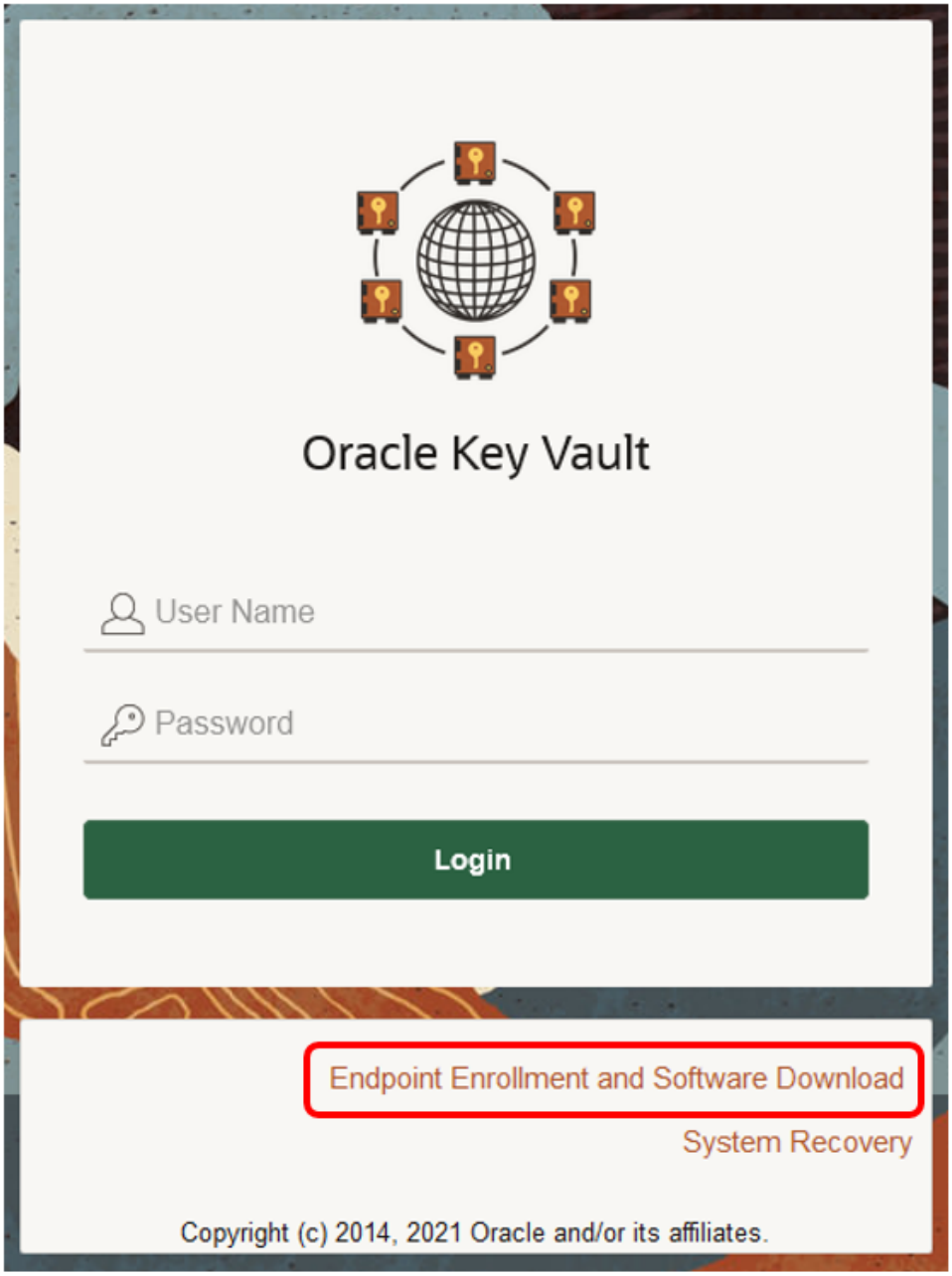
3.2 Downloading the C or Java SDK Software

You must download the appropriate Software Development Kit (SDK) software, either the C or Java version.

1. Access the Oracle Key Vault management console from the endpoint on which you wish to deploy the SDK.
2. The login page to the Oracle Key Vault management console appears.

   Do not log in.
3. Click the **Endpoint Enrollment and Software Download** link below the **Login** button.

Figure 3-1 Endpoint Enrollment and Software Download



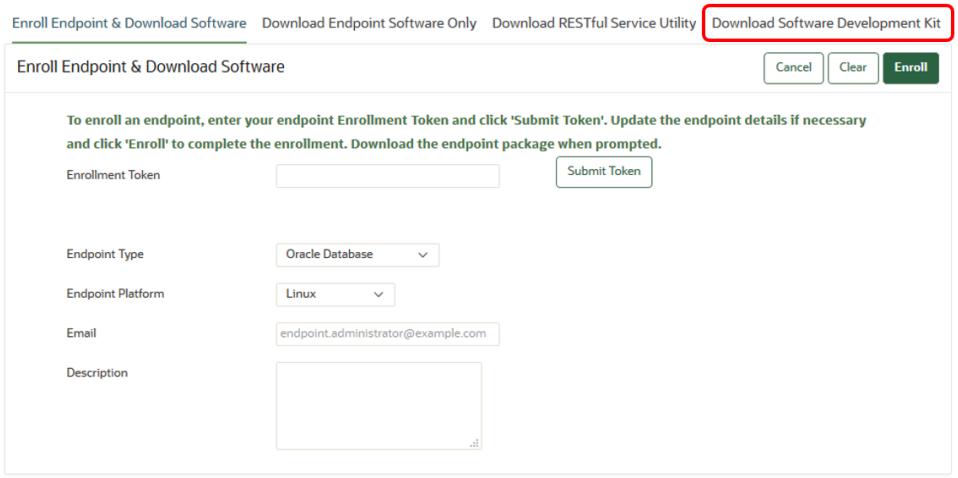
4. The **Enroll Endpoint & Download Software** screen appears.

   There are four tabs along the top.

   - Enroll Endpoint & Download Software
   - Download Endpoint Software Only
   - Download RESTful Service Utility
   - Download Software Development Kit

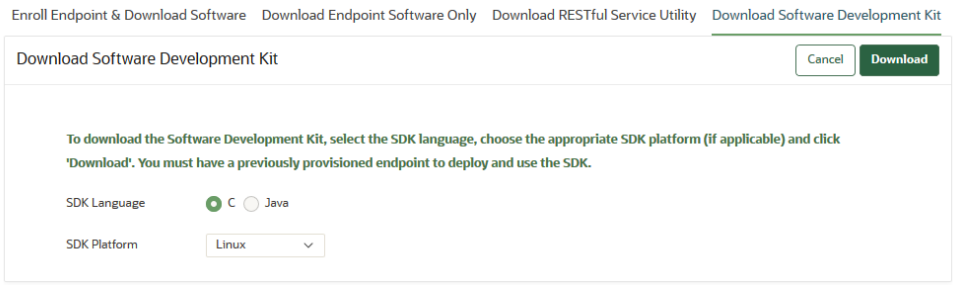
5. Click the **Download Software Development Kit** tab.

Figure 3-2 Download Software Development Kit



6. The Download Software Development Kit screen appears with the option to select either the C or Java SDK.

Figure 3-3 Select C as the SDK Language

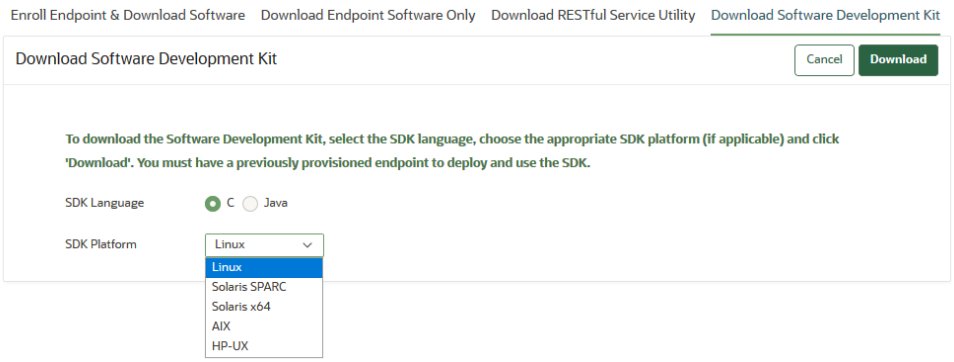


7. If you select the C SDK, you must select the platform for deployment.

   The platform options are:

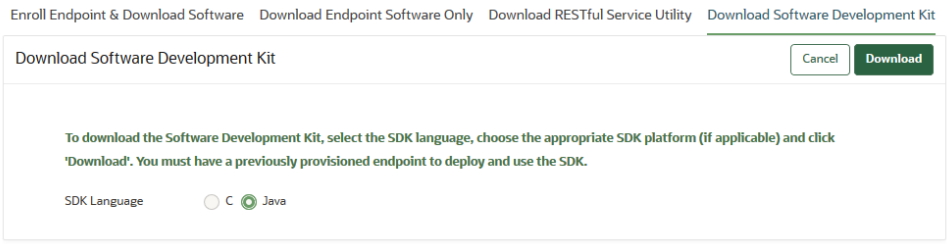
   - Linux
   - Solaris SPARC
   - Solaris x64
   - AIX
   - HP-UX

Figure 3-4 Select SDK Platform



If you choose the Java SDK, it is platform independent and does not require you to choose a platform.

Figure 3-5 Select Java as the SDK Language



8. Click **Download**.

   Save the SDK zip file to the desired location.

9. Ensure that you have the necessary administrative privileges to install software on the endpoint.

10. Check that the `OKV_HOME` environment variable is correctly set.

    See the README file included in the zip file for more information.

11. Navigate to the directory in which you saved the zip file.

12. Unzip the SDK file.

    For example, on Linux, to unzip the Java SDK file, use:

    ```
    unzip -o okv_jsdk.zip -d $OKV_HOME
    ```

    or for the C SDK file, use:

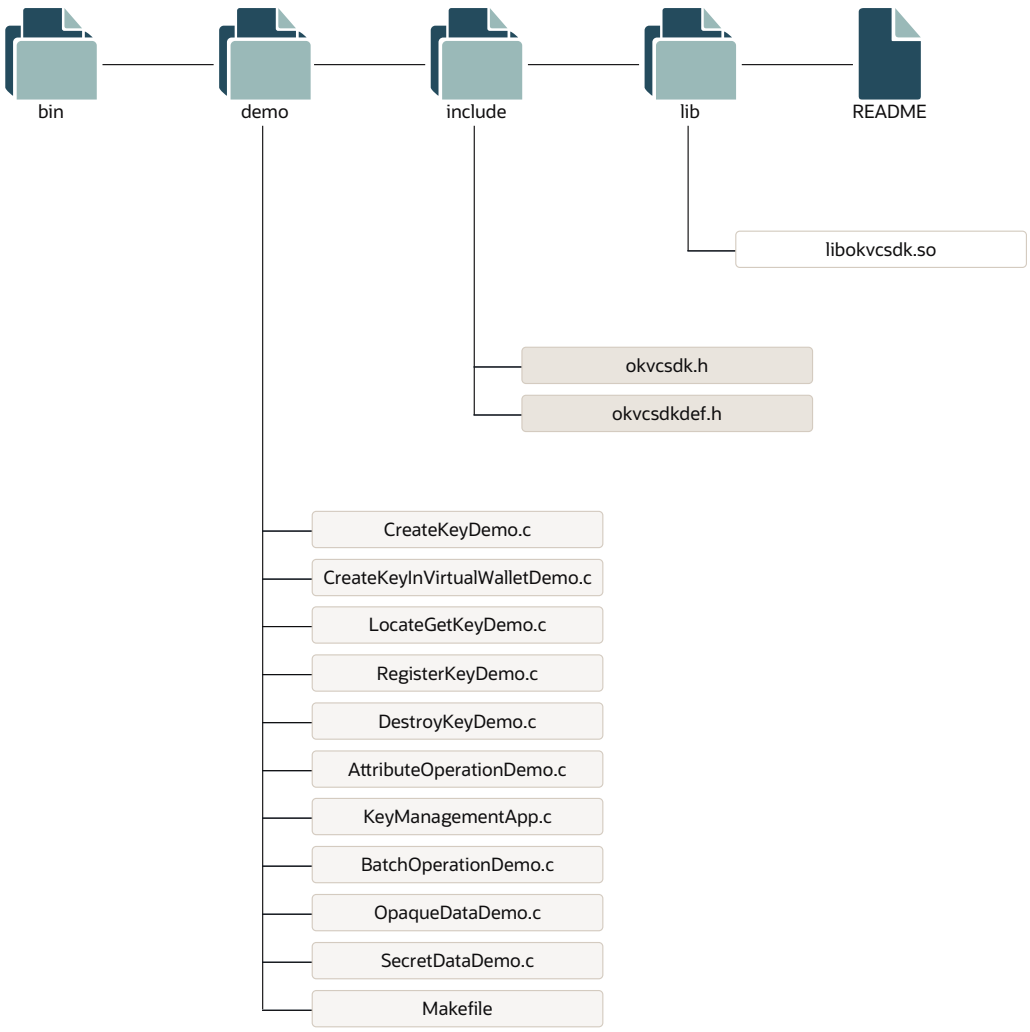
    ```
    unzip -o okv_csdk.zip -d $OKV_HOME
    ```

> **Note:**
>
> Oracle recommends you to deploy the SDK software contents under `OKV_HOME`. This applies even during redeployment or upgrade to new version of the SDK software.

3.3 Contents of the C SDK File

The contents of C SDK file include demo programs, the SDK library file, and other necessary files.

Figure 3-6 C SDK Directories and Files



The `include` directory contains header files for C SDK APIs. The `lib` directory contains the shared library object. `bin` is an empty directory to place the demo program object files and executables.

The `demo` directory contains sample programs which demonstrate the use of the C SDK APIs. The table below provides a brief description of each sample program.

Table 3-1 Sample Programs

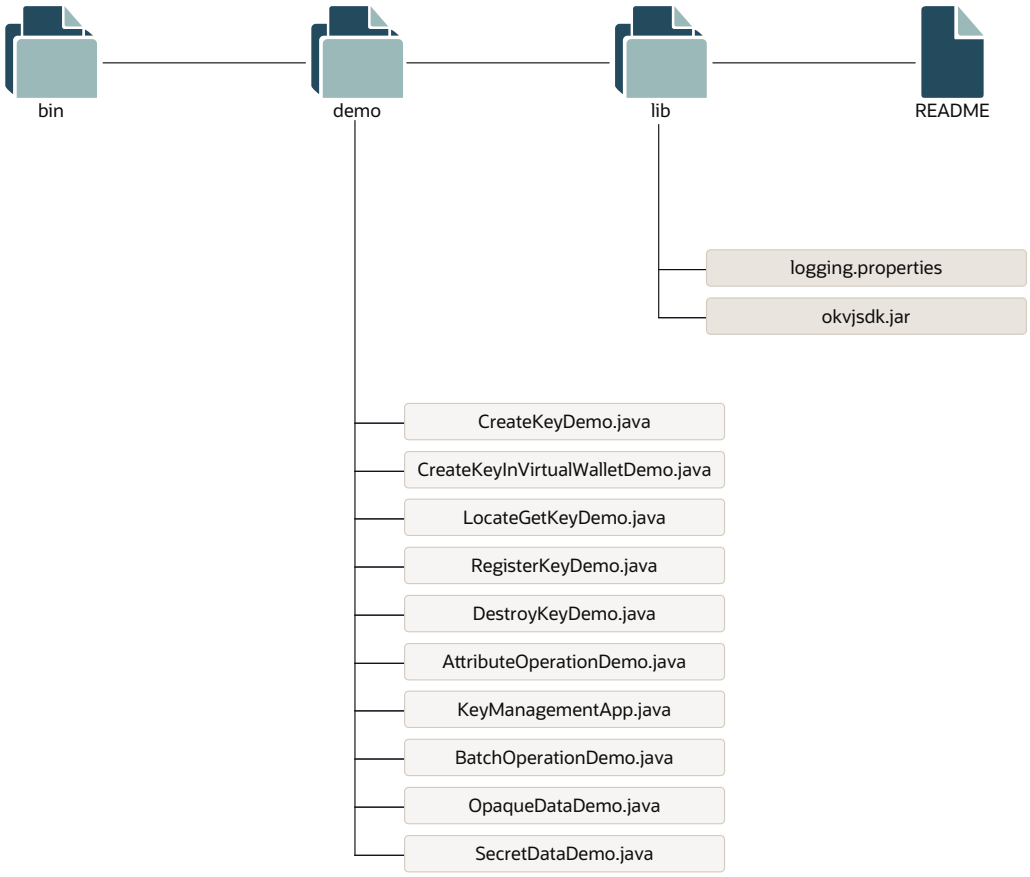
Sample Program	Description
`CreateKeyDemo.c`	Application for KMIP create and activate key operations.
`CreateKeyInVirtualWalletDemo.c`	Application for KMIP create key operation in the given wallet and activate the key.
`LocateGetKeyDemo.c`	Application for KMIP locate operation by given name and get the key details like key length, algorithm, and value.
`RegisterKeyDemo.c`	Application for KMIP register key operation.
`DestroyKeyDemo.c`	Application for KMIP locate operation by given name and destroy the key.
`AttributeOperationDemo.c`	Application for KMIP attribute operations such as add, modify, get, and delete attributes.
`KeyManagementApp.c`	Application for Key management operations such as create, activate, get, deactivate, and destroy key.
`BatchOperationDemo.c`	Application to demonstrate batching such as batching of create, activate, add attribute, get key, and destroy operations.
`OpaqueDataDemo.c`	Application for Opaque Data management operations such as register, get, add attribute, and destroy opaque data.
`SecretDataDemo.c`	Application for Secret Data management operations such as register, activate, add attribute, deactivate, and destroy secret data.

See the `README` file provided with the C SDK for complete instructions about environment configuration, compiling, and running the demo programs.

3.4 Contents of the Java SDK File

The contents of Java SDK file include demo programs, the SDK library jar file, and other necessary files.

Figure 3-7 Java SDK Directories and Files



The `lib` directory contains the Oracle Key Vault Client Java SDK jar file `okvjsdk.jar` and sample `java.util.logging` configuration file `logging.properties`. `bin` is an empty directory to place the Java demo program class files.

The `demo` directory contains sample programs which demonstrate the use of the Java SDK APIs. The table below provides a brief description of each sample program.

Table 3-2 Sample Programs

Sample Program	Description
`CreateKeyDemo.java`	Application for KMIP create and activate key operations.
`CreateKeyInVirtualWalletDemo.java`	Application for KMIP create key operation in the given wallet and activate the key.
`LocateGetKeyDemo.java`	Application for KMIP locate operation by given name and get the key details like key length, algorithm, and value.
`RegisterKeyDemo.java`	Application for KMIP register key operation.
`DestroyKeyDemo.java`	Application for KMIP locate operation by given name and destroy the key.
`AttributeOperationDemo.java`	Application for KMIP attribute operations such as add, modify, get, and delete attributes.

Table 3-2 (Cont.) Sample Programs

Sample Program	Description
`KeyManagementApp.java`	Application for Key management operations such as create, activate, get, deactivate, and destroy key.
`BatchOperationDemo.java`	Application to demonstrate batching such as batching of create, activate, add attribute, get key, and destroy operations.
`OpaqueDataDemo.java`	Application for Opaque Data management operations such as register, get, add attribute, and destroy opaque data.
`SecretDataDemo.java`	Application for Secret Data management operations such as register, activate, add attribute, deactivate, and destroy secret data.

See the `README` file provided with the Java SDK for complete instructions about environment configuration, compiling, and running the demo programs.

4

Oracle Key Vault Client SDK Program Structure

The Oracle Key Vault client SDK program structure covers areas such as the program flow, types, environment, connection, and session.

- About the Oracle Key Vault Client SDK Program Structure
  The program structure of an Oracle Key Vault SDK program describes the approach the Oracle Key Vault client will take to write an endpoint SDK program.
- Oracle Key Vault Program Flow
  The Oracle Key Vault general program flow includes creating the environment handle, followed by the connection and session, then termination and release.
- Oracle Key Vault Program Environment
  The Oracle Key Vault program environment is the region or block in the endpoint where the Oracle Key Vault functions are executed.
- Oracle Key Vault Program Connection
  The Oracle Key Vault program connection has two types of connections: explicit and intrinsic.
- Oracle Key Vault Program Session
  The Oracle Key Vault program session are of two types: Oracle Key Vault session and Oracle Key Vault call session.

4.1 About the Oracle Key Vault Client SDK Program Structure

The program structure of an Oracle Key Vault SDK program describes the approach the Oracle Key Vault client will take to write an endpoint SDK program.

The Oracle Key Vault SDK itself is written using certain assumptions and the Oracle Key Vault client must write the endpoint program code in a manner that is consistent with those assumptions.

The Oracle Key Vault client SDK is available for C and Java platforms. The Oracle Key Vault SDK follows the data types, calling conventions, syntax and semantics of the programming language.

The endpoint SDK program can be compiled and linked in the same manner as any other C or Java application. There is no need for any separate pre-processing or post compilations steps.

For an endpoint program to communicate with the Oracle Key Vault server, the endpoint SDK program must follow the program flow, link Oracle Key Vault client SDK library and account for the characteristics of the Oracle Key Vault program.

4.2 Oracle Key Vault Program Flow

The Oracle Key Vault general program flow includes creating the environment handle, followed by the connection and session, then termination and release.

Oracle Key Vault program flows are categorized by their structure and how much KMIP detail is involved to write them: basic and advanced.

- Basic Program Flow
  A basic program works seamlessly with KMIP programs.
- Advanced Program Flow
  The advanced program flow can be categorized as an Oracle Key Vault program with batching and detailed Oracle Key Vault program.

4.2.1 Basic Program Flow

A basic program works seamlessly with KMIP programs.

A basic Oracle Key Vault program has the following characteristics:

- It consists of Oracle Key Vault client SDK functions that perform broad KMIP operations such as creating, activating, retrieving, and registering keys or credentials with Oracle Key Vault server.
- The client need not be aware of the intricate KMIP details when using the functions.
- It contains functions to help the client focus on functionality such as creating and rotating a key, rather than handling details of the KMIP specification, setting up connections with Oracle Key Vault, and other operations.

Advantages of a basic program are as follows:

- It is use-case driven.
- It is KMIP agnostic. Very little KMIP knowledge is required to make use of these functions.
- It can be deployed quickly.
- It integrates easily.

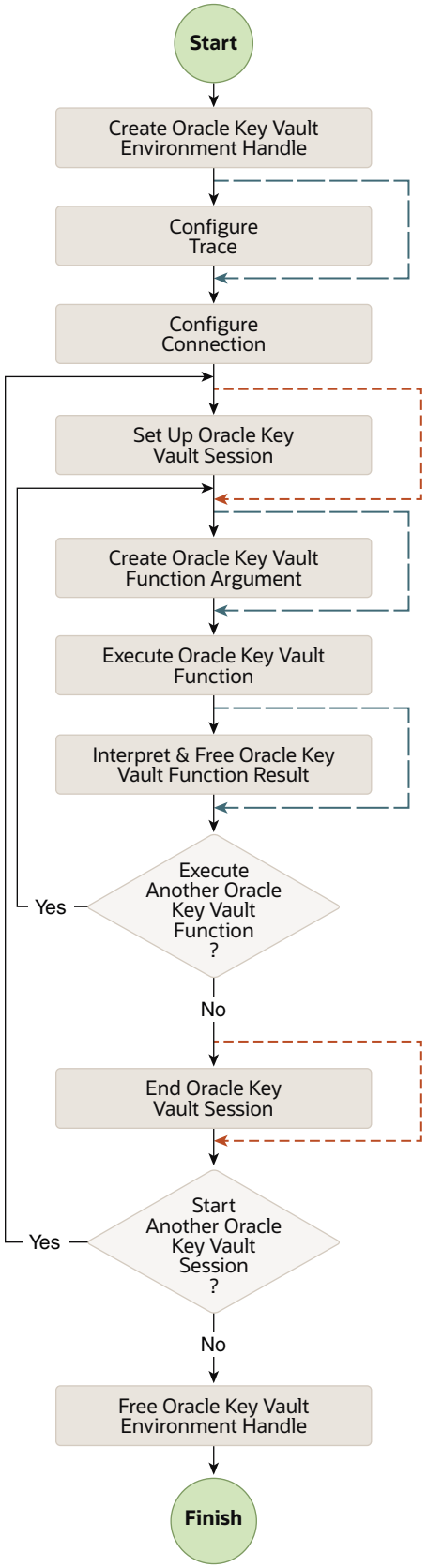
The disadvantage of a basic program is that not every KMIP detail is supported.

The basic program flow has the following structure:

1. Create the Oracle Key Vault environment handle.
2. Optionally, configure the trace information.
3. Configure the connection information.
4. Optionally, set up the Oracle Key Vault program session.
   - Create Oracle Key Vault operations or function arguments, if required.
   - Execute the Oracle Key Vault operations or functions.
   - Interpret and free up the Oracle Key Vault operations or function results if necessary.
5. Terminate the Oracle Key Vault program session if it had been established earlier.

6. Release the Oracle Key Vault environment handle.

Figure 4-1 Basic Oracle Key Vault Program Flowchart



Example 4-1 Basic Oracle Key Vault Program

```
...
OKVEnv  *env = okvEnvCreate(OKV_MEMORY_FUNCTIONS);
okvEnvSetConfig(env, (oratext *)config_file, (oratext *) connection_pwd);
okvEnvSetTrace(env, (oratext *)trace_dir, OKV_TRACE_DEBUG);
okvConnect(env);
okvCreateKey(env, ...,  unique_id, &unique_id_len);
printf("unique identifier %s", unique_id);
okvActivate(env, unique_id);
okvDisconnect(env);
okvDestroy(env, unique_id);
okvEnvFree(&env);
...
```

4.2.2 Advanced Program Flow

The advanced program flow can be categorized as an Oracle Key Vault program with batching and detailed Oracle Key Vault program.

- Oracle Key Vault Program with Batching
  Many KMIP operations such as locating and activating objects can be batched by using the `okvBatchCreate` API.
- Detailed Oracle Key Vault Program
  The detailed Oracle Key Vault program can cover most of the use cases that are allowed by the KMIP protocol.

4.2.2.1 Oracle Key Vault Program with Batching

Many KMIP operations such as locating and activating objects can be batched by using the `okvBatchCreate` API.

Batching two or more KMIP operations means that the KMIP request packets for both the operations will be aggregated and sent to the Oracle Key Vault server together in one super or batched KMIP packet. This reduces round trips for the information exchanged between the client SDK and the Oracle Key Vault server.

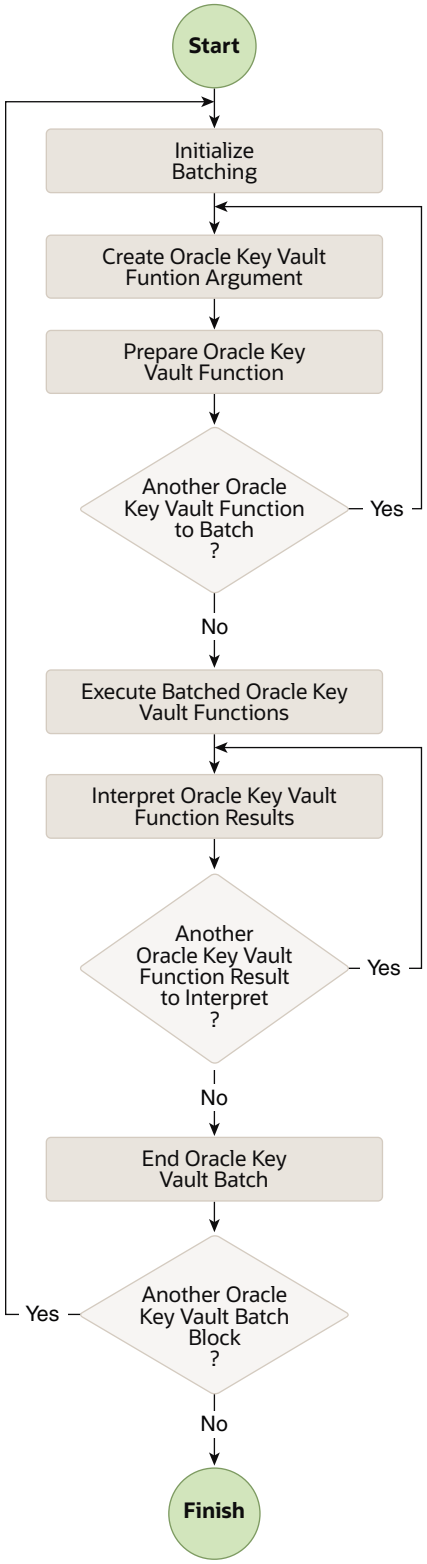
An Oracle Key Vault program with batching has the following structure:

1. Create the Oracle Key Vault environment handle.
2. Optionally, configure the trace information.
3. Configure the connection information.
   - Optionally, set up the Oracle Key Vault program session.
   - Start batching.
     - Create the Oracle Key Vault KMIP TTLV function arguments, if required.
     - Create the Oracle Key Vault functions.
   - Execute the batched operations. This will execute all the Oracle Key Vault functions and possibly process the data returned from the KMIP server.
     - Interpret any Oracle Key Vault function results, if required.
   - End batching.
   - Terminate the Oracle Key Vault program session if it was established earlier.

4. Release the Oracle Key Vault environment handle.

The Oracle Key Vault program flow with batching is same as basic program flow except for the Oracle Key Vault function execution part and is as shown in the figure below.

Figure 4-2 Advanced Oracle Key Vault Program with Batching Flowchart



Example 4-2 Advanced Oracle Key Vault Program With Batching

```
...
OKVEnv  *env = okvEnvCreate(OKV_MEMORY_FUNCTIONS);
okvEnvSetConfig(env, (oratext *)config_file, (oratext *) connection_pwd);
okvEnvSetTrace(env, (oratext *)trace_dir, OKV_TRACE_DEBUG);
okvBatchCreate(env);
okvCreateKey(env, ...,  unique_id, &unique_id_len);
okvActivate(env, (oratext *)NULL);
okvDestroy(env, (oratext *)NULL);
okvBatchExecute(env);
printf("unique identifier %s", unique_id);
okvBatchFree(env);
okvEnvFree(&env);
...
```

4.2.2.2 Detailed Oracle Key Vault Program

The detailed Oracle Key Vault program can cover most of the use cases that are allowed by the KMIP protocol.

A detailed Oracle Key Vault program consists of a set of Oracle Key Vault Client SDK functions that allow the client to write programs that are an actual representation of the KMIP packets being transported to the server. This requires the client to know KMIP protocol in some details.

The basic SDK program covers most of the common use cases. More complex operations or use cases can be done with a detailed Oracle Key Vault program.

The program structure is mostly the same as the basic Oracle Key Vault program structure. However, there is an additional operation handle that is used to define the KMIP operation.

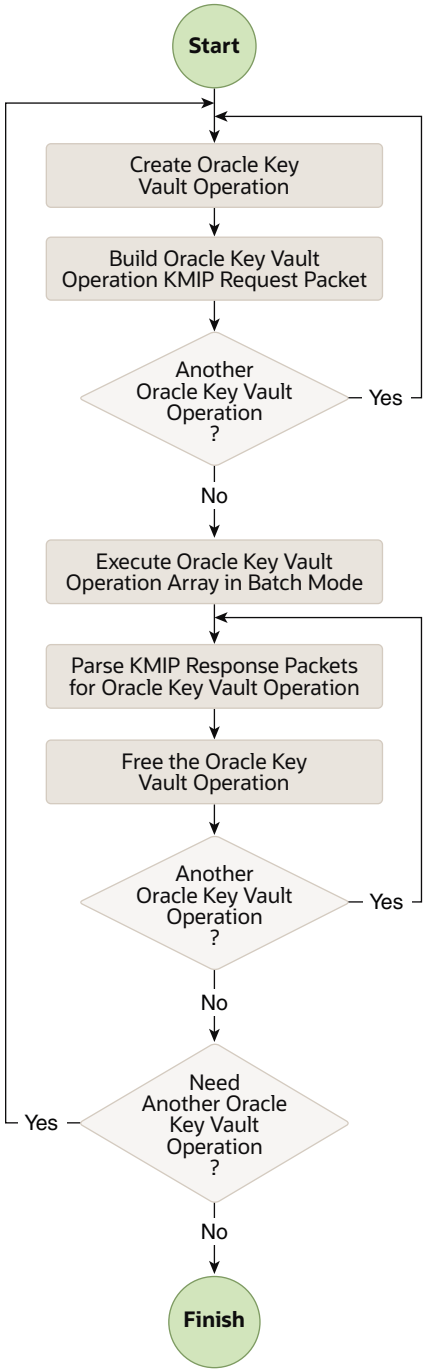
The Oracle Key Vault detailed program flow can also make use of batching the multiple KMIP operations.

A detailed Oracle Key Vault program has the following structure:

1. Create the Oracle Key Vault environment handle.
2. Optionally, configure the trace information.
3. Configure the connection information.
   - Optionally, configure the Oracle Key Vault program session.
     - Create an Oracle Key Vault operation array for multiple KMIP operations.
     - Build a KMIP request message for the Oracle Key Vault operations.
     - Execute the Oracle Key Vault operations, possibly in batch mode.
     - Unwrap the KMIP response message for the Oracle Key Vault operations.
     - Free up the Oracle Key Vault operation array.
   - Terminate the Oracle Key Vault program session if you had established it earlier.
4. Release the Oracle Key Vault environment handle.

The detailed Oracle Key Vault program flow is same as basic program flow except for the Oracle Key Vault function execution part and is as shown in the figure below.

Figure 4-3 Detailed Oracle Key Vault Program Flowchart



Example 4-3 Advanced Oracle Key Vault Detailed Program

```
...
OKVEnv  *env = okvEnvCreate(OKV_MEMORY_FUNCTIONS);

/* Allocate Operation array for two KMIP operations */
OKVOps  *ops[2];
ub4 alg = CRYPTO_ALG_AES;
okvEnvSetConfig(env, (oratext *)config_file, (oratext *) connection_pwd);
```

```
okvEnvSetTrace(env, (oratext *)trace_file, OKV_TRACE_DEBUG);

/* First OKV KMIP Operation */
ops[0] = okvOpsCreate(env, OKVOpCreate);
req = okvTTLVGetRequest(env, ops[0]);
okvTTLVAddToObject(env, req, OKVDEF_TAG_OBJ_TYPE, ...);
template = okvTTLVAddToObject(env, req, OKVDEF_TAG_TEMPLATE_ATTR_ST, ..);
attr = okvTTLVAddToObject(env, template, OKVDEF_TAG_ATTR_ST, ...);
okvTTLVAddToObject(env, attr, OKVDEF_TAG_ATTR_NAME, "Cryptographic
Algorithm" ...);
okvTTLVAddToObject(env, attr, OKVDEF_TAG_ATTR_INDEX, &ind, ..);
okvTTLVAddToObject(env, attr, OKVDEF_TAG_ATTR_VALUE, &alg, ..);

... so on for crypto algorithm len and crypto mask attributes ...

/* Second OKV KMIP Operation */
ops[1] = okvOpsCreate(env, OKVOpActivate);
req = okvTTLVGetRequest(env, ops[1]);

/* Note KMIP ID for activation is implicit since detailed operations are always
batched. */

/* Execute OKV KMIP Operations in Batch mode */
okvOpsExecuteOp(env, ops, 2);

/* Parse the result of the first OKV KMIP Operation */
resp = okvTTLVGetResponse(env, ops[0]);
tagid = okvTTLVGetChild(env, resp, 1, OKVDEF_TAG_ID, 0, &ctag_id_len);
ctag_id = okvTTLVGetValue(tagid);
ctag_id[ctag_id_len] = 0;
printf("\nCreated Key %s", ctag_id);

/* Parse the result of the second OKV KMIP Operation */
resp = okvTTLVGetResponse(env, ops[1]);
tagid = okvTTLVGetChild(env, resp, 0, OKVDEF_TAG_ID, 0, &atag_id_len);
atag_id = okvTTLVGetValue(tagid);
atag_id[atag_id_len] = 0;
printf("\nActivated Key %s", atag_id);

/* Free the two operations */
okvOpsFree(env, &ops[0]);
okvOpsFree(env, &ops[1]);
okvEnvFree(&env);
...
```

4.3 Oracle Key Vault Program Environment

The Oracle Key Vault program environment is the region or block in the endpoint where the Oracle Key Vault functions are executed.

The Oracle Key Vault program environment begins with the initialization of the Oracle Key Vault environment handle and ends with the freeing of this handle. The initialization and freeing of the Oracle Key Vault environment handle can be done in different endpoint program functions so the Oracle Key Vault environment exists across functions in the endpoint program.

The Oracle Key Vault program environment is limited to a process or thread only and, as such, the Oracle Key Vault environment will exist within a process or a thread only. The Oracle Key Vault environment handle must not be used in the threads or

processes forked after the Oracle Key Vault environment is initialized. The forked processes or threads can have their own Oracle Key Vault environments.

4.4 Oracle Key Vault Program Connection

The Oracle Key Vault program connection has two types of connections: explicit and intrinsic.

- **Explicit connections:** You must configure the connection that connects to the Oracle Key Vault server explicitly, perform a few Oracle Key Vault client SDK operations, and then disconnect from the Oracle Key Vault server.
- **Intrinsic connections:** You must configure the connection and then execute the Oracle Key Vault client SDK function. This function sets up the connections and then ends when the task is complete.

The difference between the two approaches is the number of times the connection between endpoint SDK Program and Oracle Key Vault server has to be setup. The connections are SSL connections and could prove to be costly for performance intensive applications.

There is a two-minute time-out on the Oracle Key Vault server connections if there is no activity on the connection. For explicit connections, the connection is reset automatically if it is disconnected.

4.5 Oracle Key Vault Program Session

The Oracle Key Vault program session are of two types: Oracle Key Vault session and Oracle Key Vault call session.

An Oracle Key Vault program session exists from the time the endpoint program explicitly connects to the Oracle Key Vault server, to the time it explicitly disconnects from the Oracle Key Vault server. This is described as the first case in the previous section (explicit connections). There can be multiple Oracle Key Vault function calls during an Oracle Key Vault session, that is, between the time the connection is setup and is disconnected.

The second case described in the previous section (intrinsic connections) sets up the connection only for the duration of one Oracle Key Vault client SDK function. This temporary internal session is called Oracle Key Vault program call session or Oracle Key Vault call session.

In general, if there are a number of KMIP operations to be executed together in time and program space, then it is better to set up the Oracle Key Vault session. If there are just one or two operations to be executed, and if they are batched, then there is no need to set up the Oracle Key Vault session explicitly.

Oracle Key Vault program sessions exist within an Oracle Key Vault program environment. More than one Oracle Key Vault session can exist within an Oracle Key Vault program environment in a serial fashion. One Oracle Key Vault session cannot be embedded in another Oracle Key Vault session. Since the scope of an Oracle Key Vault environment is a single process or a thread, the Oracle Key Vault session is also limited to a process or thread.

An Oracle Key Vault session by definition encompasses both batched and non-batched Oracle Key Vault functions. While batching saves the round trips to the Oracle

Key Vault server, an Oracle Key Vault session reduces the need to repeatedly set up a connection with the Oracle Key Vault server.

Part II

Oracle Key Vault Client C SDK API Reference

Part II provides information about the Oracle Key Vault C SDK APIs.

- Oracle Key Vault Datatypes and Structures
  This section describes the Oracle Key Vault datatypes and structures.
- Oracle Key Vault Management APIs
  The Oracle Key Vault management APIs control the Oracle Key Vault Interface program environment.
- Oracle Key Vault Client SDK Connection Management APIs
  This section describes the interfaces for Oracle Key Vault connection management.
- Oracle Key Vault Client SDK Memory Management APIs
  This section describes the Oracle Key Vault interfaces required to handle endpoint specified memory management.
- Oracle Key Vault Client SDK Error Handling APIs
  This section describes the interfaces for Oracle Key Vault error management.
- Oracle Key Vault Client SDK KMIP and Batch APIs
  The SDK KMIP APIs provide functions for creating keys, activating keys, adding attributes to keys, destroying keys, and other operations. The batch APIs enable you to perform these activities in a batch operation.
- Oracle Key Vault Client SDK KMIP Attributes and Custom Attributes APIs
  This section describes the interfaces that help create and interpret both KMIP attributes and KMIP custom attributes.
- Oracle Key Vault Extension Operation Management APIs
  Oracle Key Vault provides operations used to execute custom KMIP requests.
- Oracle Key Vault Client SDK TTLV Object APIs
  The SDK `TTLV` object APIs enable you to perform activities such as getting the child of an `0KVTTLV` object.
- Oracle Key Vault Client SDK Utility APIs
  You can use the Oracle Key Vault client SDK utility APIs with other Oracle Key Vault functions to simplify common operations.

5

Oracle Key Vault Datatypes and Structures

This section describes the Oracle Key Vault datatypes and structures.

- Oracle Key Vault Datatypes
  This section describes the datatypes provided with the Oracle Key Vault SDK.
- Oracle Key Vault Structures and Enumerations
  This section describes the structures and enumerations provided with the Oracle Key Vault SDK.

5.1 Oracle Key Vault Datatypes

This section describes the datatypes provided with the Oracle Key Vault SDK.

Oracle Key Vault client SDK defines a set of C datatypes that are used throughout the Oracle Key Vault client SDK. These definitions are available to the endpoint program upon inclusion of okvcsdk.h. The following table lists the datatypes and their description.

Table 5-1 Oracle Key Vault Datatypes

Datatype	Description
`ub1`	Unsigned byte of at least 1 byte.
`sb1`	Signed byte of at least 1 byte.
`ub2`	Unsigned byte of at least 2 bytes.
`sb2`	Signed byte of at least 2 bytes.
`ub4`	Unsigned byte of at least 4 bytes.
`sb4`	Signed byte of at least 4 bytes.
`ub8`	Unsigned byte of at least 8 bytes.
`sb8`	Signed byte of at least 8 bytes.
`OKVErrNo`	Same as ub4.
`OKVTag`	Same as ub4.
`OKVType`	Same as ub1.
`oratext`	Character byte of size 1 byte.

5.2 Oracle Key Vault Structures and Enumerations

This section describes the structures and enumerations provided with the Oracle Key Vault SDK.

- OKVAttr
  `OKVAttr` has a collection of all the KMIP attributes, single or multi-instance attributes, supported by the KMIP specification.

- OKVAttrNo
  `OKVAttrNo` defines the KMIP attributes with `OKVATTRMAX` as the count of the KMIP attributes.
- OKVEnv
  `OKVEnv` is the Oracle Key Vault environment handle that controls the endpoint SDK program behavior.
- OKVErr
  `OKVErr` is the Oracle Key Vault error management handle that captures errors in an Oracle Key Vault operation.
- OKVMemoryCtx
  `OKVMemoryCtx` is the Oracle Key Vault memory management context that holds the memory context and pointers to endpoint defined memory functions.
- OKVObjNo
  `OKVObjNo` defines the KMIP managed object types with `OKVOBJMAX` as the maximum possible count of the KMIP managed object types.
- OKVOps
  `OKVOps` is the Oracle Key Vault operation handle.
- OKVOpsNo
  `OKVOpsNo` defines the KMIP Operations with `OKVOPSMAX` as the count of the maximum possible KMIP operations.
- OKVServerInformation
  `OKVServerInformation` is the Oracle Key Vault specific information that is returned by the Oracle Key Vault server for the Oracle Key Vault query operation.
- OKVTTLV
  `OKVTTLV` defines the Oracle Key Vault structure for a TTLV object.

5.2.1 OKVAttr

`OKVAttr` has a collection of all the KMIP attributes, single or multi-instance attributes, supported by the KMIP specification.

Multi-instance attributes also have a field for the count of the multi-instance attribute. Attributes that have text string and byte string have a length associated with the value pointers.

Definition

```
/* Client SDK collection of attribtues */
struct OKVAttr
{
  struct
  {
    oratext *id;
    ub4      idl;
  } unique_identifier;
  ub4      name_count;
  struct
  {
    oratext *name;
    ub4      namel;
    ub4      type;
  } name[OKV_MAX_ATTR_INSTANCES];
  OKVObjNo object_type;
```

```
ub4 crypto_algorithm;
ub4 crypto_length;
ub4 crypto_parameters_count;
struct
{
  ub4 block_cipher_mode;
  ub4 padding_method;
  ub4 hashing_algorithm;
  ub4 key_role_type;
} crypto_parameters[OKV_MAX_ATTR_INSTANCES];
ub4 cert_type;
ub4 cert_length;
struct
{
  ub1 *issuer;
  ub4  issuerl;
  ub1 *serial_number;
  ub4  serial_numberl;
} X509_cert_identifier;
struct
{
  ub1 *distinguished_name;
  ub4  distinguished_namel;
  ub4  alternative_name_count;
  struct
  {
    ub1 *name;
    ub4  namel;
  } alternative_name[OKV_MAX_ALTERNATE_NAMES];
} X509_cert_subject;
struct
{
  ub1 *distinguished_name;
  ub4  distinguished_namel;
  ub4  alternative_name_count;
  struct
  {
    ub1 *name;
    ub4  namel;
  } alternative_name[OKV_MAX_ALTERNATE_NAMES];
} X509_cert_issuer;
ub4 digital_signature_algorithm_count;
ub4 digital_signature_algorithm[OKV_MAX_ATTR_INSTANCES];
ub4 digest_count;
struct
{
  ub4  hashing_algorithm;
  ub4  key_format_type;
  ub1 *digest_value;
  ub4  digest_valuel;
} digest[OKV_MAX_ATTR_INSTANCES];
ub4 crypto_usage_mask;
ub4 lease_time;
struct
{
  ub8 total;
  ub8 count;
  ub4 unit;
} usage_limits;
ub4 state;
ub8 initial_date;
```

```
  ub8 activation_date;
  ub8 process_start_date;
  ub8 protect_stop_date;
  ub8 deactivation_date;
  ub8 destroy_date;
  ub8 compromise_occurrence_date;
  ub8 compromise_date;
  struct
  {
    ub4 reason_code;
    oratext *message;
    ub4      messagel;
  } revocation_reason;
  ub8 archive_date;
  ub8 fresh;
  ub4 link_count;
  struct
  {
    ub4 type;
    oratext *linked_object_identifier;
    ub4      linked_object_identifierl;
  } link[OKV_MAX_ATTR_INSTANCES];
  ub8 last_change_date;

  /* Un-Supported Attributes * /

    Crypto Domain Parameters
    Cert_Identifier
    Cert_Subject
    Cert_Issuer
    Object_Group[]
    Contact_Information
    Application_Specific_Information[]
    Operation_Policy_Name
  */
};
typedef struct OKVAttr OKVAttr;
```

5.2.2 OKVAttrNo

`OKVAttrNo` defines the KMIP attributes with `OKVATTRMAX` as the count of the KMIP attributes.

Definition

```
/* KMIP Attributes */
typedef enum
{
  OKVAttrNone = 0,
  OKVAttrUniqueId,
  OKVAttrName,
  OKVAttrObjType,
  OKVAttrCryptoAlg,
  OKVAttrCryptoLen,
  OKVAttrCryptoParams,
  OKVAttrCryptoDomainParams,
  OKVAttrCertType,
  OKVAttrCertLength,
  OKVAttrX509CertId,
  OKVAttrX509CertSubject,
```

```
  OKVAttrX509CertIssuer,
  OKVAttrCertId,
  OKVAttrCertSubject,
  OKVAttrCertIssuer,
  OKVAttrDigitalSignAlgo,
  OKVAttrDigest,
  OKVAttrOpsPolicyName,
  OKVAttrCryptoUsageMask,
  OKVAttrLeaseTime,
  OKVAttrUsageLimits,
  OKVAttrState,
  OKVAttrInitialDate,
  OKVAttrActivationDate,
  OKVAttrProcessStartDate,
  OKVAttrProtectStopDate,
  OKVAttrDeactivationDate,
  OKVAttrDestroyDate,
  OKVAttrCompromiseOccurenceDate,
  OKVAttrCompromiseDate,
  OKVAttrRevocationReason,
  OKVAttrArchiveDate,
  OKVAttrObjectGroup,
  OKVAttrFresh,
  OKVAttrLink,
  OKVAttrAppSpecificInfo,
  OKVAttrContactInfo,
  OKVAttrLastChangeDate,
  OKVAttrInvalid = 255
} OKVAttrNo;
#define OKVATTRMAX  33
```

5.2.3 OKVEnv

`OKVEnv` is the Oracle Key Vault environment handle that controls the endpoint SDK program behavior.

`OKVEnv` also holds Service Provider Interfaces (SPI) handles used in the endpoint SDK program, the request and result OKVTTLV objects for Oracle Key Vault functions.

Definition

```
/* Oracle Key Vault Environment */
struct OKVEnv
{
   OKVConnCtx      *conn_spi;
   OKVMemoryCtx    *mem_spi;
   OKVParseCtx     *parse_spi;
   ub4              flag;
   #define OKVENV_CONN_SETUP  0x00000001
   #define OKVENV_BATCH_MODE  0x00000002
   #define OKVENV_CONN_SPI    0x00000004
   #define OKVENV_NATCONN_SPI 0x00000008
   #define OKVENV_MEM_SPI     0x00000010
   #define OKVENV_NATMEM_SPI  0x00000020
   #define OKVENV_PACK_XML    0x00000040
   OKVTTLV         *request_obj;
   OKVTTLV         *result_obj;
   OKVErr          *err;
   OKVTrcCtx       *trc_ctx;
   ub4              batch_cnt;
```

```
   OKVBatchCtx    **batch;
   ub4              batch_err_ctx_cnt;
   OKVBatchErrCtx **batch_err_ctx;
};
typedef struct OKVEnv OKVEnv;
```

Parameters

Parameter	Description
`conn_spi`	Stores the handle for the connection management SPI. If the endpoint program does not specify one then it stores the handle for the native connection management.
`mem_spi`	Stores the handle for the memory management SPI. If the endpoint program does not specify one then it stores the handle for the native memory management.
`parse_spi`	Stores the context for parse management. Since the serialization of OKVTTLV objects is internal to Oracle Key Vault Client SDK, `parse_spi` is set and unset internally only. KMIP v1.1 can have a regular TTLV packing or XML style packing. In the first version of Oracle Key Vault Client SDK, TTLV packing is supported.
`flag`	Controls the operational behavior of the Oracle Key Vault client SDK program. Most of the flags are self explanatory.
`request_obj`	For Oracle Key Vault API functions having OKVTTLV objects as arguments, the object is created beforehand. The allocated memory for this object is pointed to by `request_obj`.
`result_obj`	For Oracle Key Vault API functions that return OKVTTLV objects, the object has to be interpreted by the EndPoint program after the call is done i.e. the memory for the Oracle Key Vault API functions is cleaned up. The memory for the OKVTTLV object is not cleaned at the Oracle Key Vault function call and is pointed to by `result_obj`.
`err`	This is the error handle that captures the errors in Oracle Key Vault operation. Multiple errors can be reported for a given operation. These errors will be captured in the error stack.
`trc_ctx`	Stores the handle for trace management.
`batch_cnt`	This is the count of batch operations.
`batch`	This is the array of batch operations along with the place holders for results.
`batch_err_ctx_cnt`	This is the count of batch error context.
`batch_err_ctx`	Batch error context will hold the information such as Oracle Key Vault operation name and errors related to that operation if any.

5.2.4 OKVErr

`OKVErr` is the Oracle Key Vault error management handle that captures errors in an Oracle Key Vault operation.

Multiple errors can be reported for a given operation. These errors will be captured in the error stack.

Definition

```
/* Oracle Key Vault Error Management */
struct OKVErr
{
   #define OKVERR_CNT 100
   ub1 err_cnt;
   ub4 err_stack[OKVERR_CNT];
};
typedef struct OKVErr OKVErr;
```

Parameters

Parameter	Description
`err_cnt`	Count of errors in the error stack, which indicates the depth of the error stack.
`err_stack`	Stack of error numbers captured.

5.2.5 OKVMemoryCtx

`OKVMemoryCtx` is the Oracle Key Vault memory management context that holds the memory context and pointers to endpoint defined memory functions.

It also holds pointers to the `malloc`, `realloc`, and `free` functions supplied by the endpoint program.

Definition

```
/* Memory Function Context */
struct OKVMemoryCtx
{
   void     *ctx;                                                  /* Context */
   void *  (*okvMalloc)(void *ctx, size_t size);                   /* Malloc */
   void *  (*okvRealloc)(void *ctx, void **ptr, size_t size);      /* Realloc */
   void    (*okvFree)(void *ctx, void **ptr);                      /* Free */
};
typedef struct OKVMemoryCtx OKVMemoryCtx;
```

Parameters

Parameter	Description
`ctx`	The endpoint program defined memory context.
`okvMalloc`	Pointer to the endpoint program defined function to allocate memory. This function should clear the memory allocated, that is, set all allocated bytes to zero.
`okvRealloc`	Pointer to the endpoint program defined function to re-allocate the size of the previously allocated and possibly populated memory.
`okvFree`	Pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`. The pointer should be set to `NULL` after freeing it.

Related Topics

- okvFree
  `okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`.
- okvMalloc
  `okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.
- okvRealloc
  `okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory.

5.2.6 OKVObjNo

`OKVObjNo` defines the KMIP managed object types with `OKVOBJMAX` as the maximum possible count of the KMIP managed object types.

Definition

```
/* OKV KMIP Managed Objects */
typedef enum
{
   OKVObjNone = 0,          /* No Object Type */
   OKVObjCert = 1,          /* Certificate */
   OKVObjSymmetric,         /* Symmetric Key */
   OKVObjPublic,            /* Public Key */
   OKVObjPrivate,           /* Private Key */
   OKVObjTemplate = 6,      /* Template */
   OKVObjSecret,            /* Secret Data */
   OKVObjOpaque             /* Opaque Object */
} OKVObjNo;
#define OKVOBJMAX          8
```

5.2.7 OKVOps

`OKVOps` is the Oracle Key Vault operation handle.

Definition

```
/* Oracle Key Vault KMIP Operation */
struct OKVOps
{
  OKVOpsNo      ops;
  OKVErr        err;
  OKVTTLV       *item;
  OKVTTLV       *req;
  ub4           res;
  OKVTTLV       *resp;
  OKVErr        *errb;
};
typedef struct OKVOps OKVOps;
```

OKVOps captures the request and response OKVTTLV structures for a given Oracle Key Vault KMIP operation along with the result (pass or fail) of the operation.

Parameters

Parameter	Description
ops	KMIP operation associated with this Oracle Key Vault operation handle.
err	Error handle for batch operations.
item	Batch item of this KMIP operation.
req	KMIP Request OKVTTLV object.
res	Result of the KMIP operation.
resp	KMIP Response OKVTTLV object.
errb	Error handle pointer for batch operations.

5.2.8 OKVOpsNo

`OKVOpsNo` defines the KMIP Operations with `OKVOPSMAX` as the count of the maximum possible KMIP operations.

Definition

```
/* KMIP Operations */
typedef enum
{
   OKVOpNone = 0,                  /* Wrong Operation */
   OKVOpCreate = 1,                /* Create */
   OKVOpCreateKeyPair,
   OKVOpRegister,                  /* Register */
   OKVOpRekey,                     /* Rekey */
   OKVOpDeriveKey,
   OKVOpCertify,
   OKVOpRecertify,
   OKVOpLocate,                    /* Locate */
   OKVOpCheck,                     /* Check */
   OKVOpGet,                       /* Get */
   OKVOpGetAttributes,             /* Get Attributes */
   OKVOpGetAttributeList,          /* Get Attribute List */
   OKVOpAddAttribute,              /* Add Attribute */
   OKVOpModifyAttribute,           /* Modify Attribute */
   OKVOpDeleteAttribute,           /* Delete Attribute */
   OKVOpObtainLease,
   OKVOpGetUsageAllocation,
   OKVOpActivate,                  /* Activate */
   OKVOpRevoke,                    /* Revoke */
   OKVOpDestroy,                   /* Destroy */
   OKVOpArchive,
   OKVOpRecover,
   OKVOpValidate,
   OKVOpQuery,                     /* Query */
   OKVOpCancel,
   OKVOpPoll,
   OKVOpNotify,
   OKVOpPut,
   OKVOpRekeyKeyPair,
   OKVOpDiscoverVersions           /* Discover Versions */
```

```
} OKVOpsNo;
#define OKVOPSMAX                  30
```

5.2.9 OKVServerInformation

`OKVServerInformation` is the Oracle Key Vault specific information that is returned by the Oracle Key Vault server for the Oracle Key Vault query operation.

Definition

```
struct OKVServerInformation
{
   oratext server_name[30];
   oratext server_version[30];
};
typedef struct OKVServerInformation OKVServerInformation;
```

Parameters

Parameter	Description
`server_name`	Should be ORACLE KEYVAULT SERVER if the endpoint program is communicating with the Oracle Key Vault server.
`server_version`	The version of the Oracle Key Vault server the endpoint program is communicating with.

5.2.10 OKVTTLV

`OKVTTLV` defines the Oracle Key Vault structure for a TTLV object.

Definition

```
/* Oracle Key Vault KMIP TTLV Structure */
struct OKVTTLV
{
   OKVTag    tag;
   OKVType   typ;
   ub4       len;
   ub1      *val;
   ub4       ttlv_array_cnt;
   OKVTTLV **ttlv_array;
};
typedef struct OKVTTLV OKVTTLV;
```

Parameters

Parameter	Description
`tag`	The tag value of the TTLV object.
`typ`	The type value of the TTLV object.
`len`	The length of the value of the TTLV object.
`val`	The value of the TTLV object.
`ttlv_array_cnt`	The count of the child TTLV objects for this TTLV object.
`ttlv_array`	An array of the child TTLV objects for this TTLV object.

6
Oracle Key Vault Management APIs

The Oracle Key Vault management APIs control the Oracle Key Vault Interface program environment.

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.
- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.1 okvEnvCreate

`okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.

Category

Management API

Purpose

`okvEnvCreate` is used to create the Oracle Key Vault environment handle which in turn holds error handle, tracing context and the various Service Provider Interface handles required by the Endpoint program. All of the Oracle Key Vault functions are done within the purview of this handle. The Oracle Key Vault environment begins with a successful call to this function.

Syntax

```
OKVEnv *okvEnvCreate(OKVMemoryCtx *memctx);
```

Parameters

Parameter	IN/OUT	Description
memctx	IN	Oracle Key Vault memory management context.

Return Values

Return Value	Description
OKVEnv*	Oracle Key Vault environment handle. **Success:** A valid pointer to the Oracle Key Vault environment handle is returned. **Failure:** A NULL pointer is returned.

Comments

Endpoints using the Oracle Key Vault memory management should pass in OKV_MEMORY_FUNCTIONS to okvEnvCreate.

The Oracle Key Vault client SDK provides the client with an option to define their own memory management context and memory management functions.

Suppose the client has defined clientMalloc(), clientRealloc(), clientFree(), and clientCtxP for memory management. The client can make use of these functions by initializing the Oracle Key Vault environment handle with the Oracle Key Vault memory context structure.

```
...
OKVMemoryCtx clientMemory;
clientMemory.ctx = clientCtxP;
clientMemory.okvMalloc = &clientMalloc;
clientMemory.okvRealloc = &clientRealloc;
clientMemory.okvFree = &clientFree;
okvEnvCreate(&clientMemory);
...
```

Example

```
OKVEnv  *env = (OKVEnv *) NULL;

/* Setup the Oracle Key Vault Environment handle */
printf("\nSetting up Oracle Key Vault Environment handle\n");
env = okvEnvCreate(OKV_MEMORY_FUNCTIONS);
if (!env)
{
   printf("\tCould not setup the Oracle Key Vault Environment handle\n");
   return 1;
}
else
{
   printf("\tSuccessfully setup Oracle Key Vault Environment handle\n\n");
}
```

Related Topics

- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.
- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.2 okvEnvFree

`okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.

Category

Management API

Purpose

`okvEnvFree` is used to free the Oracle Key Vault environment handle. It marks the end of Oracle Key Vault environment.

Syntax

```
void okvEnvFree(OKVEnv **env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault Interface environment handle.

Return Values

No values returned.

Comments

None.

Example

```
OKVEnv  *env = (OKVEnv *) NULL;

/* Setup the Oracle Key Vault Environment handle */
printf("\nSetting up Oracle Key Vault Environment handle\n");
```

```
env = okvEnvCreate(OKV_MEMORY_FUNCTIONS);

if (!env)
{
   printf("\tCould not setup the Oracle Key Vault Environment handle\n");
   return 1;
}
else
{
   printf("\tSuccessfully setup Oracle Key Vault Environment handle\n\n");
}
...
/* Do some KMIP operations */
...
/* Free the env handle */
printf("Releasing the Environment handle\n");
okvEnvFree(&env);
return 0;
```

Related Topics

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.3 okvEnvFreeResultObj

`okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.

Category

Management API

Purpose

`okvEnvFreeResultObj` will free the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.

Oracle Key Vault KMIP functions at times return OKVTTLV descriptors that have to be explored further either using Oracle Key Vault helper or Oracle Key Vault KMIP Extension functions. The OKVTTLV parent for the result descriptor is stored in Oracle Key Vault environment handle and is not freed upon completion of the Oracle Key Vault function.

After the Oracle Key Vault function result descriptor has been interpreted, it can be freed using `okvEnvFreeResultObj`.

Syntax

```
void okvEnvFreeResultObj(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault Interface environment handle.

Return Values

No values returned.

Comments

The subsequent Oracle Key Vault functions calls will free the result descriptor of the previous Oracle Key Vault function.

Example

```
/* Setup the Oracle Key Vault Environment handle 'env' */
...
/* Do some KMIP operations */
...
okvEnvFreeResultObj(env);
```

Related Topics

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.4 okvEnvGetOpRequestObj

`okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.

Category

Management API

Purpose

Some KMIP functions will use an OKVTTLV descriptor argument to build the TTLV Request. `okvEnvGetOpRequestObj` is used to get the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.

Oracle Key Vault functions like `okvAttrAddUniqueID` will create an OKVTTLV attribute descriptor but these functions need a parent OKVTTLV descriptor. `okvEnvGetOpRequestObj()` returns the parent OKVTTLV descriptor for such Oracle Key Vault functions.

Syntax

```
OKVTTLV *okvEnvGetOpRequestObj(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	`IN`	Oracle Key Vault environment handle.

Return Values

Return Value	Description
`OKVTTLV *`	OKVTTLV object. **Success:** A valid pointer to the OKVTTLV descriptor is returned. **Failure:** A `NULL` pointer is returned.

Comments

None.

Example

```
OKVTTLV *request = (OKVTTLV *)NULL;
...
/* Setup the Oracle Key Vault Environment handle 'env' */
...
request = okvEnvGetOpRequestObj(env);
```

Related Topics

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.
- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.

- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.5 okvEnvSetConfig

`okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.

Category

Management API

Purpose

`okvEnvSetConfig` is used to specify the password for the SSL connection wallet. The SSL connection wallet will be created at the time of enrollment. The password specified at the time of endpoint enrollment is the one used with this function. If no password was specified at the time of enrollment, `NULL` should be used as the password.

`okvEnvSetConfig` will also specify the configuration file that will be used to set up the connection configuration for holding details like the server IP address, server port number, and other information which will be needed during establishing connection with the Oracle Key Vault server.

Syntax

```
OKVErrNo okvEnvSetConfig(OKVEnv *env, oratext *conn_config_file,
                         oratext *conn_pwd);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`conn_config_file`	`IN`	Oracle Key Vault connection configuration file.
`conn_pwd`	`IN`	Password of the SSL connection wallet.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The connection configuration file is the `okvclient.ora` file, which is created at the time of Oracle Key Vault endpoint enrollment.

If you do not specify the full path to the connection configuration file, then the API tries to look for configuration file under `$OKV_HOME/conf`. Make sure to set up the `$OKV_HOME` to the directory where the endpoint software was installed.

Example

```
oratext *pwd = (oratext *)NULL;
...
/* Setup the Oracle Key Vault Environment handle 'env' */
...
printf("Setting up Oracle Key Vault connection configuration\n");
okvEnvSetConfig(env, (oratext *)NULL, (oratext *)pwd);
```

Related Topics

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.
- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

6.6 okvEnvSetTrace

`okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

Category

Management API

Purpose

`okvEnvSetTrace` is used to specify the location of the trace file and also the trace level. The trace file will record the internal traces generated by the endpoint SDK program and can be useful for diagnostics and debugging. The level of tracing determines how fine-grained or coarse-grained the tracing should be. In ascending order, debug levels take the following values:

- `OKV_TRACE_OFF` - No tracing
- `OKV_TRACE_FATAL` - Fatal level only tracing

- `OKV_TRACE_ERROR` - Error tracing
- `OKV_TRACE_WARN` - Warning tracing
- `OKV_TRACE_INFO` – Informational tracing
- `OKV_TRACE_DEBUG` – Debug tracing
- `OKV_TRACE_ALL` - All tracing

Syntax

```
OKVErrNo okvEnvSetTrace(OKVEnv *env, oratext *trace_file, ub1 debug_level);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`trace_file`	`IN`	Location of client SDK trace file.
`debug_level`	`IN`	Level of tracing.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
oratext *trace_dir = (oratext *)NULL;
...
/* Setup the Oracle Key Vault Environment handle 'env' */
...
trace_dir = (oratext *)"/tmp";
printf("Setting up Tracing\n");
okvEnvSetTrace(env, (oratext *)trace_dir, OKV_TRACE_INFO);
```

Related Topics

- okvEnvCreate
  `okvEnvCreate` creates the Oracle Key Vault environment handle and the Oracle Key Vault SDK program begins with a successful call to this function.
- okvEnvFree
  `okvEnvFree` frees the Oracle Key Vault environment handle, which marks the end of the Oracle Key Vault environment.

- okvEnvFreeResultObj
  `okvEnvFreeResultObj` frees the OKVTTLV result descriptor that was returned by the Oracle Key Vault functions.
- okvEnvGetOpRequestObj
  `okvEnvGetOpRequestObj` gets the root (top-level parent) of the OKVTTLV descriptor for Oracle Key Vault KMIP functions.
- okvEnvSetConfig
  `okvEnvSetConfig` sets up the Oracle Key Vault connection configuration context.

7

Oracle Key Vault Client SDK Connection Management APIs

This section describes the interfaces for Oracle Key Vault connection management.

- okvConnect
  `okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.
- okvConnSendRecvBytes
  `okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.
- okvConnSet
  `okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.
- okvConnUnSet
  `okvConnUnSet` is used to free the client connection provider.
- okvDisconnect
  `okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

7.1 okvConnect

`okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.

Category

Connection management API

Purpose

`okvConnect` is used to begin the Oracle Key Vault session and create a connection to the Oracle Key Vault Server. Subsequent Oracle Key Vault functions will use this connection for communicating with the Oracle Key Vault Server.

Syntax

```
OKVErrNo okvConnect(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVErrNo err_no;
...
/* Setup the Oracle Key Vault Environment handle 'env' */
...
printf("Setting up Oracle Key Vault session\n");
err_no = okvConnect(env);

if (err_no)
{
   printf("Could not setup the Oracle Key Vault session\n");
   return 1;
}
else
{
   printf("Successfully setup Oracle Key Vault Session\n");
}
```

Related Topics

- okvConnSendRecvBytes
  `okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.
- okvConnSet
  `okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.
- okvConnUnSet
  `okvConnUnSet` is used to free the client connection provider.
- okvDisconnect
  `okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

7.2 okvConnSendRecvBytes

`okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.

Category

Connection management API

Purpose

`okvConnSendRecvBytes` is used to send a KMIP request message to the Oracle Key Vault server and retrieve the KMIP response message sent back from the Oracle Key Vault server. The response returned by the Oracle Key Vault server cannot exceed the maximum payload supported between the Oracle Key Vault client and server.

Syntax

```
OKVErrNo okvConnSendRecvBytes(OKVEnv *env,
                              ub1 *reqmsg, ub4 reqmsgl,
                              ub1 **respmsg, ub4 *respmsgl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`reqmsg`	`IN`	KMIP TTLV request message.
`reqmsg1`	`IN`	KMIP TTLV request message length.
`respmsg`	`IN/OUT`	KMIP TTLV response message.
`respmsg1`	`IN/OUT`	KMIP TTLV response message length.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The endpoint program needs to allocate space for the response message. If the response message cannot fit in the allocated space the function will return an error.

Example

```
OKVErrNo err_no;
...
```

```
/* Setup the Oracle Key Vault Environment handle 'env' */
...
err_no = okvConnSendRecvBytes(env, (ub1 *)NULL, (ub4)0, (ub1 **)NULL, (ub4 *)0);

if (err_no)
{
   printf("Error while executing okvConnSendRecvBytes\n");
   return 1;
}
else
{
   printf("Successfully executed okvConnSendRecvBytes\n");
}
```

Related Topics

- okvConnect
  `okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.
- okvConnSet
  `okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.
- okvConnUnSet
  `okvConnUnSet` is used to free the client connection provider.
- okvDisconnect
  `okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

7.3 okvConnSet

`okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.

Category

Connection management API

Purpose

`okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server. Subsequent Oracle Key Vault functions will use this connection for communicating with the Oracle Key Vault server.

Syntax

```
OKVErrNo okvConnSet(OKVEnv *env,
                    void *connCtx,
                    OKVErrNo (*connectFn)(void *ctx),
                    void (*disconnectFn)(void *ctx),
                    OKVErrNo (*sendRecvFn)(void *ctx,
                                           ub1 *send_bytes,
                                           ub4 send_bytes_len,
                                           ub1 **recv_bytes,
                                           ub4 *recv_bytes_len));
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
connCtx	IN	Client defined connection context.
connectFn	IN	Client defined connect function.
disconnectFn	IN	Client defined disconnect function.
sendRecvFn	IN	Client defined send response to Oracle Key Vault server and receive response from Oracle Key Vault server function.
ctx	IN	Client defined connection context.
send_bytes	IN	KMIP TTLV request message.
send_bytes_len	IN	KMIP TTLV request message length.
recv_bytes	IN/OUT	KMIP TTLV response message.
recv_bytes_len	IN/OUT	KMIP TTLV response message length.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The Oracle Key Vault Client SDK also provides native connection management support. The endpoint program does not have to do anything special if it wants the Oracle Key Vault client SDK to take care of connection management.

Example

```
/* Suppose the client has defined clientConnect(), clientDisconnect(),
   clientSendRecv() and clientNatCtxP for connection management. The
   client can make use of these functions by initializing the Oracle Key
   Vault environment handle with the Oracle Key Vault connection context
   structure */
...
OKVErrNo err_no;
err_no = okvConnSet(env, clientNatCtxP, clientConnect(...),
clientDisconnect(...), clientSendRecv(...));
```

```
if (err_no)
{
   printf("Error while setting the client defined connection functions to
environment handle\n");
}
else
{
   printf("Successfully set the environment handle with client defined
connection functions");
}
```

Related Topics

- okvConnect
  `okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.
- okvConnSendRecvBytes
  `okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.
- okvConnUnSet
  `okvConnUnSet` is used to free the client connection provider.
- okvDisconnect
  `okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

7.4 okvConnUnSet

`okvConnUnSet` is used to free the client connection provider.

Category

Connection management API

Purpose

`okvConnUnSet` is used to free the client connection provider.

Syntax

```
void okvConnUnSet(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

No values returned.

Comments

The Oracle Key Vault client SDK also provides native connection management support. The endpoint program does not have to do anything special if it wants the Oracle Key Vault client SDK to take care of connection management.

Example

```
okvConnUnSet(env);
```

Related Topics

- okvConnect
  `okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.
- okvConnSendRecvBytes
  `okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.
- okvConnSet
  `okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.
- okvDisconnect
  `okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

7.5 okvDisconnect

`okvDisconnect` ends the Oracle Key Vault interface session and disconnects the Secure Sockets Layer (SSL) connection between the endpoint and the Oracle Key Vault server.

Category

Connection management API

Purpose

`okvDisconnect` ends the Oracle Key Vault session and will disconnect SSL connection between the endpoint program and the Oracle Key Vault server.

Syntax

```
OKVErrNo okvDisconnect(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVErrNo err_no;
printf("Disconnnecting Oracle Key Vault Session\n");
err_no = okvDisconnect(env);

if (err_no)
{
   printf("Could not disconnect from Oracle Key Vault session\n");
   return 1;
}
else
{
   printf("Successfully disconnected from Oracle Key Vault Session\n");
}
```

Related Topics

- okvConnect
  `okvConnect` begins the Oracle Key Vault interface session and creates a connection to the Oracle Key Vault server.
- okvConnSendRecvBytes
  `okvConnSendRecvBytes` sends a KMIP request message to the Oracle Key Vault server and retrieves the KMIP response message sent back from the Oracle Key Vault server.
- okvConnSet
  `okvConnSet` can be used to set up the user client connection provider which can be used to establish connection to the Oracle Key Vault server.
- okvConnUnSet
  `okvConnUnSet` is used to free the client connection provider.

8

Oracle Key Vault Client SDK Memory Management APIs

This section describes the Oracle Key Vault interfaces required to handle endpoint specified memory management.

- okvFree
  `okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`.
- okvMalloc
  `okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.
- okvRealloc
  `okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory.

8.1 okvFree

`okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`.

Category

Memory management API

Purpose

`okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`. The pointer should be set to NULL after freeing it.

Syntax

```
void (*okvFree)(void *ctx, void **ptr);
```

Parameters

Parameter	IN/OUT	Description
`ctx`	`IN`	Memory context (context of `OKVMemoryCtx`)
`ptr`	`IN`	Previously allocated memory

Return Values

No values returned.

Comments

None.

Related Topics

- okvMalloc
  `okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.
- okvRealloc
  `okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory.

8.2 okvMalloc

`okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.

Category

Memory management API

Purpose

`okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.

Syntax

```
void * (*okvMalloc)(void *ctx, size_t size);
```

Parameters

Parameter	IN/OUT	Description
`ctx`	`IN`	Memory context of `OKVMemoryCtx`.
`size`	`IN`	Size of memory to be allocated.

Return Values

Return Value	Description
`void*`	Memory pointer to allocated memory. **Success:** A valid pointer to the allocated memory is returned. **Failure:** A NULL pointer is returned if the function cannot allocate memory of requested size.

Comments

None.

Related Topics

- okvFree
  `okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`.
- okvRealloc
  `okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory.

8.3 okvRealloc

`okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory.

Category

Memory management API

Purpose

`okvRealloc` is a pointer to the endpoint program defined function that should reallocate memory. The data in the previously allocated memory should be copied to the newly allocated memory.

Syntax

```
void * (*okvRealloc)(void *ctx, void **ptr, size_t size);
```

Parameters

Parameter	IN/OUT	Description
`ctx`	`IN`	Memory context (context of `OKVMemoryCtx`).
`ptr`	`IN`	Previously allocated memory.
`size`	`IN`	Size of the memory to be allocated.

Return Values

Return Value	Description
`void*`	Memory pointer to reallocated memory. **Success:** A valid pointer to the re-allocated memory is returned. **Failure:** A NULL pointer is returned if the function cannot allocate memory of requested size.

Comments

None.

Related Topics

- okvFree
  `okvFree` is a pointer to the endpoint program defined function to free the memory allocated using `okvMalloc` or `okvRealloc`.
- okvMalloc
  `okvMalloc` is a pointer to the endpoint program defined function that should allocate memory.

9

Oracle Key Vault Client SDK Error Handling APIs

This section describes the interfaces for Oracle Key Vault error management.

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.1 okvErrGetDepth

`okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.

Catetory

Error handling API

Purpose

`okvErrGetDepth` is used to return the depth of the error stack otherwise 0 is returned if no error was recorded in the error stack.

Syntax

```
ub1 okvErrGetDepth(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
ub1	Oracle Key Vault error stack depth. **Success:** A non-zero positive number, which is the depth of the error stack, is returned. **Failure:** Zero is returned, since depth is zero if no error is recorded.

Comments

None.

Example

```
ub4 err_depth = okvErrGetDepth(env);

if (err_depth == 0)
{
   printf("No Error \n");
}
else
{
   printf("Error\n");
}
```

Related Topics

- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.2 okvErrGetDepthForBatch

`okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.

Catetory

Error handling API

Purpose

`okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number, otherwise 0 is returned.

Syntax

```
ub1 okvErrGetDepthForBatch(OKVEnv *env, ub4 batchjobnum);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`batchjobnum`	`IN`	Batch job number.

Return Values

Return Value	Description
`ub1`	Oracle Key Vault error stack depth. **Success:** A non-zero positive number which is the depth of the error stack for the respective batch job is returned. **Failure:** Zero is returned, since depth is zero if no error is recorded for the respective batch job.

Comments

Here `batchjobnum` is the number in which the operations are batched. For example, if the user wants to create a key, activate a key, revoke it, and then destroy it, all of these operations are batched in the same order. That is, `batchjobnum` for create a key is `1`, `batchjobnum` for activate a key is `2`, `batchjobnum` for revoke is `3` and `batchjobnum` for destroy is `4`. Once the batch job execution finishes, the user can pass in the batch job number for `okvErrGetDepthForBatch` to get the depth of the errors from the respective batch job error stack.

If batch job number provided is invalid, then zero (0) is returned by this API.

Example

```
ub4 err_depth = 0;
okvBatchCreate(env);
okvCreateKey(...); /* 1st Batch Job */
okvActivate(...);  /* 2nd Batch Job */
```

```
okvRevoke(...);    /* 3rd Batch Job */
okvDestroy(...);   /* 4th Batch Job */
okvBatchExecute(env);

/* Check error on top of the error stack for 1st batch job */
if (okvErrGetNumForBatch(env, 1))
{
   printf("Found error for 1st batch job");
   err_depth = okvErrGetDepthForBatch(env, 1);
   printf("Depth of the error stack is %d", err_depth);
}
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.3 okvErrGetNum

`okvErrGetNum` returns the error number on top of the error stack.

Catetory

Error handling API

Purpose

`okvErrGetNum` returns the error number on top of the error stack. `OKV_SUCCESS` is returned if no error recorded.

Syntax

```
OKVErrNo okvErrGetNum(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** The error number on top of the error stack is returned. **Failure:** OKV_SUCCESS (0) is returned if there is no error.

Comments

None.

Example

```
printf("Setting up Oracle Key Vault session\n");
okvConnect(env);

if (okvErrGetNum(env))
{
   printf("Could not setup the Oracle Key Vault session\n"};
   return 1;
}
else
{
   printf("Successfully setup Oracle Key Vault session\n\n");
}
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.

- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.4 okvErrGetNumAtDepth

`okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.

Catetory

Error handling API

Purpose

`okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack otherwise `OKV_SUCCESS` is returned.

Syntax

```
OKVErrNo okvErrGetNumAtDepth(OKVEnv *env, ub1 depth);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`depth`	`IN`	Depth at which the error number is needed.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** The error number at the specified depth in the error stack is returned. **Otherwise:** `OKV_SUCCESS` (0) is returned if there is no error.

Comments

`okvErrGetNumAtDepth` returns `OKV_SUCCESS` if the depth specified is more than that returned by `okvErrGetDepth`.

Example

```
ub4 err_depth = okvErrGetDepth(env);

while (err_depth > 0)
{
   ub4 error = 0;

   /* Get error number to get error text */
```

```
   error = okvErrGetNumAtDepth(env, err_depth--);
   printf("Error %d: %s \n", error, okvGetTextForErrNum(error));
}
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.5 okvErrGetNumAtDepthForBatch

`okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.

Catetory

Error handling API

Purpose

`okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number, otherwise `OKV_SUCCESS` is returned.

Syntax

```
OKVErrNo okvErrGetNumAtDepthForBatch(OKVEnv *env, ub4 batchjobnum,
                                     ub1 depth);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`batchjobnum`	`IN`	Batch job number.
`depth`	`IN`	Depth at which the error number is needed.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** The error number at the specified depth in the error stack is returned for the respective batch job number. **Otherwise:** OKV_SUCCESS (0) is returned if there is no error.

Comments

batchjobnum is the number in which the operations are batched. For example, if the user wants to create a key, activate a key, revoke it, and then destroy it, all of these operations are batched in the same order; that is, batchjobnum for create a key is 1, batchjobnum for activate a key is 2, batchjobnum for revoke is 3, and batchjobnum for destroy is 4. Once the batch execution finishes, the user can pass in the batch job number and the depth at which the error is needed for okvErrGetNumAtDepthForBatch to get the error details.

If the batch job number provided is invalid, then OKV_SUCCESS (0) is returned by this API. okvErrGetNumAtDepthForBatch returns OKV_SUCCESS if the depth specified is more than that returned by okvErrGetDepthForBatch.

Example

```
ub4 err_depth = 0;
okvBatchCreate(env);
okvCreateKey(...); /* 1st Batch Job */
okvActivate(...);  /* 2nd Batch Job */
okvRevoke(...);    /* 3rd Batch Job */
okvDestroy(...);   /* 4th Batch Job */
okvBatchExecute(env);

/* Check error on top of the error stack for 1st batch job */
if (okvErrGetNumForBatch(env, 1))
{
   printf("Found error for 1st Batch Job");
   err_depth = okvErrGetDepthForBatch(env, 1);
   printf("Depth of the error stack is %d", err_depth);

   while (err_depth > 0)
   {
      ub4 error = 0;
      /* Get error number to get error text */
      error = okvErrGetNumAtDepthForBatch(env, 1, err_depth--);
      printf("Error %d: %s \n", error, okvGetTextForErrNum(error));
   }
}
```

Related Topics

- okvErrGetDepth
  okvErrGetDepth is used to return the depth of the error stack otherwise 0 is returned.

- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.6 okvErrGetNumForBatch

`okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.

Catetory

Error handling API

Purpose

`okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number. `OKV_SUCCESS` is returned if no error recorded.

Syntax

```
OKVErrNo okvErrGetNumForBatch(OKVEnv *env, ub4 batchjobnum);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`batchjobnum`	`IN`	Batch job number.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** The error number on top of the error stack for the respective batch job is returned. **Failure:** `OKV_SUCCESS` (0) is returned if there is no error.

Comments

`batchjobnum` is the number in which the operations are batched. For example, if the user wants to create a key, activate a key, revoke it and then destroy it, all of these operations are batched in the same order. That is, `batchjobnum` for create a key is `1`, `batchjobnum` for activate a key is `2`, `batchjobnum` for revoke is `3`, and `batchjobnum` for destroy is `4`. Once the batch execution finishes, the user can pass in the batch job number for `okvErrGetNumForBatch` to get the error from the top of the respective batch job error stack.

If batch job number provided is invalid, then `OKV_SUCCESS` (0) is returned by this API.

Example

```
okvBatchCreate(env);
okvCreateKey(...); /* 1st Batch Job */
okvActivate(...);  /* 2nd Batch Job */
okvRevoke(...);    /* 3rd Batch Job */
okvDestroy(...);   /* 4th Batch Job */
okvBatchExecute(env);

/* Check error on top of the error stack for 1st Batch Job */
if (okvErrGetNumForBatch(env, 1))
{
   printf("Found error for 1st Batch Job");
}
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.7 okvErrReset

`okvErrReset` resets the error stack in the environment handle.

Catetory

Error handling API

Purpose

`okvErrReset` resets the error stack in the environment handle.

Syntax

```
void okvErrReset(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

No values returned.

Comments

None.

Example

```
/* Do some KMIP operations like adding attribute to an object and get the same
   attributes, below is an illustration of using 'okvErrReset' when getting
   the individual attibutes
*/

ub4 umask = 0;
ub4 state = 0;
printf("Trying to get Crypto Usage Mask value from State Attribute\n");
okvAttrGetCryptoUsageMask(env, ttlv, &umask);

if (okvErrGetNum(env))
/* Check for errors */
okvErrReset(env);   /* this will come in handy when we don't want to keep
                       getting the same errors for the next set of API
                       calls where we try to get the individual Attribute
                       values */

printf("Trying to get State value from State Attribute\n");
okvAttrGetState(env, ttlv, &state);

if (okvErrGetNum(env))
/* Check for errors */
okvErrReset(env);
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvGetTextForErrNum
  `okvGetTextForErrNum` returns the text of the error number.

9.8 okvGetTextForErrNum

`okvGetTextForErrNum` returns the text of the error number.

Category

Error handling API

Purpose

`okvGetTextForErrNum` returns the text of the the error number. If the error number is not valid, a pointer to `NULL` is returned.

Syntax

```
oratext *okvGetTextForErrNum(OKVErrNo err_no);
```

Parameters

Parameter	IN/OUT	Description
`errno`	`IN`	Oracle Key Vault error number

Return Values

Return Value	Description
`oratext *`	Oracle Key Vault error text. **Success:** Pointer to text of the error is returned. **Failure:** `NULL` pointer is returned if the error number is not valid.

Comments

None.

Example

```
ub4 error = 0;
printf("Error %d: %s \n", error, okvGetTextForErrNum(error));
```

Related Topics

- okvErrGetDepth
  `okvErrGetDepth` is used to return the depth of the error stack otherwise `0` is returned.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNum
  `okvErrGetNum` returns the error number on top of the error stack.
- okvErrGetNumAtDepth
  `okvErrGetNumAtDepth` returns the error number at the specified depth of the error stack.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrReset
  `okvErrReset` resets the error stack in the environment handle.

10

Oracle Key Vault Client SDK KMIP and Batch APIs

The SDK KMIP APIs provide functions for creating keys, activating keys, adding attributes to keys, destroying keys, and other operations. The batch APIs enable you to perform these activities in a batch operation.

- Oracle Key Vault Client SDK KMIP APIs
  This section describes the interfaces for the Oracle Key Vault functions for KMIP operations.
- Oracle Key Vault Client SDK Batch APIs
  This section describes the interfaces for the Oracle Key Vault KMIP batch functions.

10.1 Oracle Key Vault Client SDK KMIP APIs

This section describes the interfaces for the Oracle Key Vault functions for KMIP operations.

- About the Oracle Key Vault Client SDK KMIP APIs
  The Oracle Key Vault KMIP APIs enable you to perform actions such as managing keys and secret data.
- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.

- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.1 About the Oracle Key Vault Client SDK KMIP APIs

The Oracle Key Vault KMIP APIs enable you to perform actions such as managing keys and secret data.

Many of these functions take unique identifier as an input argument. As per KMIP v1.1, the unique identifier can be optional when the commands are batched given that the KMIP server can determine a single unique identifier from the previous commands. So if the previous command like locate returns only one unique identifier for possible use-cases, and the following commands operate on this unique identifier, the unique identifier can be omitted from the following commands.

The unique identifiers, along with their lengths, will be returned from the KMIP API functions. The unique identifiers needs to be NULL terminated by the endpoint program.

The endpoint program can pass in a larger buffer and the length will be used to determine the actual unique identifier bits in the larger buffer.

10.1.2 okvActivate

`okvActivate` implements the KMIP Activate operation.

Category

KMIP API

Purpose

`okvActivate` implements the KMIP Activate operation. It will activate the KMIP Object identified by unique identifier.

Syntax

```
OKVErrNo okvActivate(OKVEnv *env, oratext *uid);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create the KMIP object, say Key for example and get its unique
   identifier as part of its creation in ‘uid’ and then activate
   it as below
*/

okvActivate(env, &uid[0]);

if (okvErrGetNum(env))
{
   printf("Error while activating the object the object\n");
}
```

Related Topics

- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.

- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.3 okvAddAttribute

`okvAddAttribute` implements the KMIP add attribute operation.

Catetory

KMIP API

Purpose

`okvAddAttribute` implements the KMIP add attribute operation. It adds an attribute to the KMIP Object specified by the unique identifier.

Syntax

```
OKVErrNo okvAddAttribute(OKVEnv *env, oratext *uid,
                         OKVTTLV **attr);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).
`attr`	`IN`	Attribute object to be added to the KMIP object.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The API has the below exceptions when used with Oracle Key Vault server:

- If attributes are added to opaque or template objects using this API, the Oracle Key Vault server will give a general failure error and the attributes will not be added to the KMIP object. As a workaround the attributes can be added during the registration using APIs `okvRegOpaqueData` and `okvRegTemplate`.
- If a single instance attribute is added to an object using this API and the object already has an instance of the attribute, then the attribute won't get overwritten, but a general failure error will be thrown. Please use the `okvModifyAttribute` API to modify attribute values.
- If a protect stop date later than the deactivation date is passed in request TTLV, the server will not give an error and the protect stop date will be added to registered KMIP object.
- If a process start date is added using this API, the server will give permission denied error. As a workaround the attribute can be added during registration using APIs `okvRegKey`, `okvRegSecretData`, `okvRegOpaqueData`, and `okvRegTemplate`.

Example

```
/* Create the KMIP object, say Key for example and get its unique
   identifier as part of its creation in ‘uid’, we can add an attribute
   to it as shown below */

OKVTTLV *attr_in = (OKVTTLV *)NULL;
OKVTTLV *req = (OKVTTLV *)NULL;
oratext *name_value = (oratext *)"attribute_name";
ub4 name_valuel = strlen("attribute_name");
req = okvEnvGetOpRequestObj(env);
attr_in = okvAddAttributeObject(env, req, OKVAttrName, (ub4) 0);

okvAttrAddName(env, attr_in, name_value, name_valuel, 1);
okvAddAttribute(env, &uid[0], &attr_in);

if (okvErrGetNum(env))
{
   printf("Error while adding Name attribute to the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.

- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.4 okvCreateKey

`okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.

Category

KMIP API

Purpose

`okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object. The unique identifier of the symmetric key object generated by the OKV server is returned as `ouid`. The maximum length of the unique ID for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

Syntax

```
OKVErrNo okvCreateKey(OKVEnv *env,
                      OKVType alg, ub4 len, ub4 mask,
                      OKVTTLV *template_names_attrs,
                      oratext *wallet_name, ub4 wallet_namel,
                      oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`alg`	`IN`	Symmetric key algorithm.
`len`	`IN`	Key length of the symmetric key.
`mask`	`IN`	Cryptographic mask usage of the symmetric key.

Parameter	IN/OUT	Description
`template_names_attrs`	`IN`	Template names or attributes that will form the template-attribute.
`wallet_name`	`IN`	Wallet name value.
`wallet_namel`	`IN`	Length of the wallet name value.
`ouid`	`OUT`	Unique identifier of the generated key.
`ouidl`	`OUT`	Unique identifier length of the generated key.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

It is recommended to add a KMIP Name attribute to symmetric key object to help identify it in future.

The caller has to allocate the memory for the unique identifier returned by this function. The length of allocated memory is passed in via the length of the unique identifier argument `ouidl`. Sufficient memory to store the returned unique identifier should be allocated.

A symmetric key of the specified length for the specified algorithm and for a specific use (cryptographic usage mask) is generated in wallet `wallet_name` if specified else it is generated in the default wallet if present with the endpoint.

Example

```
/* Parameters to create symmetric key */
/* Tag for Cryptographic Algortihm AES */
OKVType  algo = CRYPTO_ALG_AES;

/* Key length 128, because AES keys are 128, 192 or 256 bits in length*/
ub4      key_len = 128;
ub4      usage_mask = CRYPTO_MASK_ENCRYPT | CRYPTO_MASK_DECRYPT;
oratext  uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4      uidl = sizeof(uid);
OKVTTLV *template = (OKVTTLV *)NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;

/* Set up the environment handle 'env' and also the memory and connection
   management as shown in previous sections */
memset(uid, 0, uidl);
template = okvEnvGetOpRequestObj(env);

/* Add name attribute object to template attribute */
attr_in = okvAddAttributeObject(env, template, OKVAttrName, (ub4) 0);
okvAttrAddName(env, attr_in, (oratext *)"My New Key", strlen("My New Key"), 1);
printf("\tCreating a Symmetric key\n");
```

```
okvCreateKey(env, algo, key_len, usage_mask, template,
             (oratext *)NULL, (ub4)0, &uid[0], uidl);

if (okvErrGetNum(env))
{
   printf("Error while creating the key\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.

- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.5 okvDeleteAttribute

`okvDeleteAttribute` implements the KMIP delete attribute operation.

Catetory

KMIP API

Purpose

`okvDeleteAttribute` implements the KMIP delete attribute operation. It deletes an attribute specified by an attribute name and attribute index of the KMIP object specified by the unique identifier.

Syntax

```
OKVErrNo okvDeleteAttribute(OKVEnv *env, oratext *uid,
                            oratext *attr_name,
                            ub4 attr_index,
                            OKVTTLV **attr);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).
`attr_name`	`IN`	Name of attribute to be deleted.
`attr_index`	`IN`	Index of the attribute to be deleted.
`attr`	`OUT`	Attribute object deleted from the KMIP object.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create the KMIP object, say Key for example and get its unique identifier
   as part of its creation in 'uid'. Add contact information attribute and
   to delete it, it can done as shown below */

OKVTTLV *attr_del = (OKVTTLV *) NULL;
oratext *attr_name = okvGetTextForAttributeNum(OKVAttrContactInfo);

/* Passing Contact Info attribute to be deleted at attribute index 0 */
okvDeleteAttribute(env, &uid[0], attr_name, (ub4)0, &attr_del);

if (okvErrGetNum(env))
{
   printf("Error while deleting the contact info attribute of the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.

- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.6 okvDestroy

`okvDestroy` implements the KMIP destroy operation.

Catetory

KMIP API

Purpose

`okvDestroy` implements the KMIP destroy operation. It will destroy the KMIP object specified by unique identifier.

Syntax

```
OKVErrNo okvDestroy(OKVEnv *env, oratext *uid);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create the KMIP object, say Key for example and get its unique
   identifier as part of its creation in 'uid', activate it,
   revoke it and then you can destroy it as below */

okvDestroy(env, &uid[0]);

if (okvErrGetNum(env))
{
   printf("Error while destroying the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.

- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.7 okvGetAttributeList

`okvGetAttributeList` implements KMIP get attribute list operation.

Catetory

KMIP API

Purpose

`okvGetAttributeList` implements KMIP get attribute list operation. It retrieves the names of the regular and custom attributes of a KMIP object specified by the unique identifier.

`attr_names_count` specifies how many `attr_names` can be accommodated in `attr_names` and `okvGetAttributeList()` will only copy that many attribute names. If there are more attributes then `attr_names_count` will be modified to store the actual number of attributes in the returned request.

Syntax

```
OKVErrNo okvGetAttributeList(OKVEnv *env, oratext *uid,
                             ub4 *attr_names_count, oratext **attr_names);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault Interface environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).
`attr_names_count`	`IN/OUT`	Count of names of the attributes names retrieved.
`attr_names`	`OUT`	Names of the attributes names retrieved.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The list retrieved is unordered.

Example

```
/* Create the KMIP object, say Key for example and get its unique identifier
   as part of its creation in 'uid'. Add attributes like say name, contact info
   and then names of these attributes can be retrieved as below */
…
ub4       attr_list_count = 0;
oratext **attr_names = (oratext **)NULL;

printf("\tGetting the count of attributes associated with the key\n");
okvGetAttributeList(env, uid, &attr_list_count, (oratext **)NULL);

if (okvErrGetNum(env))
{
   printf("Error while getting the attribute list count\n");
}

printf("\tGetting the attribute names associated with the key\n");
attr_names = (oratext **)calloc(attr_list_count, sizeof(oratext *));

for (i = 0; i < attr_list_count; i++)
{
   /* Allocate memory to hold attribute names */
   attr_names[i] = (oratext *)calloc(OKV_NAME_MAXLEN,
   sizeof(oratext));
}

okvGetAttributeList(env, uid, &attr_list_count, (oratext **)attr_names);

if (okvErrGetNum(env))
{
   printf("Error while getting the attribute list names\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.

- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.8 okvGetAttributes

`okvGetAttributes` implements KMIP get attribute operation.

Catetory

KMIP API

Purpose

`okvGetAttributes` implements KMIP Get Attribute operation. It retrieves the specified list of regular and custom attributes for a given KMIP object specified by the unique identifier.

Syntax

```
OKVErrNo okvGetAttributes(OKVEnv *env, oratext *uid,
                          ub4 attr_names_count,
                          oratext **attr_names,
                          OKVTTLV **attrs);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).
`attr_names_count`	`IN`	Count of names of the attributes retrieved.
`attr_names`	`IN`	Names of the attributes to be retrieved.
`attrs`	`OUT`	Attributes retrieved.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The list retrieved is unordered.

Example

```
/* Create the KMIP object, say Key for example and get its unique
   identifier as part of its creation in 'uid'. Add contact information
   Name, Cryptographic Alogirthm, Cryptographic Length attributes to the
   created key and then all of these attributes can be retrieved as below */
...
OKVTTLV *attrs = (OKVTTLV *) NULL;
oratext *attr_name_list[3];

attr_name_list[0] = okvGetTextForAttributeNum(OKVAttrName);
attr_name_list[1] = okvGetTextForAttributeNum(OKVAttrCryptoAlg);
attr_name_list[2] = okvGetTextForAttributeNum(OKVAttrCryptoLen);

okvGetAttributes(env, uid, 3, (oratext **) attr_name_list, &attrs);

if (okvErrGetNum(env))
{
```

```
   printf("Error while getting the attributes of the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.

- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.9 okvGetKey

`okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.

Catetory

KMIP API

Purpose

`okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.

For the specified unique identifier of the symmetric key, the key length, key algorithm, the actual key bytes in the endpoint program supplied buffer and the actual length of the key bytes are returned.

If the length of the key buffer supplied is smaller than the length of the actual key retrieved from the Oracle Key Vault Server, then the `keyl` is populated with the actual key length but key is set to `NULL`.

Syntax

```
OKVErrNo okvGetKey(OKVEnv *env, oratext *uid,
                   ub4 *key_alg, ub4 *key_len,
                   ub1 *key, ub4 *keyl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier of the key (can be `NULL` for batching).
`key_alg`	`OUT`	Symmetric key algorithm.
`key_len`	`OUT`	Key length of the symmetric key provided at the time of creation.
`key`	`OUT`	Symmetric key.
`keyl`	`OUT`	Length of the symmetric key.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create a symmetric Key for example and get its unique
   identifier as part of its creation in 'uid' */
...
ub4 type = 0, key_len = 0, keyl = 0;
ub1 key[100];
memset(key, 0, sizeof(key));
keyl = sizeof(key);
printf("\tGetting the key\n");
okvGetKey(env, &uid[0], &type, &key_len, &key[0], &keyl);

if (okvErrGetNum(env))
{
   printf("Error while getting the key\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.

- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.10 okvGetOpaqueData

`okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.

Catetory

KMIP API

Purpose

`okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.

For the specified unique identifier of the opaque data, the opaque data type, the opaque data length and the opaque data bytes in the endpoint program supplied buffer is returned.

If the length of the opaque data buffer supplied is smaller than the length of the actual opaque data retrieved from the Oracle Key Vault server, then `opaque_datal` is populated with the actual opaque data length but `opaque_data` is set to `NULL`.

Syntax

```
OKVErrNo okvGetOpaqueData(OKVEnv *env, oratext *uid,
                          ub4 *opaque_data_type,
                          ub1 *opaque_data, ub4 *opaque_datal);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Parameter	IN/OUT	Description
uid	IN	Unique identifier of the opaque object (can be NULL for batching).
opaque_data_type	OUT	Type of opaque object.
opaque_data	OUT	Opaque object.
opaque_datal	OUT	Length of opaque object.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create an opaque data for example and get its unique
   identifier as part of its creation in 'uid' */
...
ub1     *data;
ub4      data_len = 128001;
ub4      opaque_data_type = 0;
data = (ub1 *)malloc(data_len*sizeof(ub1));

okvGetOpaqueData(env, &uid[0], &opaque_data_type, &data[0], &data_len);

if (okvErrGetNum(env))
{
   printf("Error while getting the opaque data\n");
}
```

Related Topics

- okvActivate
  okvActivate implements the KMIP Activate operation.
- okvAddAttribute
  okvAddAttribute implements the KMIP add attribute operation.
- okvCreateKey
  okvCreateKey implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  okvDeleteAttribute implements the KMIP delete attribute operation.
- okvDestroy
  okvDestroy implements the KMIP destroy operation.

- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.11 okvGetSecretData

`okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.

Catetory

KMIP API

Purpose

`okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.

For the specified unique identifier of the secret data, the secret data type, the secret data length and the secret data bytes in the endpoint program supplied buffer is returned.

If the length of the secret data buffer supplied is smaller than the length of the actual secret data retrieved from the Oracle Key Vault server, then the `secret_datal` is populated with the actual secret data length but `secret_data` is set to `NULL`.

Syntax

```
OKVErrNo okvGetSecretData(OKVEnv *env, oratext *uid,
                          ub4 *secret_data_type,
                          ub1 *secret_data, ub4 *secret_datal);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier of the secret data (can be `NULL` for batching).
`secret_data_type`	`OUT`	Type of secret data.
`secret_data`	`OUT`	Secret data.
`secret_datal`	`OUT`	Length of secret data.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create a secret data for example and get its unique
   identifier as part of its creation in 'uid' */
...
ub1 data[100];
ub4 data_len = sizeof(data);
ub4 secret_data_type;

/* Get Secret Data associated with uid */
printf("\tGetting the secret data\n");

okvGetSecretData(env, &uid[0], &secret_data_type, &data[0], &data_len);

if (okvErrGetNum(env))
{
```

```
    printf("Error while getting the secret data\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.

- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.12 okvGetTemplate

`okvGetTemplate` implements the KMIP Get operation for the KMIP template object.

Catetory

KMIP API

Purpose

`okvGetTemplate` implements the KMIP Get operation for the KMIP template object.

The template is a list of attributes, which can be interpreted using the Oracle Key Vault utility or Oracle Key Vault KMIP extension functions. For the specified unique identifier of the template, the attributes of the template object are returned.

Syntax

```
OKVErrNo okvGetTemplate(OKVEnv *env, oratext *uid,
                        OKVTTLV **attrs_template);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier of the template (can be NULL for Batching).
`attrs_template`	`OUT`	Attributes of the template.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

This API will throw a general failure error while retrieving the template with process start date and protect stop date in the case if we register the template with these attributes.

Example

```
/* Create a template for example and get its unique
   identifier as part of its creation in 'uid' */
...
OKVTTLV *template = (OKVTTLV *) NULL;
```

```
okvGetTemplate(env, uid, &template);

if (okvErrGetNum(env))
{
   printf("Error while getting the template\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.

- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.13 okvLocate

`okvLocate` implements the KMIP locate operation.

Catetory

KMIP API

Purpose

`okvLocate` implements the KMIP locate operation.

The locate operation will look up all the objects in Oracle Key Vault that match the attributes specified in the `locate_attrs`.

`uid_cnt` should indicate the actual number of unique identifiers strings.

`uids` are the UIDS of the objects that match the attributes specified in `locate_attrs`.

Syntax

```
OKVErrNo okvLocate(OKVEnv *env,
                   ub4 uid_max, ub4 storage_status, OKVTTLV *locate_attrs,
                   ub4 *uid_cnt, oratext **uids);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid_max`	`IN`	Maximum number of unique identifiers expected.
`storage_status`	`IN`	Look for archived or online objects.
`locate_attrs`	`IN`	Attributes that define the locate search.
`uid_cnt`	`OUT`	Number of unique identifiers returned by the server.
`uids`	`OUT`	Unique identifiers that match the locate attributes and storage status criteria.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The caller has to allocate sufficient memory for the unique identifiers `uids` returned by this function. The only exception is irrespective of what the `uid_max` value is, all the UIDs are returned by the Oracle Key Vault server that match the attributes specified in `locate_attrs`.

Example

```
/* Set up the environment handle 'env' and also the memory and connection
   management as shown in previous sections. Create a key with a name
   attribute value "MyNewKey" for example and get its unique
   identifier as part of its creation in 'uid'. Locate operation can be done as
   shown below. */
...
ub4       locate_count = 0;
ub4       uid_max = 1;
OKVTTLV  *loc_attrs = (OKVTTLV *)NULL;
OKVTTLV  *attr_in = (OKVTTLV *)NULL;
oratext **locate_uids = (oratext **) malloc(uid_max * sizeof(oratext *));
ub4       itr = 0;

for (itr = 0; itr < uid_max; itr++)
{
   locate_uids[itr] = (oratext *) malloc(OKV_UNIQUE_ID_MAXLEN * sizeof(oratext));
}

loc_attrs = okvEnvGetOpRequestObj(env);
attr_in = okvAddAttributeObject(env, loc_attrs, OKVAttrName, (ub4) 0);

/* Add name attribute value to name attribute object */
okvAttrAddName(env, attr_in, (oratext *) "MyNewKey", strlen("MyNewKey"), 1);
okvLocate(env, uid_max, 1, loc_attrs, &locate_count, locate_uids);

if (okvErrGetNum(env))
{
   printf("Error while Locating the Key: MyNewKey\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.

- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.14 okvModifyAttribute

`okvModifyAttribute` implements the KMIP modify attribute operation.

Catetory

KMIP API

Purpose

`okvModifyAttribute` implements the KMIP modify attribute operation. It modifies an attribute of the KMIP Object specified by the unique identifier.

Syntax

```
OKVErrNo okvModifyAttribute(OKVEnv *env, oratext *uid,
                            OKVTTLV **attr);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier (can be `NULL` for batching).
`attr`	`IN`	Attribute object to be modified by the KMIP object.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create the KMIP object, say Key for example and get its unique identifier
   as part of its creation in 'uid', we can add an attribute and modify it
   as shown shown below */

OKVTTLV *templ = (OKVTTLV *) NULL;
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_temp = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *) NULL;
oratext *contact = (oratext *)"9123456789";
ub4      contactl = strlen((char *) contact);
oratext *new_contact = (oratext *)"abc.xyz@com";
ub4 new_contactl = strlen((char *) new_contact);
templ = okvEnvGetOpRequestObj(env);
attr_temp = okvAddAttributeObject(env, templ, OKVAttrContactInfo, (ub4) 0);

okvAttrAddContactInfo(env, attr_temp, contact, contactl);
okvAddAttribute(env, &uid[0], &attr_temp);

/* Now let's modify the contact info attribute */
req = okvEnvGetOpRequestObj(env);
attr_in = okvAddAttributeObject(env, req, OKVAttrContactInfo, (ub4) 0);
okvAttrAddContactInfo(env, attr_in, new_contact, new_contactl);
okvModifyAttribute(env, &uid[0], &attr_in);
```

```
if (okvErrGetNum(env))
{
   printf("Error while modifying the contact info attribute of the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.

- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.15 okvQueryCapability

`okvQueryCapability` implements the KMIP query operation.

Catetory

KMIP API

Purpose

`okvQueryCapability` implements the KMIP query operation.

KMIP query operation returns the Oracle Key Vault server supported items such as KMIP operations, objects, server information. Items can also be retrieved one at a time. Query function specifies which items should be returned. The supported values of query functions are

- `OKVDEF_QUERY_OPERATIONS`
- `OKVDEF_QUERY_OBJECTS`
- `OKVDEF_QUERY_SERVER_INFO`

The count of the item not requested in the query function will be zero. If server information is not requested then `server_information` can be `NULL`.

Syntax

```
OKVErrNo okvQueryCapability(OKVEnv *env,
                            ub4  query_function_cnt, ub4 *query_function,
                            ub4 *operation_cnt, OKVOpsNo *operation,
                            ub4 *object_type_cnt, OKVObjNo *object_type,
                            OKVServerInformation *server_information);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`query_function_cnt`	`IN`	Count of KMIP functions requested.
`query_function`	`IN`	KMIP functions requested.
`operation_cnt`	`OUT`	Count of KMIP operations supported.
`operation`	`OUT`	KMIP operations supported.
`object_type_cnt`	`OUT`	Count of KMIP managed objects supported.

Parameter	IN/OUT	Description
object_type	OUT	KMIP managed objects supported.
server_information	OUT	Oracle Key Vault specific information.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. */
...
OKVServerInformation s;
ub4 query_function[1];
ub4 operation_cnt = 30;
OKVOpsNo operation[30];
ub4 object_type_cnt = 30;
OKVObjNo object_type[30];
query_function[0] = 3;

okvQueryCapability(env,1, query_function, &operation_cnt, operation,
                   &object_type_cnt, object_type, &s);

if (okvErrGetNum(env))
{
   printf("Error while executing okvQueryCapability");
}
```

Related Topics

- okvActivate
  okvActivate implements the KMIP Activate operation.
- okvAddAttribute
  okvAddAttribute implements the KMIP add attribute operation.
- okvCreateKey
  okvCreateKey implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  okvDeleteAttribute implements the KMIP delete attribute operation.
- okvDestroy
  okvDestroy implements the KMIP destroy operation.

- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.16 okvRegKey

`okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.

Catetory

KMIP API

Purpose

`okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.

The unique identifier of the symmetric key object created by the Oracle Key Vault server is returned as `ouid`.

The maximum length of the unique ID for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

The new key created can take in additional attributes specified with the help of attributes or also with the template names. The attributes, and also the template names if specified, will be the template attribute added to KMIP register operation.

Syntax

```
OKVErrNo okvRegKey(OKVEnv *env,
                   OKVType key_alg, ub4 key_len, ub1 *key, ub4 keyl,
                   ub4 mask, OKVTTLV *template_names_attrs,
                   oratext *wallet_name, ub4 wallet_namel,
                   oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`key_alg`	`IN`	Symmetric key algorithm.
`key_len`	`IN`	Key length of the symmetric key.
`key`	`IN`	Symmetric key.
`keyl`	`IN`	Symmetric key length.
`mask`	`IN`	Cryptographic mask usage of the symmetric key.
`template_names_attrs`	`IN`	Template names or attributes that will form the template-attribute.
`wallet_name`	`IN`	Wallet name value.
`wallet_namel`	`IN`	Length of the wallet name value.
`ouid`	`OUT`	Unique identifier of the generated key.
`ouidl`	`OUT`	Unique identifier length of the generated key.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

It is recommended to add a KMIP name attribute to symmetric key object to help identify it in future.

The caller has to allocate the memory for the unique identifier returned by this function. The length of allocated memory is passed in via the length of the unique identifier argument `ouidl`. Sufficient memory to store the returned unique identifier should be allocated.

A symmetric key of the specified length for a specified algorithm and use (cryptographic usage mask) is registered with Oracle Key Vault server in wallet

`wallet_name` if specified else it is registered in the default wallet if present with the endpoint.

Example

```
/* Tag for Cryptographic Algortihm AES */
OKVType   algo = CRYPTO_ALG_AES;

/* Key length 128, because AES keys are 128, 192 or 256 bits in length*/
ub4       key_len = 128;
ub4       usage_mask = CRYPTO_MASK_ENCRYPT | CRYPTO_MASK_DECRYPT;

/* Key */
ub1       key[] = "770A8A65DA156D24";

/* Length of symmetric key */
ub4       keyl = strlen((const char *)key);
oratext   uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4       uidl = sizeof(uid);
OKVTTLV  *template = (OKVTTLV *)NULL;
OKVTTLV  *attr_in = (OKVTTLV *)NULL;

/* Set up the environment handle 'env' and also the memory
   and connection management as shown in previous sections */
memset(uid, 0, uidl);
template = okvEnvGetOpRequestObj(env);

/* Add name attribute object to request handle */
attr_in = okvAddAttributeObject(env, template, OKVAttrName, (ub4) 0);

/* Add name attribute value to name attribute object */
okvAttrAddName(env, attr_in, (oratext *)"My Key for Register operation",
               strlen("My Key for Register operation"), 1);
okvRegKey(env, algo, key_len, &key[0], keyl, usage_mask, template,
          (oratext *)NULL, (ub4)0, uid, &uidl);

if (okvErrGetNum(env))
{
   printf("Error while registering the key\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.

- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.17 okvRegOpaqueData

`okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.

Catetory

KMIP API

Purpose

`okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.

The opaque data is a byte string. A text file, text string, binary file, or key can also be uploaded as an opaque data.

The unique identifier of the opaque data object created in the Oracle Key Vault server for the registered opaque data is returned as `ouid`.

The maximum length of the unique ID for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

The new opaque data created can take in additional attributes specified with the help of attributes or also with the template names. The attributes, and also the template names if specified, will be the template attribute added to KMIP Register operation.

Syntax

```
OKVErrNo okvRegOpaqueData(OKVEnv *env,
                          ub4 opaque_data_type, ub1 *opaque_data,
                          ub4 opaque_datal,
                          OKVTTLV *template_names_attrs,
                          oratext *wallet_name, ub4 wallet_namel,
                          oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`opaque_data_type`	`IN`	Type of opaque object being registered.
`opaque_data`	`IN`	Opaque object being registered.
`opaque_data1`	`IN`	Length of opaque object being registered
`template_names_attrs`	`IN`	Template names or attributes that will form the template-attribute.
`wallet_name`	`IN`	Wallet name value.
`wallet_namel`	`IN`	Length of the wallet name value.
`ouid`	`OUT`	Unique identifier of the registered opaque object.
`ouid1`	`OUT`	Unique identifier length of the registered opaque object.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

It is recommended to add a KMIP Name attribute to opaque data object to help identify it in future.

The caller has to allocate the memory for the unique identifier returned by this function. The length of allocated memory is passed in via the length of the unique identifier argument `ouidl`. Sufficient memory to store the returned unique identifier should be allocated.

An opaque data of a specific type is registered with Oracle Key Vault server in wallet `wallet_name` if specified else it is registered in the default wallet if present with the endpoint. The size of the opaque data is limited by the maximum size of the object handled by the Oracle Key Vault server.

An exception is that the attributes lease time, deactivation date, destroy date, compromise occurrence date, compromise date, and revocation reason are not supported for opaque objects.

Example

```
oratext *opaque_data = (oratext *)"MyNewData";
ub4 opaque_datal = strlen("MyNewData");
oratext  uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4      uidl = sizeof(uid);

/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections */
memset(uid, 0, uidl);

/* Register Opaque Data */
printf("\tRegistering opaque data\n");

okvRegOpaqueData(env, OKVDEF_TAG_OPAQUE_DATA_TYPE,
                 opaque_data, opaque_datal,
                 (OKVTTLV *)NULL, (oratext *)NULL, (ub4)0,
                 &uid[0], uidl);

if (okvErrGetNum(env))
{
   printf("Error while registering the opaque data\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.

- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.18 okvRegSecretData

`okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.

Catetory

KMIP API

Purpose

`okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.

Secret data is usually a password or a string.

The unique identifier of the secret data object created in the Oracle Key Vault server for the registered secret data is returned as `ouid`.

The maximum length of the unique ID for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

The new secret data created can take in additional attributes specified with the help of attributes or also with the template names. The attributes, and also the template names if specified, will be the template attribute added to KMIP register operation.

Syntax

```
OKVErrNo okvRegSecretData(OKVEnv *env,
                          ub4 secret_data_type, ub1 *secret_data,
                          ub4 secret_datal, ub4 mask,
                          OKVTTLV *template_names_attrs,
                          oratext *wallet_name, ub4 wallet_namel,
                          oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
secret_data_type	IN	Type of secret data being registered.
secret_data	IN	Secret data being registered.
secret_datal	IN	Length of secret data being registered.
mask	IN	Cryptographic mask usage of the symmetric key.
template_names_attrs	IN	Template names or attributes that will form the template-attribute.
wallet_name	IN	Wallet name value.
wallet_namel	IN	Length of the wallet name value.
ouid	OUT	Unique identifier of the registered secret data.
ouidl	OUT	Unique identifier length of the registered secret data.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

It is recommended to add a KMIP name attribute to secret data object to help identify it in future.

The caller has to allocate the memory for the unique identifier returned by this function. The length of allocated memory is passed in via the length of the unique identifier

(ouidl) argument. Sufficient memory to store the returned unique identifier should be allocated.

A secret data of a specific type is registered with Oracle Key Vault server in wallet (wallet_name) if specified else it is registered in the default wallet if present with the endpoint.

The size of the secret data is limited by the maximum size of the object handled by the Oracle Key Vault server.

Example

```
/* Parameters for registering secret data */
/* Tag for secret data type */
OKVType   type = OKVDEF_SECRET_DATA_TYPE_PASSWORD;
ub4       usage_mask = CRYPTO_MASK_ENCRYPT | CRYPTO_MASK_DECRYPT;
oratext  *secret = (oratext *)"MyNewSecret";
ub4       secretl = strlen("MyNewSecret");
oratext   uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4       uidl = sizeof(uid);

/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections */
memset(uid, 0, uidl);

/* Register Secret data */
printf("\tRegistering the secret data\n");

okvRegSecretData(env, type, secret, secretl,
                 usage_mask, (OKVTTLV *)NULL,
                 (oratext *)NULL, (ub4)0,
                 &uid[0], uidl);

if (okvErrGetNum(env))
{
   printf("Error while registering the secret data\n");
}
```

Related Topics

- okvActivate
  okvActivate implements the KMIP Activate operation.
- okvAddAttribute
  okvAddAttribute implements the KMIP add attribute operation.
- okvCreateKey
  okvCreateKey implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  okvDeleteAttribute implements the KMIP delete attribute operation.
- okvDestroy
  okvDestroy implements the KMIP destroy operation.
- okvGetAttributeList
  okvGetAttributeList implements KMIP get attribute list operation.
- okvGetAttributes
  okvGetAttributes implements KMIP get attribute operation.

- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.19 okvRegTemplate

`okvRegTemplate` implements the KMIP Register operation for the KMIP template object.

Catetory

KMIP API

Purpose

`okvRegTemplate` implements the KMIP Register operation for the KMIP template object (not to be confused with KMIP template-attribute object).

A template object is a collection of attributes.

The unique identifier of the template object created with the specified attributes in Oracle Key Vault for the registered template is returned as `ouid`.

The maximum length of the unique identifier for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

Syntax

```
OKVErrNo okvRegTemplate(OKVEnv *env,
                        OKVTTLV *attrs_template,
                        oratext *wallet_name, ub4 wallet_namel,
                        oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`attrs_template`	`IN`	Attributes of the template.
`wallet_name`	`IN`	Wallet name value.
`wallet_namel`	`IN`	Length of the wallet name value.
`ouid`	`OUT`	Unique identifier of the registered template.
`ouid1`	`OUT`	Unique identifier length of the registered template.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The caller has to allocate the memory for the unique identifier returned by this function. The length of allocated memory is passed in via the length of the unique identifier (`ouidl`) argument. Sufficient memory to store the returned unique identifier should be allocated.

The template will be registered with the wallet specified in `wallet_name` argument else it will be registered with the default wallet associated with the endpoint in case the default wallet exists.

Example

```
oratext  uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4      uidl = sizeof(uid);

/* Set up the environment handle 'env' also the memory and connection
   management as shown in previous sections */
memset(uid, 0, uidl);
okvRegTemplate(env, (OKVTTLV *)NULL, (oratext *)NULL, (ub4)0, &uid[0], &uidl);

if (okvErrGetNum(env))
{
   printf("Error while registering the Template\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.20 okvRekey

`okvRekey` implements the KMIP rekey operation.

Catetory

KMIP API

Purpose

`okvRekey` implements the KMIP rekey operation.

The rekey operation requires the unique identifier of the symmetric key that needs to be re-keyed. Most of the attributes are carried over from the old key. Some attributes are modified per KMIP defined rules.

The unique identifier of the new symmetric key object generated by the Oracle Key Vault server is returned as `ouid`.

The maximum length of the unique ID for Oracle Key Vault SDK is defined by `OKV_UNIQUE_ID_MAXLEN`.

Syntax

```
OKVErrNo okvRekey(OKVEnv *env, oratext *uid,
                  ub4 offset, OKVTTLV *template_names_attrs,
                  oratext *ouid, ub4 *ouidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`uid`	`IN`	Unique identifier of the key being rekeyed (can be NULL for Batching).
`offset`	`IN`	Time interval between initialization (creation) date and activation date.
`template_names_attrs`	`IN`	Template names or attributes that will form the template-attribute for rekey.
`ouid`	`OUT`	Unique identifier of the new generated symmetric key.
`ouidl`	`OUT`	Unique identifier length of the newly generated symmetric key.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The caller has to allocate the memory for the unique identifier of the new symmetric key returned by this function. The length of allocated memory is passed in via the length of the unique identifier (`ouidl`) argument.

The API has the below exceptions when used with Oracle Key Vault server:

- The server doesn't impose any restrictions on the number of times a key can be rekeyed. For example, if a key K1 is rekeyed and a new key K2 is created, the server allows the key K1 to be rekeyed again and key K3 would be created.
- If a template is passed for rekey, the template attributes are not added for the new key. As a workaround, this can be added post rekey using `okvAddAttribute`.

Example

```
/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. Create
   a key for example and get its unique identifier as part of
   its creation in 'uid'. Rekey can done as shown below */
…
oratext  rekey_uid[OKV_UNIQUE_ID_MAXLEN + 1];
ub4      rekey_uidl = sizeof(rekey_uid);

memset(rekey_uid, 0, rekey_uidl);
okvRekey(env, uid, (ub4)0, (OKVTTLV *)NULL, &rekey_uid[0], &rekey_uidl);

if (okvErrGetNum(env))
{
   printf("Error while Re-Keying in the Key\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.
- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.

- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRevoke
  `okvRevoke` implements the KMIP revoke operation.

10.1.21 okvRevoke

`okvRevoke` implements the KMIP revoke operation.

Catetory

KMIP API

Purpose

`okvRevoke` implements the KMIP revoke operation. It revokes the KMIP object specified by the unique identifier with a revocation reason and the date when the object was compromised.

Syntax

```
OKVErrNo okvRevoke(OKVEnv *env, oratext *uid,
                   ub4 revocation_reason,
                   oratext *revocation_msg,
                   ub8 comp_occurrence_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
uid	IN	Unique identifier (can be NULL for batching).
revocation_reason	IN	Revocation reason for revoking the KMIP object.
revocation_msg	IN	Revocation message for revoking the KMIP object.
comp_occurrence_date	IN	Date when the KMIP object compromise occurred.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Create the KMIP object, say Key for example and get its unique identifier
   as part of its creation in 'uid', activate it and then revoke it as below */

okvRevoke(env, &uid[0], (ub4)1, (oratext *)"Retiring the key",
          (ub8)time((time_t *)NULL));

if (okvErrGetNum(env))
{
   printf("Error while revoking the object\n");
}
```

Related Topics

- okvActivate
  `okvActivate` implements the KMIP Activate operation.
- okvAddAttribute
  `okvAddAttribute` implements the KMIP add attribute operation.

- okvCreateKey
  `okvCreateKey` implements the KMIP Create operation for the KMIP symmetric key object.
- okvDeleteAttribute
  `okvDeleteAttribute` implements the KMIP delete attribute operation.
- okvDestroy
  `okvDestroy` implements the KMIP destroy operation.
- okvGetAttributeList
  `okvGetAttributeList` implements KMIP get attribute list operation.
- okvGetAttributes
  `okvGetAttributes` implements KMIP get attribute operation.
- okvGetKey
  `okvGetKey` implements the KMIP get operation for the KMIP symmetric key object.
- okvGetOpaqueData
  `okvGetOpaqueData` implements the KMIP get operation for the KMIP opaque data object.
- okvGetSecretData
  `okvGetSecretData` implements the KMIP get operation for the KMIP secret data object.
- okvGetTemplate
  `okvGetTemplate` implements the KMIP Get operation for the KMIP template object.
- okvLocate
  `okvLocate` implements the KMIP locate operation.
- okvModifyAttribute
  `okvModifyAttribute` implements the KMIP modify attribute operation.
- okvQueryCapability
  `okvQueryCapability` implements the KMIP query operation.
- okvRegKey
  `okvRegKey` implements the KMIP register operation for the KMIP symmetric key object.
- okvRegOpaqueData
  `okvRegOpaqueData` implements the KMIP register operation for the KMIP opaque data object.
- okvRegSecretData
  `okvRegSecretData` implements the KMIP Register operation for the KMIP secret data object.
- okvRegTemplate
  `okvRegTemplate` implements the KMIP Register operation for the KMIP template object.
- okvRekey
  `okvRekey` implements the KMIP rekey operation.

10.2 Oracle Key Vault Client SDK Batch APIs

This section describes the interfaces for the Oracle Key Vault KMIP batch functions.

- okvBatchCreate
  `okvBatchCreate` will mark the start of Oracle Key Vault batching.
- okvBatchExecute
  `okvBatchExecute` will execute the batched Oracle Key Vault functions.
- okvBatchFree
  `okvBatchFree` will mark the end of Oracle Key Vault batching.
- okvGetBatchOperationCount
  `okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.
- okvGetBatchOperationName
  `okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

10.2.1 okvBatchCreate

`okvBatchCreate` will mark the start of Oracle Key Vault batching.

Catetory

KMIP batch API

Purpose

`okvBatchCreate` will mark the start of Oracle Key Vault batching. All Oracle Key Vault functions after this command will be batched.

Syntax

```
OKVErrNo okvBatchCreate(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. */
```

```
printf("Start preparing batch operations\n");
okvBatchCreate(env);

if (okvErrGetNum(env))
{
   printf("Error while initiating the Batch");
}
...
/* All Oracle Key Vault functions will be batched */
```

Related Topics

- okvBatchExecute
  `okvBatchExecute` will execute the batched Oracle Key Vault functions.
- okvBatchFree
  `okvBatchFree` will mark the end of Oracle Key Vault batching.
- okvGetBatchOperationCount
  `okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.
- okvGetBatchOperationName
  `okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

10.2.2 okvBatchExecute

`okvBatchExecute` will execute the batched Oracle Key Vault functions.

Catetory

KMIP batch API

Purpose

`okvBatchExecute` will execute the batched Oracle Key Vault functions. The batched Oracle Key Vault functions are functions between `okvBatchCreate` and `okvBatchExecute`. If there are errors, then the errors for all the batched Oracle Key Vault functions should be checked to verify which operations failed and which were successful.

Syntax

```
OKVErrNo okvBatchExecute(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

To check for individual operations in the batch after its execution, these APIs can be made use of: `okvErrGetNumForBatch`, `okvErrGetDepthForBatch`, and `okvErrGetNumAtDepthForBatch`. The errors for individual batch operations can be checked only before freeing the batch context, that is, before calling `okvBatchFree`.

Example

```
/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. */
printf("Start preparing batch operations\n");
okvBatchCreate(env);
...
/* All Oracle Key Vault functions will be batched */
...
printf("Executing batch operation\n");
okvBatchExecute(env);

if (okvErrGetNum(env))
{
   printf("Error while executing the batch");
}
```

Related Topics

- okvBatchCreate
  `okvBatchCreate` will mark the start of Oracle Key Vault batching.
- okvBatchFree
  `okvBatchFree` will mark the end of Oracle Key Vault batching.
- okvErrGetDepthForBatch
  `okvErrGetDepthForBatch` is used to return the depth of the error stack for the provided batch job number otherwise 0 is returned.
- okvErrGetNumAtDepthForBatch
  `okvErrGetNumAtDepthForBatch` returns the error number at the specified depth of the error stack for the provided batch job number.
- okvErrGetNumForBatch
  `okvErrGetNumForBatch` returns the error number on top of the error stack for the provided batch job number.
- okvGetBatchOperationCount
  `okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.

- okvGetBatchOperationName
  `okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

10.2.3 okvBatchFree

`okvBatchFree` will mark the end of Oracle Key Vault batching.

Catetory

KMIP batch API

Purpose

`okvBatchFree` will mark the end of Oracle Key Vault batching. Essentially the memory allocated for complex arguments will be released. Subsequent Oracle Key Vault functions will not be batched unless `okvBatchCreate` is called again.

Syntax

```
OKVErrNo okvBatchFree(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The errors for individual batch operations can be checked only before freeing the batch context, that is, before calling `okvBatchFree`.

Example

```
/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. */
printf("Start preparing batch operations\n");
okvBatchCreate(env);
...
/* All Oracle Key Vault functions will be batched */
...
printf("Executing batch operation\n");
okvBatchExecute(env);
...
/* Check for Individual Batch Operation errors */
...
```

```
printf("Ending batch operations\n\n");
okvBatchFree(env);

if (okvErrGetNum(env))
{
   printf("Error while ending the batch");
}
```

Related Topics

- okvBatchCreate
  `okvBatchCreate` will mark the start of Oracle Key Vault batching.
- okvBatchExecute
  `okvBatchExecute` will execute the batched Oracle Key Vault functions.
- okvGetBatchOperationCount
  `okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.
- okvGetBatchOperationName
  `okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

10.2.4 okvGetBatchOperationCount

`okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.

Catetory

KMIP batch API

Purpose

`okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.

Syntax

```
ub4 okvGetBatchOperationCount(OKVEnv *env);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.

Return Values

Return Value	Description
ub4	Count of the batched operations. **Success:** A valid positive non-zero number is returned. **Failure:** Zero.

Comments

This function should be called before calling `okvBatchFree` else it will return zero.

Example

```
ub4 batch_cnt = 0;

/* Set up the environment handle 'env' and also the memory and
   connection management as shown in previous sections. */
printf("\t\tStart preparing batch operations\n");
okvBatchCreate(env);
...
/* All Oracle Key Vault functions will be Batched */
...
printf("\t\tExecuting batch operation\n");
okvBatchExecute(env);
...
/* Check for Individual Batch Operation errors */
...
batch_cnt = okvGetBatchOperationCount(env);
printf("Batch operations count is: %d\n", batch_cnt);
```

Related Topics

- okvBatchCreate
  `okvBatchCreate` will mark the start of Oracle Key Vault batching.
- okvBatchExecute
  `okvBatchExecute` will execute the batched Oracle Key Vault functions.
- okvBatchFree
  `okvBatchFree` will mark the end of Oracle Key Vault batching.
- okvGetBatchOperationName
  `okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

10.2.5 okvGetBatchOperationName

`okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

Catetory

KMIP batch API

Purpose

`okvGetBatchOperationName` returns the name of the respective batch job number that is passed with this function.

Syntax

```
oratext *okvGetBatchOperationName(OKVEnv *env, ub1 batchjobnum);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`batchjobnum`	`IN`	Batch job number.

Return Values

Return Value	Description
`oratext *`	Batch job operation name. **Success:** A pointer to batch job operation name is returned. **Failure:** `NULL` pointer is returned.

Comments

This function should be called before calling `okvBatchFree` else it will return a pointer to `NULL`.

`batchjobnum` is the number in which the operations are batched. For example, if the user wants to create a key, activate a key, revoke it, and then destroy it, all of these operations are batched in the same order, that is, `batchjobnum` for create a key is 1, `batchjobnum` for activate a key is 2, `batchjobnum` for revoke is 3, and `batchjobnum` for destroy is 4.

Example

```
ub4 batch_cnt = 0;
ub4 batch_job_num = 0;

/* Set up the environment handle 'env' also the memory and
   connection management as shown in previous sections. */
printf("\t\tStart preparing batch operations\n");
okvBatchCreate(env);
...
/* All Oracle Key Vault functions will be Batched */
...
printf("\t\tExecuting batch operation\n");
okvBatchExecute(env);
...
/* Check for Individual Batch Operation errors */
...
batch_cnt = okvGetBatchOperationCount(env);
printf("Batch Operations count is: %d\n", batch_cnt);

if (batch_cnt)
{
   for(batch_job_num=1; batch_job_num<=batch_cnt; batch_job_num++)
   {
      printf("\t\t\t%d: %s\n", batch_job_num, okvGetBatchOperationName(env,
batch_job_num));
   }
}
```

Related Topics

- okvBatchCreate
  `okvBatchCreate` will mark the start of Oracle Key Vault batching.
- okvBatchExecute
  `okvBatchExecute` will execute the batched Oracle Key Vault functions.
- okvBatchFree
  `okvBatchFree` will mark the end of Oracle Key Vault batching.

- okvGetBatchOperationCount
  `okvGetBatchOperationCount` gets the count of batched Oracle Key Vault operations.

11

Oracle Key Vault Client SDK KMIP Attributes and Custom Attributes APIs

This section describes the interfaces that help create and interpret both KMIP attributes and KMIP custom attributes.

- Oracle Key Vault KMIP Attributes APIs
  Oracle Key Vault provides a set of APIs to manage KMIP attributes.
- Oracle Key Vault KMIP Custom Attribute APIs
  The KMIP Custom Attribute APIs are listed in this section.

11.1 Oracle Key Vault KMIP Attributes APIs

Oracle Key Vault provides a set of APIs to manage KMIP attributes.

- About the Oracle Key Vault KMIP Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.
- Attribute Index and Element Index
  Element index can be used to retrieve child TTLV objects from the parent TTLV object and attribute index is the property of a KMIP Attribute.
- okvAddAttributeObject
  `okvAddAttributeObject` adds an attribute object to the parent `TTLV` object.
- okvAttrAddActivationDate
  `okvAttrAddActivationDate` adds the activation date attribute to OKVTTLV parent object.
- okvAttrAddCompromiseDate
  `okvAttrAddCompromiseDate` adds a compromise date attribute to OKVTTLV parent object.
- okvAttrAddCompromiseOccurrenceDate
  `okvAttrAddCompromiseOccurrenceDate` adds a compromise occurrence date attribute to OKVTTLV parent object.
- okvAttrAddContactInfo
  `okvAttrAddContactInfo` adds a contact information attribute to the OKVTTLV parent object.
- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.

- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrAddCryptoUsageMask
  `okvAttrAddCryptoUsageMask` adds a cryptographic usage mask attribute to the OKVTTLV parent object.
- okvAttrAddDeactivationDate
  `okvAttrAddDeactivationDate` adds a deactivation date attribute to the OKVTTLV parent object.
- okvAttrAddDestroyDate
  `okvAttrAddDestroyDate` adds a destroy date attribute to the OKVTTLV parent object.
- okvAttrAddDigest
  `okvAttrAddDigest` adds a digest attribute to the OKVTTLV object.
- okvAttrAddFresh
  `okvAttrAddFresh` adds a fresh attribute to the OKVTTLV parent object.
- okvAttrAddLeaseTime
  `okvAttrAddLeaseTime` adds a lease time attribute to the OKVTTLV parent object.
- okvAttrAddName
  `okvAttrAddName` adds a name attribute to the OKVTTLV parent object.
- okvAttrAddObjectGroup
  `okvAttrAddObjectGroup` adds an object group attribute to the OKVTTLV parent object.
- okvAttrAddObjectType
  `okvAttrAddObjectType` adds an object type attribute to the OKVTTLV parent object.
- okvAttrAddProcessStartDate
  `okvAttrAddProcessStartDate` adds a process start date attribute to the OKVTTLV parent object.
- okvAttrAddProtectStopDate
  `okvAttrAddProtectStopDate` adds a protect stop date attribute to the OKVTTLV parent object.
- okvAttrAddRevocationReason
  `okvAttrAddRevocationReason` adds the revocation reason attribute to the OKVTTLV parent object.
- okvAttrAddUniqueID
  `okvAttrAddUniqueID` adds a unique identifier attribute to the OKVTTLV parent object.
- okvAttrAddUsageLimits
  `okvAttrAddUsageLimits` adds a usage limits attribute to the OKVTTLV parent object.
- okvAttrGetActivationDate
  `okvAttrGetActivationDate` gets the activation date attribute.
- okvAttrGetArchiveDate
  `okvAttrGetArchiveDate` gets the archive date attribute.

- okvAttrGetCompromiseDate
  `okvAttrGetCompromiseDate` gets the compromise date attribute.
- okvAttrGetCompromiseOccurrenceDate
  `okvAttrGetCompromiseOccurrenceDate` gets the compromise occurrence date attribute.
- okvAttrGetContactInfo
  `okvAttrGetContactInfo` gets the contact information attribute.
- okvAttrGetContactInfoLen
  `okvAttrGetContactInfoLen` gets the length of the contact information of the contact information attribute.
- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.
- okvAttrGetCryptoUsageMask
  `okvAttrGetCryptoUsageMask` gets the cryptographic usage mask attribute.
- okvAttrGetDeactivationDate
  `okvAttrGetDeactivationDate` gets the deactivation date attribute.
- okvAttrGetDestroyDate
  `okvAttrGetDestroyDate` gets the destroy date attribute.
- okvAttrGetDigest
  `okvAttrGetDigest` gets the digest attribute found at or after element index `elem_index`.
- okvAttrGetDigestLen
  `okvAttrGetDigestLen` gets the length of the digest value of the digest attribute found at or after element index `elem_index`.
- okvAttrGetFresh
  `okvAttrGetFresh` gets the fresh attribute.
- okvAttrGetInitialDate
  `okvAttrGetInitialDate` gets the initial date attribute.
- okvAttrGetLastChangeDate
  `okvAttrGetLastChangeDate` gets the last change date attribute.
- okvAttrGetLeaseTime
  `okvAttrGetLeaseTime` gets the lease time attribute.
- okvAttrGetName
  `okvAttrGetName` gets the name value, attribute index, and name type of the name attribute found at or after element index `elem_index`.
- okvAttrGetNameValueLen
  `okvAttrGetNameValueLen` gets the length of the name value of the name attribute found at or after element index `elem_index`.

- okvAttrGetObjectGroup
  `okvAttrGetObjectGroup` gets the object group of the object group attribute found at or after element index `elem_index`.
- okvAttrGetObjectGroupLen
  `okvAttrGetObjectGroupLen` gets the length of the object group found at or after element index `elem_index`.
- okvAttrGetObjectType
  `okvAttrGetObjectType` gets the object type attribute.
- okvAttrGetProcessStartDate
  `okvAttrGetProcessStartDate` gets the process start date attribute.
- okvAttrGetProtectStopDate
  `okvAttrGetProtectStopDate` gets the protect stop date attribute.
- okvAttrGetRevocationReason
  `okvAttrGetRevocationReason` gets the revocation reason attribute.
- okvAttrGetRevocationReasonMessageLen
  `okvAttrGetRevocationReasonMessageLen` gets the length of the revocation message of the revocation reason attribute.
- okvAttrGetState
  `okvAttrGetState` gets the state attribute.
- okvAttrGetUniqueID
  `okvAttrGetUniqueID` gets the unique ID attribute.
- okvAttrGetUniqueIDLen
  `okvAttrGetUniqueIDLen` gets the length of the unique identifier attribute.
- okvAttrGetUsageLimits
  `okvAttrGetUsageLimits` gets the usage limits attribute.
- okvGetAttributeObject
  `okvGetAttributeObject` gets the attribute, its name, and index from the parent attribute found at element index `elem_index`.

11.1.1 About the Oracle Key Vault KMIP Attribute APIs

This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.

For the functions in this section, all of the functions with arguments `env` and `ttlv` have the following meaning:

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault Environment handle.
ttlv	IN	TTLV parent object.

For the functions in this section that return the TTLV object (`OKVTTLV *`), the return value has the following interpretation:

- On success, a valid TTLV object for the attribute is returned.
- On failure, a NULL pointer is returned.

For the functions in this section that return the length of the retrieved object, the return value has the following interpretation:

- On success, the length of the buffer is returned.
- On failure, zero (0) is returned.

For the functions in this section that return the Oracle Key Vault error number, the return value has the following interpretation:

- On success, `OKV_SUCCESS` is returned.
- On failure, a valid error number is returned for the error on top of the error stack.

Oracle Key Vault functions defined in this section add and retrieve attributes of a specific KMIP data type to and from the OKVTTLV object. The functions have the following construction in general:

- `okvAttrAdd`*`Attr`*: Adds the attribute to the OKVTTLV parent object.
- `okvAttrGet`*`Attr`*`Len`: Gets the value length of the attribute in OKVTTLV parent object for KMIP Attributes that don't have fixed length.
- `okvAttrGet`*`AttrElement`*`Len`: Gets the value length of the element of the attribute in the OKVTTLV parent object for STRUCTURE attributes that have child values with variable length. An example is the name value of the `name` attribute.
- `okvAttrGet`*`Attr`*: Gets the attribute in the OKVTTLV parent object.

11.1.2 Attribute Index and Element Index

Element index can be used to retrieve child TTLV objects from the parent TTLV object and attribute index is the property of a KMIP Attribute.

An OKVTTLV parent object can have a number of child OKVTTLV objects. The child objects usually have different KMIP tags. Some of the child objects like, multi-instance attributes can be repeated, so there can be a few child objects with the same tag.

The immediate child objects of an OKVTTLV object can be thought of as an indexed list. The child OKVTTLV objects can be retrieved one by one from the parent object by specifying the index. This index is the element index of the child TTLV object.

okvTTLVGetChild should be used to retrieve the child by specifying the element index.

In most cases however, the child being looked up is known. The child is usually identified by a tag. But what is not known is the index of the child TTLV object.

okvTTLVGetChildByTag is used to lookup the child with a given tag.

With this API we generally guess that the child is found at particular index. The API will look for the first occurrence of the child from this index onwards and will return the actual index where the child was found.

Usually we should start with index 0, to get the first occurrence of the child with a given tag. The returned element index will specify where the child was found.

The element index can be incremented by one and `okvTTLVGetChildByTag` can be called again to get the second occurrence of the child with the same tag.

The first occurrence of child with a given tag can also be done by using `okvTTLVGetFirstChildByTag`.

This behavior is true for any Oracle Key Vault function that takes element index as an argument.

As an example using the OKVTTLV Parent Object table below, calling `okvTTLVGetChildByTag("Name", &elem_index)` with `elem_index` set to 1 will return object with Ref# C, and set the `elem_index` to 2. Again calling `okvTTLVGetChildByTag("Object Group", &elem_index)` with `elem_index` set to 4 will return object with Ref# E, and set the `elem_index` to 4.

Table 11-1 OKVTTLV Parent Object

Ref #	Attribute Object	Element Index	Attribute Index
A	Name	0	1
B	Object Group	1	1
C	Name	2	3
D	Name	3	2
E	Object Group	4	2

Attribute index is the property of a KMIP Attribute. Oracle Key Vault returns the attribute index to the endpoint program when requested or the endpoint program can add one when modifying or adding an attribute. Oracle Key Vault does not do any special processing for the attribute index and does not use it for lookup operations.

11.1.3 okvAddAttributeObject

`okvAddAttributeObject` adds an attribute object to the parent `TTLV` object.

Catetory

KMIP attribute API

Purpose

`okvAddAttributeObject` adds an attribute object to the parent TTLV object. The actual attribute value can be added using the functions defined in this section.

Syntax

```
OKVTTLV* okvAddAttributeObject(OKVEnv *env, OKVTTLV *ttlv,
                               OKVAttrNo attrno, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault Environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`attrno`	`IN`	Attribute number of the attribute being added.
`attr_index`	IN	Attribute index of the attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with attribute object TTLV indicated by argument `attrno` added. **Failure:** `NULL` pointer.

Comments

The API can be used to add attribute TTLVs to the parent TTLV object. The below example shows how a contact info attribute can be added to the parent TTLV object.

Example

```
oratext *contact = (oratext *)"sample contact info";
ub4      contactl = strlen((char *) contact);
OKVTTLV *resultttlv;

resultttlv = okvAddAttributeObject(env, ttlv, OKVAttrContactInfo, (ub4) 0);
okvAttrAddContactInfo(env, resultttlv, contact, contactl);
```

Related Topics

- okvGetAttributeObject
  `okvGetAttributeObject` gets the attribute, its name, and index from the parent attribute found at element index `elem_index`.

11.1.4 okvAttrAddActivationDate

`okvAttrAddActivationDate` adds the activation date attribute to OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddActivationDate` adds the activation date attribute to OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddActivationDate(OKVEnv *env, OKVTTLV *ttlv,
                                  ub8 activation_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.

Parameter	IN/OUT	Description
ttlv	IN	Pointer to parent OKVTTLV object.
activation_date	IN	Activation date.

Return Values

Return Value	Description
OKVTTLV *	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with activation date attribute value added. **Failure:** NULL pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
...
okvAttrAddActivationDate(env, attr_in, (ub8) current_time + 10000);
```

Related Topics

- okvAttrGetActivationDate
  `okvAttrGetActivationDate` gets the activation date attribute.

11.1.5 okvAttrAddCompromiseDate

`okvAttrAddCompromiseDate` adds a compromise date attribute to OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddCompromiseDate` adds a compromise date attribute to OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCompromiseDate(OKVEnv *env, OKVTTLV *ttlv,
                                  ub8 comp_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to OKVTTLV object.

Parameter	IN/OUT	Description
comp_date	IN	Compromise date.

Return Values

Return Value	Description
OKVTTLV *	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with compromise date attribute value added. **Failure:** NULL pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
OKVTTLV *resultttlv;
resultttlv = okvAttrAddCompromiseDate(env, ttlv, (ub8) current_time + 24*3600);
```

Related Topics

- okvAttrGetCompromiseDate
  okvAttrGetCompromiseDate gets the compromise date attribute.

11.1.6 okvAttrAddCompromiseOccurrenceDate

okvAttrAddCompromiseOccurrenceDate adds a compromise occurrence date attribute to OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

okvAttrAddCompromiseOccurrenceDate adds a compromise occurrence date attribute to OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCompromiseOccurrenceDate(OKVEnv *env, OKVTTLV *ttlv,
                                            ub8 comp_occr_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to parent OKVTTLV object.
comp_occr_date	IN	Compromise occurrence date.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with compromise occurrence date attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
ub8 comp_date;
ctime(&comp_date);
resulttlv = okvAttrAddCompromiseOccurrenceDate(env, ttlv, comp_date);
```

Related Topics

- okvAttrGetCompromiseOccurrenceDate
  `okvAttrGetCompromiseOccurrenceDate` gets the compromise occurrence date attribute.

11.1.7 okvAttrAddContactInfo

`okvAttrAddContactInfo` adds a contact information attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddContactInfo` adds a contact information attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddContactInfo(OKVEnv *env, OKVTTLV *ttlv,
                               oratext *contact_info, ub4 contact_infol);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`contact_info`	`IN`	Contact information.
`contact_infol`	`IN`	Length of contact information.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with contact info attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resulttlv;
oratext *new_contact = (oratext *)"abc.xyz@com";
ub4 new_contactl = strlen((char *) new_contact);
resulttlv = okvAttrAddContactInfo(env, attr_in, new_contact, new_contactl);
```

Related Topics

- okvAttrGetContactInfoLen
  `okvAttrGetContactInfoLen` gets the length of the contact information of the contact information attribute.
- okvAttrGetContactInfo
  `okvAttrGetContactInfo` gets the contact information attribute.

11.1.8 okvAttrAddCryptoAlgo

`okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCryptoAlgo(OKVEnv *env, OKVTTLV *ttlv,
                              ub4 crypto_alg);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.

Parameter	IN/OUT	Description
crypto_alg	IN	Cryptographic algorithm.

Return Values

Return Value	Description
OKVTTLV *	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with cryptographic algorithm attribute value added. **Failure:** NULL pointer.

Comments

None.

Example

```
attr = okvAddAttributeObject(env, templ, OKVAttrCryptoAlg, 0);
okvAttrAddCryptoAlgo(env, attr, (ub4)CRYPTO_ALG_AES);
```

Related Topics

- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.
- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.
- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

11.1.9 okvAttrAddCryptoLen

`okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCryptoLen(OKVEnv *env, OKVTTLV *ttlv,
                             ub4 crypto_len);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`crypto_len`	`IN`	Cryptographic length.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with cryptographic length attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
attr = okvAddAttributeObject(env, templ, OKVAttrCryptoLen, 0);
okvAttrAddCryptoLen(env, attr, (ub4)128);
```

Related Topics

- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.
- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.
- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

11.1.10 okvAttrAddCryptoParams

`okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCryptoParams(OKVEnv *env, OKVTTLV *ttlv,
                                ub4 block_cipher_mode, ub4 padding_method,
                                ub4 hashing_algorithm, ub4 key_role_type);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent TTLV.
`block_cipher_mode`	`IN`	Block cipher mode.
`padding_method`	`IN`	Padding method.
`hashing_algorithm`	`IN`	Hashing algorithm.
`key_role_type`	`IN`	Key role type.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with cryptographic parameters added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
okvAttrAddCryptoParams(env, ttlv, BLK_CIPHER_ECB, PADDING_SSL3,
                       HASH_ALG_MD4, KEY_ROLE_CVK);
```

Related Topics

- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.

- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.
- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

11.1.11 okvAttrAddCryptoUsageMask

`okvAttrAddCryptoUsageMask` adds a cryptographic usage mask attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddCryptoUsageMask` adds a cryptographic usage mask attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddCryptoUsageMask(OKVEnv *env, OKVTTLV *ttlv,
                                   ub4 mask);
```

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`mask`	`IN`	Cryptographic usage mask.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with cryptographic usage mask attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resultttlv;
resultttlv = okvAttrAddCryptoUsageMask(env, ttlv, (ub4)12);
```

Related Topics

- okvAttrGetCryptoUsageMask
  `okvAttrGetCryptoUsageMask` gets the cryptographic usage mask attribute.

11.1.12 okvAttrAddDeactivationDate

`okvAttrAddDeactivationDate` adds a deactivation date attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddDeactivationDate` adds a deactivation date attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddDeactivationDate(OKVEnv *env, OKVTTLV *ttlv,
                                    ub8 deactivation_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`deactivation_date`	`IN`	Deactivation date

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with deactivation date attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
...
okvAttrAddDeactivationDate(env, ttlv, (ub8) current_date + 1800);
```

Related Topics

- okvAttrGetDeactivationDate
  `okvAttrGetDeactivationDate` gets the deactivation date attribute.

11.1.13 okvAttrAddDestroyDate

`okvAttrAddDestroyDate` adds a destroy date attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddDestroyDate` adds a destroy date attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddDestroyDate(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 destroy_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`destroy_date`	`IN`	Destroy date.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with destroy date attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
...
okvAttrAddDestroyDate(env, attr_in, (ub8) current_time + 24*3*3600);
```

Related Topics

- okvAttrGetDestroyDate
  `okvAttrGetDestroyDate` gets the destroy date attribute.

11.1.14 okvAttrAddDigest

`okvAttrAddDigest` adds a digest attribute to the OKVTTLV object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddDigest` adds a digest attribute to the OKVTTLV object.

Syntax

```
OKVTTLV *okvAttrAddDigest(OKVEnv *env, OKVTTLV *ttlv,
                          ub4 hash_algo, ub4 key_format_type,
                          oratext *digest, ub4 digestl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`hash_algo`	`IN`	Hash algorithm.
`key_format_type`	`IN`	Key format type.
`digest`	`IN`	Digest value.
`digestl`	`IN`	Digest value length.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with digest attribute values added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
oratext digest[] = "Sample_Digest";
okvAddAttributeObject(env, ttlv, OKVAttrDigest, 1);
resulttlv = okvAttrAddDigest(env, ttlv, HASH_ALG_SHA_1, OKVDEF_KEY_FMT_OPAQUE,
                             &digest[0], sizeof(digest))
```

Related Topics

- okvAttrGetDigest
  `okvAttrGetDigest` gets the digest attribute found at or after element index `elem_index`.
- okvAttrGetDigestLen
  `okvAttrGetDigestLen` gets the length of the digest value of the digest attribute found at or after element index `elem_index`.

11.1.15 okvAttrAddFresh

`okvAttrAddFresh` adds a fresh attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddFresh` adds a fresh attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddFresh(OKVEnv *env, OKVTTLV *ttlv, ub8 fresh);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`fresh`	`IN`	Fresh attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with fresh attribute value added. **Failure:** NULL pointer.

Comments

The fresh attribute can take values 0 or 1 for false or true respectively.

Example

```
OKVTTLV *resulttlv;
resulttlv = okvAttrAddFresh(env, attr_in, (ub8) 1);
```

Related Topics

- okvAttrGetFresh
  `okvAttrGetFresh` gets the fresh attribute.

11.1.16 okvAttrAddLeaseTime

`okvAttrAddLeaseTime` adds a lease time attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddLeaseTime` adds a lease time attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddLeaseTime(OKVEnv *env, OKVTTLV *ttlv,
                             ub4 lease_time);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`lease_time`	`IN`	Lease time.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with lease time attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resulttlv;
resulttlv = okvAttrAddLeaseTime(env, ttlv, (ub4)100);
```

Related Topics

- okvAttrGetLeaseTime
  `okvAttrGetLeaseTime` gets the lease time attribute.

11.1.17 okvAttrAddName

`okvAttrAddName` adds a name attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddName` adds a name attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddName(OKVEnv *env, OKVTTLV *ttlv,
                        oratext *name, ub4 namel, ub4 typ);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`name`	`IN`	Name value of the name attribute.
`name1`	`IN`	Length of name value of the name attribute.
`typ`	`IN`	Type of the name attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with name attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
oratext[] name_value = "sample name";
ub4 name_valuel = strlen((const char *)name_value);
okvAttrAddName(env, attr_in, (oratext *) name_value, name_valuel, 1);
```

Related Topics

- okvAttrGetName
  `okvAttrGetName` gets the name value, attribute index, and name type of the name attribute found at or after element index `elem_index`.

- okvAttrGetNameValueLen
  `okvAttrGetNameValueLen` gets the length of the name value of the name attribute found at or after element index `elem_index`.

11.1.18 okvAttrAddObjectGroup

`okvAttrAddObjectGroup` adds an object group attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddObjectGroup` adds an object group attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddObjectGroup(OKVEnv *env, OKVTTLV *ttlv,
                               oratext *object_group,
                               ub4 object_groupl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`object_group`	`IN`	Object group of the object group attribute.
`object_groupl`	`IN`	Length of object group of the object group attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with object group attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resulttlv;
oratext obj_grp[] = "Object_Group";
resulttlv = okvAttrAddObjectGroup(env, ttlv, &obj_grp[0], sizeof(obj_grp));
```

Related Topics

- okvAttrGetObjectGroup
  `okvAttrGetObjectGroup` gets the object group of the object group attribute found at or after element index `elem_index`.
- okvAttrGetObjectGroupLen
  `okvAttrGetObjectGroupLen` gets the length of the object group found at or after element index `elem_index`.

11.1.19 okvAttrAddObjectType

`okvAttrAddObjectType` adds an object type attribute to the OKVTTLV parent object.

Catetory

KMIP atrribute API

Purpose

`okvAttrAddObjectType` adds an object type attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddObjectType(OKVEnv *env, OKVTTLV *ttlv,
                              OKVObjNo typ);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`typ`	`IN`	Type of the object type attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with object type attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
okvAttrAddObjectType(env, req, OKVObjSymmetric);
```

Related Topics

- okvAttrGetObjectType
  `okvAttrGetObjectType` gets the object type attribute.

11.1.20 okvAttrAddProcessStartDate

`okvAttrAddProcessStartDate` adds a process start date attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddProcessStartDate` adds a process start date attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddProcessStartDate(OKVEnv *env, OKVTTLV *ttlv,
                                    ub8 process_start_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`process_start_date`	`IN`	Process start date.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with process start date attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
...
okvAttrAddProcessStartDate(env, attr_temp, (ub8) current_time + 1000);
```

Related Topics

- okvAttrGetProcessStartDate
  `okvAttrGetProcessStartDate` gets the process start date attribute.

11.1.21 okvAttrAddProtectStopDate

`okvAttrAddProtectStopDate` adds a protect stop date attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddProtectStopDate` adds a protect stop date attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddProtectStopDate(OKVEnv *env, OKVTTLV *ttlv,
                                   ub8 protect_stop_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`protect_stop_date`	`IN`	Protect stop date.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with protect stop date attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
time_t   current_time = time((time_t *)NULL);
...
okvAttrAddProtectStopDate(env, attr_in, (ub8) current_time + 24*3600);
```

Related Topics

- okvAttrGetProtectStopDate
  `okvAttrGetProtectStopDate` gets the protect stop date attribute.

11.1.22 okvAttrAddRevocationReason

`okvAttrAddRevocationReason` adds the revocation reason attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddRevocationReason` adds the revocation reason attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddRevocationReason(OKVEnv *env, OKVTTLV *ttlv,
                                   ub4 code, oratext *msg, ub4 msgl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`code`	`IN`	Revocation code.
`msg`	`IN`	Revocation message.
`msgl`	`IN`	Revocation message length.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with revocation attribute values added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
oratext msg[] = "Object_Compromised";
OKVTTLV *resulttlv;
resulttlv = okvAttrAddRevocationReason(env, ttlv, OKVDEF_REV_CODE_KEY_COMP,
&msg[0], sizeof(msg));
```

Related Topics

- okvAttrGetRevocationReason
  `okvAttrGetRevocationReason` gets the revocation reason attribute.

- okvAttrGetRevocationReasonMessageLen
  `okvAttrGetRevocationReasonMessageLen` gets the length of the revocation message of the revocation reason attribute.

11.1.23 okvAttrAddUniqueID

`okvAttrAddUniqueID` adds a unique identifier attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddUniqueID` adds a unique identifier attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddUniqueID(OKVEnv *env, OKVTTLV *ttlv,
                            oratext *uid, ub4 uidl);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`uid`	`IN`	Unique identifier.
`uid1`	`IN`	Unique identifier length.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with the unique identifier attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resultttlv;
oratext uidval[] = "12345678901234567890";
resultttlv = okvAttrAddUniqueID(env, ttlv, &uidval[0], strlen(uidval));
```

Related Topics

- okvAttrGetUniqueID
  `okvAttrGetUniqueID` gets the unique ID attribute.

- okvAttrGetUniqueIDLen
  `okvAttrGetUniqueIDLen` gets the length of the unique identifier attribute.

11.1.24 okvAttrAddUsageLimits

`okvAttrAddUsageLimits` adds a usage limits attribute to the OKVTTLV parent object.

Catetory

KMIP attribute API

Purpose

`okvAttrAddUsageLimits` adds a usage limits attribute to the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvAttrAddUsageLimits(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 total, ub8 count, ub4 unit);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`total`	`IN`	Usage limits total.
`count`	`IN`	Usage limits count.
`unit`	`IN`	Usage limits type.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to OKVTTLV object. **Success:** Pointer to OKVTTLV object with usage limits attribute value added. **Failure:** `NULL` pointer.

Comments

None.

Example

```
OKVTTLV *resulttlv;
resulttlv = okvAttrAddUsageLimits(env, ttlv, (ub8) 256, (ub8) 256, (ub4) 1);
```

Related Topics

- okvAttrGetUsageLimits
  `okvAttrGetUsageLimits` gets the usage limits attribute.

11.1.25 okvAttrGetActivationDate

`okvAttrGetActivationDate` gets the activation date attribute.

Catetory

KMIP attribute API

Purpose

`okvAttrGetActivationDate` gets the activation date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetActivationDate(OKVEnv *env, OKVTTLV *ttlv,
                                  ub8 *activation_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`activation_date`	`OUT`	Activation date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 act_date;
okvAttrGetActivationDate(env, ttlv, &act_date);
```

Related Topics

- okvAttrAddActivationDate
  `okvAttrAddActivationDate` adds the activation date attribute to OKVTTLV parent object.

11.1.26 okvAttrGetArchiveDate

`okvAttrGetArchiveDate` gets the archive date attribute.

Catetory

KMIP attribute API

Purpose

`okvAttrGetArchiveDate` gets the archive date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetArchiveDate(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 *archive_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV Object.
`archive_date`	`OUT`	Archive date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 archive_date;
okvAttrGetArchiveDate(env, ttlv, &archive_date);
```

11.1.27 okvAttrGetCompromiseDate

`okvAttrGetCompromiseDate` gets the compromise date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetCompromiseDate` gets the compromise date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetCompromiseDate(OKVEnv *env, OKVTTLV *ttlv,
                                  ub8 *comp_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV object.
`comp_date`	`OUT`	Compromise date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 comp_date;
okvAttrGetCompromiseDate(env, ttlv, &comp_date);
```

Related Topics

- okvAttrAddCompromiseDate
  `okvAttrAddCompromiseDate` adds a compromise date attribute to OKVTTLV parent object.

11.1.28 okvAttrGetCompromiseOccurrenceDate

`okvAttrGetCompromiseOccurrenceDate` gets the compromise occurrence date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetCompromiseOccurrenceDate` gets the compromise occurrence date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetCompromiseOccurrenceDate(OKVEnv *env, OKVTTLV *ttlv,
                                            ub8 *comp_occr_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault Environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV object.
`comp_occr_date`	`OUT`	Compromise occurrence date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 comp_occ_date;
OKVTTLV *resultttlv;
resultttlv = okvAttrGetCompromiseOccurrenceDate(env, ttlv, &comp_occ_date);
```

Related Topics

- okvAttrAddCompromiseOccurrenceDate
  `okvAttrAddCompromiseOccurrenceDate` adds a compromise occurrence date attribute to OKVTTLV parent object.

11.1.29 okvAttrGetContactInfo

`okvAttrGetContactInfo` gets the contact information attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetContactInfo` gets the contact information attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetContactInfo(OKVEnv *env, OKVTTLV *ttlv,
                               oratext *contact_info, ub4 *contact_infol);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault Environment handle
`ttlv`	`IN`	Pointer to the parent OKVTTLV Object
`contact_info`	`OUT`	Contact information
`contact_infol`	`OUT`	Length of contact information

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory for the argument `contact_info` must be allocated before you execute the `okvAttrGetContactInfo` function call. You can find the size by using `okvAttrGetContactInfoLen`.

Example

```
ub4 contactl;
contactl = okvAttrGetContactInfoLen(env, ttlv);
contact  = calloc(contactl, sizeof(oratext));
okvAttrGetContactInfo(env, ttlv, contact, &contactl);
```

Related Topics

- okvAttrAddContactInfo
  `okvAttrAddContactInfo` adds a contact information attribute to the OKVTTLV parent object.
- okvAttrGetContactInfoLen
  `okvAttrGetContactInfoLen` gets the length of the contact information of the contact information attribute.

11.1.30 okvAttrGetContactInfoLen

`okvAttrGetContactInfoLen` gets the length of the contact information of the contact information attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetContactInfoLen` gets the length of the contact information attribute.

Syntax

```
ub4 okvAttrGetContactInfoLen(OKVEnv *env, OKVTTLV *ttlv));
```

Parameters

Parameters	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault Environment handle
`ttlv`	`IN`	Pointer to parent OKVTTLV Object

Return Values

Return Value	Description
`ub4`	Length of the attribute value. **Success:** Length of the contact information attribute value. **Failure:** 0.

Comments

None.

Example

```
ub4 cinfol;
cinfol = okvAttrGetContactInfoLen(env, ttlv);
```

Related Topics

- okvAttrAddContactInfo
  `okvAttrAddContactInfo` adds a contact information attribute to the OKVTTLV parent object.
- okvAttrGetContactInfo
  `okvAttrGetContactInfo` gets the contact information attribute.

11.1.31 okvAttrGetCryptoAlgo

`okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute from the parent OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetCryptoAlgo(OKVEnv *env, OKVTTLV *ttlv,
                              ub4 *crypto_alg);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault Environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV object.
`crypto_alg`	`OUT`	Cryptographic algorithm.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4    alg = 0;
okvAttrGetCryptoAlgo(env, ttlv, &alg);
```

Related Topics

- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.

- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.
- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

11.1.32 okvAttrGetCryptoLen

`okvAttrGetCryptoLen` gets the cryptographic length attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetCryptoLen` gets the cryptographic length attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetCryptoLen(OKVEnv *env, OKVTTLV *ttlv,
                             ub4 *crypto_len);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle
`ttlv`	`IN`	Pointer to the OKVTTLV object
`crypto_len`	`OUT`	Cryptographic length

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 len = 0;
okvAttrGetCryptoLen(env, ttlv, &len);
```

Related Topics

- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.
- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.
- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrGetCryptoParams
  `okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

11.1.33 okvAttrGetCryptoParams

`okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index`.

Category

KMIP attribute API

Purpose

`okvAttrGetCryptoParams` gets the cryptographic parameters attribute found at or after element index `elem_index` in the parent TTLV.

Syntax

```
OKVErrNo okvAttrGetCryptoParams(OKVEnv *env, OKVTTLV *ttlv,
                                ub4 *elem_index,
                                ub4 *block_cipher_mode, ub4 *padding_method,
                                ub4 *hashing_algorithm, ub4 *key_role_type);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent TTLV object.
`elem_index`	`IN/OUT`	Element index.
`block_cipher_mode`	`OUT`	Block cipher mode.
`padding_method`	`OUT`	Padding method.
`hashing_algorithm`	`OUT`	Hashing algorithm.
`key_role_type`	`OUT`	Key role type.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4  bcm, pm, ha, krt, ind;
okvAttrGetCryptoParams(env, ttlv, &ind, &bcm, &pm, &ha, &krt);
```

Related Topics

- okvAttrAddCryptoAlgo
  `okvAttrAddCryptoAlgo` adds a cryptographic algorithm attribute to the OKVTTLV parent object.
- okvAttrAddCryptoLen
  `okvAttrAddCryptoLen` adds a cryptographic length attribute to the OKVTTLV parent object.
- okvAttrAddCryptoParams
  `okvAttrAddCryptoParams` adds cryptographic parameters to the OKVTTLV parent object.
- okvAttrGetCryptoAlgo
  `okvAttrGetCryptoAlgo` gets the cryptographic algorithm attribute.
- okvAttrGetCryptoLen
  `okvAttrGetCryptoLen` gets the cryptographic length attribute.

11.1.34 okvAttrGetCryptoUsageMask

`okvAttrGetCryptoUsageMask` gets the cryptographic usage mask attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetCryptoUsageMask` gets the cryptographic usage mask attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetCryptoUsageMask(OKVEnv *env, OKVTTLV *ttlv,
                                   ub4 *mask);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle
ttlv	IN	Pointer to parent OKVTTLV object
mask	OUT	Cryptographic usage mask

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 mask = 0;
okvAttrGetCryptoUsageMask(env, ttlv, &mask);
```

Related Topics

- okvAttrAddCryptoUsageMask
  `okvAttrAddCryptoUsageMask` adds a cryptographic usage mask attribute to the OKVTTLV parent object.

11.1.35 okvAttrGetDeactivationDate

`okvAttrGetDeactivationDate` gets the deactivation date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetDeactivationDate` gets the deactivation date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetDeactivationDate(OKVEnv *env, OKVTTLV *ttlv,
                                    ub8 *deactivation_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle
ttlv	IN	Pointer to OKVTTLV object
deactivation_date	OUT	Deactivation date

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 deact_date;
okvAttrGetDeactivationDate(env, ttlv, &deact_date);
```

Related Topics

- okvAttrAddDeactivationDate
  `okvAttrAddDeactivationDate` adds a deactivation date attribute to the OKVTTLV parent object.

11.1.36 okvAttrGetDestroyDate

`okvAttrGetDestroyDate` gets the destroy date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetDestroyDate` gets the destroy date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetDestroyDate(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 *destroy_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle

Parameter	IN/OUT	Description
ttlv	IN	Pointer to OKVTTLV object
destroy_date	OUT	Destroy date

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 destroy_date;
okvAttrGetDestroyDate(env, ttlv, &destroy_date);
```

11.1.37 okvAttrGetDigest

okvAttrGetDigest gets the digest attribute found at or after element index elem_index.

Category

KMIP attribute API

Purpose

okvAttrGetDigest gets the digest parameters for the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetDigest(OKVEnv *env, OKVTTLV *ttlv,
                          ub4 *elem_index,
                          ub4 *hash_algo, ub4 *key_format_type,
                          oratext *digest, ub4 *digestl);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to parent OKVTTLV object.

Parameter	IN/OUT	Description
elem_index	IN/OUT	The API will look for the first occurrence of the child with tag OKVDEF_TAG_DIGEST from this index onwards and will return the actual index where the child was found.
hash_algo	OUT	Hash algorithm.
key_format_type	OUT	Key format type.
digest	OUT	Digest value.
digestl	IN/OUT	Digest value length.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory to store the digest value has to be allocated before this API is called. okvAttrGetDigestLen can be used to get the digest length.

Example

```
ub4 ei = 0, digestl = 0, hash_algo, key_format_type;
digestl = okvAttrGetDigestLen(env, ttlv, &ei);
oratext * digest = (oratext *) calloc(digestl ,sizeof(oratext) );
okvAttrGetDigest(env, ttlv, &ei, &hash_algo, &key_format_type,
                 &digest[0], &digestl);
```

Related Topics

- okvAttrAddDigest
  okvAttrAddDigest adds a digest attribute to the OKVTTLV object.
- okvAttrGetDigestLen
  okvAttrGetDigestLen gets the length of the digest value of the digest attribute found at or after element index elem_index.

11.1.38 okvAttrGetDigestLen

okvAttrGetDigestLen gets the length of the digest value of the digest attribute found at or after element index elem_index.

Category

KMIP attribute API

Purpose

okvAttrGetDigestLen gets the length of the digest attribute from the OKVTTLV object.

Syntax

```
ub4 okvAttrGetDigestLen(OKVEnv *env, OKVTTLV *ttlv,
                        ub4 *elem_index);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to the OKVTTLV object.
elem_index	IN/OUT	The API will look for the first occurrence of the child with tag OKVDEF_TAG_DIGEST from this index onwards and will return the actual index where the child was found.

Return Values

Return Value	Description
ub4	Length of the attribute value. **Success:** Length of the digest attribute. **Failure:** 0.

Comments

None.

Example

```
ub4 ei = 0;
ub4 digestl;
digestl = okvAttrGetDigestLen(env, ttlv, &ei)
```

Related Topics

- okvAttrAddDigest
  `okvAttrAddDigest` adds a digest attribute to the OKVTTLV object.
- okvAttrGetDigest
  `okvAttrGetDigest` gets the digest attribute found at or after element index `elem_index`.

11.1.39 okvAttrGetFresh

`okvAttrGetFresh` gets the fresh attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetFresh` gets the fresh attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetFresh(OKVEnv *env, OKVTTLV *ttlv, ub8 *fresh);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle
`ttlv`	`IN`	Pointer to the parent OKVTTLV Object
`fresh`	`OUT`	Fresh attribute

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 fresh_val;
okvAttrGetFresh(env, attr, &fresh_val);
```

Related Topics

- okvAttrAddFresh
  `okvAttrAddFresh` adds a fresh attribute to the OKVTTLV parent object.

11.1.40 okvAttrGetInitialDate

`okvAttrGetInitialDate` gets the initial date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetInitialDate` gets the initial date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetInitialDate(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 *initial_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault Environment handle.
ttlv	IN	Pointer to OKVTTLV object.
initial_date	OUT	Initial date.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 ini_date;
okvAttrGetInitialDate(env, ttlv, &ini_date);
```

11.1.41 okvAttrGetLastChangeDate

`okvAttrGetLastChangeDate` gets the last change date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetLastChangeDate` gets the last change date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetLastChangeDate(OKVEnv *env, OKVTTLV *ttlv,
                                  ub8 *last_change_date);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.

Parameter	IN/OUT	Description
ttlv	IN	Pointer to OKVTTLV object.
last_change_date	OUT	Last change date.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 last_change_date;
okvAttrGetLastChangeDate(env, ttlv, &last_change_date);
```

11.1.42 okvAttrGetLeaseTime

`okvAttrGetLeaseTime` gets the lease time attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetLeaseTime` gets the lease time attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetLeaseTime(OKVEnv *env, OKVTTLV *ttlv,
                             ub4 *lease_time);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault Environment handle.
ttlv	IN	Pointer to the OKVTTLV object.
lease_time	OUT	Lease time.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 lease_time;
okvAttrGetLeaseTime(env, ttlv, &lease_time);
```

11.1.43 okvAttrGetName

okvAttrGetName gets the name value, attribute index, and name type of the name attribute found at or after element index elem_index.

Category

KMIP attribute API

Purpose

okvAttrGetName gets the name value, name length, and name type of the name attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetName(OKVEnv *env, OKVTTLV *ttlv,
                        ub4 *elem_index,
                        oratext *name, ub4 namel, ub4 *typ)
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to the parent TTLV object.
elem_index	IN/OUT	The API will look for the first occurrence of the child with tag OKVDEF_TAG_NAME_ST from this index onwards and will return the actual index where the child was found.
name	OUT	Name value of the name attribute.
name1	IN/OUT	Length of name value of the name attribute.

Parameter	IN/OUT	Description
typ	OUT	Type of the name attribute.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory to store the name value should be allocated before the function call.

Example

```
ub4 ei = 0;
ub4 nametyp;
namel = okvAttrGetNameValueLen(env, ttlv, &ei);
oratext * name = (oratext *) calloc(namel, sizeof(oratext));
okvAttrGetName(env, ttlv, &ei, name, namel, &nametyp);
```

Related Topics

- okvAttrGetNameValueLen
  `okvAttrGetNameValueLen` gets the length of the name value of the name attribute found at or after element index `elem_index`.
- okvAttrAddName
  `okvAttrAddName` adds a name attribute to the OKVTTLV parent object.

11.1.44 okvAttrGetNameValueLen

`okvAttrGetNameValueLen` gets the length of the name value of the name attribute found at or after element index `elem_index`.

Category

KMIP attribute API

Purpose

`okvAttrGetNameValueLen` gets the length of the name attribute from the OKVTTLV object.

Syntax

```
ub4 okvAttrGetNameValueLen(OKVEnv *env, OKVTTLV *ttlv,
                           ub4 *elem_index;
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to the OKVTTLV buffer to search the name length.
elem_index	IN/OUT	The API will look for the first occurrence of the child with tag OKVDEF_TAG_NAME_ST from this index onwards and will return the actual index where the child was found.

Return Values

Return Value	Description
ub4	Length of the attribute value. **Success:** Length of the name attribute. **Failure:** 0.

Comments

None.

Example

```
ei = 0;
namel = okvAttrGetNameValueLen(env, ttlv, &ei);
```

Related Topics

- okvAttrGetName
  okvAttrGetName gets the name value, attribute index, and name type of the name attribute found at or after element index elem_index.
- okvAttrAddName
  okvAttrAddName adds a name attribute to the OKVTTLV parent object.

11.1.45 okvAttrGetObjectGroup

okvAttrGetObjectGroup gets the object group of the object group attribute found at or after element index elem_index.

Category

KMIP attribute API

Purpose

okvAttrGetObjectGroup gets the object group attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetObjectGroup(OKVEnv *env, OKVTTLV *ttlv,
                               ub4 *elem_index,
                               oratext *object_group, ub4 *object_groupl);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Pointer to the Oracle Key Vault environment handle.
ttlv	IN	Pointer to OKVTTLV object.
elem_index	IN/OUT	The API will look for the first occurrence of the child with tag OKVDEF_TAG_OBJ_GROUP from this index onwards and will return the actual index where the child was found.
object_group	OUT	Object group of the object group attribute.
object_groupl	IN/OUT	Length of object group of the object group attribute.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory for object_group must be allocated before you execute the function call. Use okvAttrGetObjectGroupLen to find the size of memory to be allocated.

Example

```
ub4 index = 0, obj_grpl;
obj_grpl = okvAttrGetObjectGroupLen(env, ttlv, &index);
oratext *obj_group = (oratext *)calloc(obj_grpl, sizeof(oratext);
okvAttrGetObjectGroup(env, ttlv, &index, &obj_group, &obj_grpl);
```

Related Topics

- okvAttrGetObjectGroupLen
  okvAttrGetObjectGroupLen gets the length of the object group found at or after element index elem_index.
- okvAttrAddObjectGroup
  okvAttrAddObjectGroup adds an object group attribute to the OKVTTLV parent object.

11.1.46 okvAttrGetObjectGroupLen

`okvAttrGetObjectGroupLen` gets the length of the object group found at or after element index `elem_index`.

Category

KMIP attribute API

Purpose

`okvAttrGetObjectGroupLen` gets the length of the object group from the OKVTTLV object.

Syntax

```
ub4 okvAttrGetObjectGroupLen(OKVEnv *env, OKVTTLV *ttlv,
                             ub4 *elem_index);
```

Parameters

Parameter	IN/OUT	Element index
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV object.
`elem_index`	`IN/OUT`	The API will look for the first occurrence of the child with tag `OKVDEF_TAG_OBJ_GROUP` from this index onwards and will return the actual index where the child was found.

Return Values

Return Value	Description
`ub4`	Length of the attribute value. **Success:** Length of the object group attribute value. **Failure:** 0.

Comments

None.

Example

```
ub4 obj_grpl, index = 0;
obj_grpl = okvAttrGetObjectGroupLen(env, ttlv, &index);
```

Related Topics

- okvAttrAddObjectGroup
  `okvAttrAddObjectGroup` adds an object group attribute to the OKVTTLV parent object.
- okvAttrGetObjectGroup
  `okvAttrGetObjectGroup` gets the object group of the object group attribute found at or after element index `elem_index`.

11.1.47 okvAttrGetObjectType

`okvAttrGetObjectType` gets the object type attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetObjectType` gets the object type attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetObjectType(OKVEnv *env, OKVTTLV *ttlv,
                              OKVObjNo *typ)
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to OKVTTLV object.
`typ`	`OUT`	Type of the object type attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** 0KV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 typ;
okvAttrGetObjectType(env, ttlv, &typ);
```

Related Topics

- okvAttrAddObjectType
  `okvAttrAddObjectType` adds an object type attribute to the OKVTTLV parent object.

11.1.48 okvAttrGetProcessStartDate

`okvAttrGetProcessStartDate` gets the process start date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetProcessStartDate` gets the process start date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetProcessStartDate(OKVEnv *env, OKVTTLV *ttlv,
                                    ub8 *process_start_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to OKVTTLV object.
`process_start_date`	`OUT`	Process start date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8  pstart_date;
okvAttrGetProcessStartDate(env, ttlv, &pstart_date);
```

Related Topics

- okvAttrAddProcessStartDate
  `okvAttrAddProcessStartDate` adds a process start date attribute to the OKVTTLV parent object.

11.1.49 okvAttrGetProtectStopDate

`okvAttrGetProtectStopDate` gets the protect stop date attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetProtectStopDate` gets the protect stop date attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetProtectStopDate(OKVEnv *env, OKVTTLV *ttlv,
                                   ub8 *protect_stop_date);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to OKVTTLV parent object.
`protect_stop_date`	`OUT`	Protect stop date.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub8 pstop_date;
okvAttrGetProtectStopDate(env, ttlv, &pstop_date);
```

Related Topics

- okvAttrAddProtectStopDate
  `okvAttrAddProtectStopDate` adds a protect stop date attribute to the OKVTTLV parent object.

11.1.50 okvAttrGetRevocationReason

`okvAttrGetRevocationReason` gets the revocation reason attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetRevocationReason` gets the revocation reason attribute parameters from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetRevocationReason(OKVEnv *env, OKVTTLV *ttlv,
                                    oratext *msg, ub4 *msgl, ub4 *code);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to parent OKVTTLV object.
`code`	`OUT`	Revocation code.
`msg`	`OUT`	Revocation message.
`msgl`	`IN/OUT`	Revocation message length.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory for the `msg` argument must be allocated before the function call. The size of memory to be allocated can be retrieved by using `okvAttrGetRevocationReasonMessageLen`.

Example

```
ub4 code;
msgl = okvAttrGetRevocationReasonMessageLen(env, ttlv);
revoke_reason = calloc(msgl, sizeof(oratext));
okvAttrGetRevocationReason(env, ttlv, revoke_reason, &msgl, &code);
```

Related Topics

- okvAttrAddRevocationReason
  `okvAttrAddRevocationReason` adds the revocation reason attribute to the OKVTTLV parent object.
- okvAttrGetRevocationReasonMessageLen
  `okvAttrGetRevocationReasonMessageLen` gets the length of the revocation message of the revocation reason attribute.

11.1.51 okvAttrGetRevocationReasonMessageLen

`okvAttrGetRevocationReasonMessageLen` gets the length of the revocation message of the revocation reason attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetRevocationReasonMessageLen` gets the length of the revocation message of the revocation reason attribute from the OKVTTLV object.

Syntax

```
ub4 okvAttrGetRevocationReasonMessageLen(OKVEnv *env, OKVTTLV *ttlv);
```

Parameters

Parameters	IN/OUT	Description
`env`	`IN`	Pointer to Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV object.

Return Values

Return Value	Description
`ub4`	Length of the attribute value. **Success:** Length of the revocation message. **Failure:** 0.

Comments

None.

Example

```
ub4 msgl;
msgl = okvAttrGetRevocationReasonMessageLen(env, ttlv);
```

Related Topics

- okvAttrAddRevocationReason
  `okvAttrAddRevocationReason` adds the revocation reason attribute to the OKVTTLV parent object.
- okvAttrGetRevocationReason
  `okvAttrGetRevocationReason` gets the revocation reason attribute.

11.1.52 okvAttrGetState

`okvAttrGetState` gets the state attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetState` gets the state attribute from OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetState(OKVEnv *env, OKVTTLV *ttlv, ub4 *state);
```

Parameters

Parameter	IN/OUT	State
`env`	`IN`	Pointer to parent OKVTTLV object.
`ttlv`	`IN`	Pointer to OKVTTLV parent object.
`state`	`OUT`	State.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 val_state;
okvAttrGetState(env, attr, &val_state)
```

11.1.53 okvAttrGetUniqueID

`okvAttrGetUniqueID` gets the unique ID attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetUniqueID` gets the unique ID attribute from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetUniqueID(OKVEnv *env, OKVTTLV *ttlv,
                            ub4 *elem_index,
                            oratext *uid, ub4 *uidl)
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to the OKVTTLV parent object.
`elem_index`	`IN/OUT`	The API will look for the first occurrence of the child with tag `OKVDEF_TAG_ID` from this index onwards and will store the actual index where the child was found.
`uid`	`OUT`	Pointer to memory storing the unique identifier.
`uid1`	`IN/OUT`	Unique identifier length.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

The memory to store the UID should be allocated before API is called. To fetch the length, execute `okvAttrGetUniqueIDLen`.

Example

```
ub4 index = 0;
ouidl = okvAttrGetUniqueIDLen(env, ttlv, &index);
oratext *ouid = (oratext *) calloc (ouidl,sizeof(oratext));
okvAttrGetUniqueID(env, ttlv, &i, &ouid[0], &ouidl);
```

Related Topics

- okvAttrAddUniqueID
  `okvAttrAddUniqueID` adds a unique identifier attribute to the OKVTTLV parent object.
- okvAttrGetUniqueIDLen
  `okvAttrGetUniqueIDLen` gets the length of the unique identifier attribute.

11.1.54 okvAttrGetUniqueIDLen

`okvAttrGetUniqueIDLen` gets the length of the unique identifier attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetUniqueIDLen` gets the unique identifier length in the TTLV.

Syntax

```
ub4 okvAttrGetUniqueIDLen(OKVEnv *env, OKVTTLV *ttlv,
                          ub4 *elem_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object containing the UID.
`elem_index`	`IN/OUT`	The API will look for the first occurrence of the child with tag `OKVDEF_TAG_ID` from this index onwards and will store the actual index where the child was found.

Return Values

Return Value	Description
`ub4`	Length of the attribute value. **Success:** Positive integer indicating the length of UID. **Failure:** 0.

Comments

None.

Example

```
ouidl = okvAttrGetUniqueIDLen(env, ttlv, &index);
okvAttrGetUniqueID(env, ttlv, &index, &ouid[0], &ouidl);
```

Related Topics

- okvAttrAddUniqueID
  `okvAttrAddUniqueID` adds a unique identifier attribute to the OKVTTLV parent object.
- okvAttrGetUniqueID
  `okvAttrGetUniqueID` gets the unique ID attribute.

11.1.55 okvAttrGetUsageLimits

`okvAttrGetUsageLimits` gets the usage limits attribute.

Category

KMIP attribute API

Purpose

`okvAttrGetUsageLimits()` gets the usage limit attributed from the OKVTTLV object.

Syntax

```
OKVErrNo okvAttrGetUsageLimits(OKVEnv *env, OKVTTLV *ttlv,
                               ub8 *total, ub8 *count, ub4 *unit);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	Oracle TTLV parent object.
`total`	`OUT`	Usage limits total.
`count`	`OUT`	Usage limits count.
`unit`	`OUT`	Usage limits type.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
ub4 utype;
ub8 utotal, ucount;
okvAttrGetUsageLimits(env, ttlv, &utotal, &ucount, &utype);
```

Related Topics

- okvAttrAddUsageLimits
  `okvAttrAddUsageLimits` adds a usage limits attribute to the OKVTTLV parent object.

11.1.56 okvGetAttributeObject

`okvGetAttributeObject` gets the attribute, its name, and index from the parent attribute found at element index `elem_index`.

Category

KMIP attribute API

Purpose

`okvGetAttributeObject` gets the attribute, its name and index from the parent attribute found at element index `elem_index`. The actual attribute value can be retrieved using one of the other functions defined in this section.

Syntax

```
OKVErrNo okvGetAttributeObject(OKVEnv *env, OKVTTLV *ttlv,
                               ub4 *elem_index,
                               OKVAttrNo *attrno, ub4 *attr_index, OKVTTLV **attr);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Pointer to the Oracle Key Vault environment handle.
`ttlv`	`IN`	Pointer to OKVTTLV object.
`elem_index`	`IN/OUT`	The API will return the attribute number, attribute index and attribute value at the child at index pointed to by `elem_index`.
`attrno`	`OUT`	Oracle Key Vault attribute number.
`attr_index`	`OUT`	Attribute index.
`attr`	`OUT`	TTLV object containing attribute value.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

This API is useful when it is required to parse the parent TTLV and find the various attribute and their corresponding values. The following example shows how we can use this API to parse a KMIP TTLV for name attribute.

Example

```
ub4 i = 0,  ret_name_attrl, attrno, ret_index_name_attr;
OKVTTLV *name_attr_out;
oratext * ret_name_attr;

while(OKV_ATTR_NOT_FOUND !=
        okvGetAttributeObject(env, attr_out_list, &i,
                              &attrno, &ret_index_name_attr, &name_attr_out))
{
   if(attrno == OKVAttrName)
   {
      ret_name_attrl = okvAttrGetNameValueLen(env, name_attr_out, &i);
      ret_name_attr = calloc(ret_name_attrl, sizeof(oratext));
      okvAttrGetName(env, name_attr_out, &i,
                     ret_name_attr, ret_name_attrl, &typ);
      break;
   }
   else
   {
      i++;
   }
}
```

Related Topics

- okvAddAttributeObject
  `okvAddAttributeObject` adds an attribute object to the parent `TTLV` object.

11.2 Oracle Key Vault KMIP Custom Attribute APIs

The KMIP Custom Attribute APIs are listed in this section.

- About the KMIP Custom Attributes API
  The Oracle Key Vault KMIP custom attributes follow a set of strict guidelines.
- okvCustomAttrAddBoolean
  `okvCustomAttrAddBoolean` adds a Boolean data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddByteString
  `okvCustomAttrAddByteString` adds a byte string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddDateTime
  `okvCustomAttrAddDateTime` adds a date time data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddEnum
  `okvCustomAttrAddEnum` adds an enumeration data type to the custom attribute specified by the OKVTTLV parent object.

- okvCustomAttrAddInteger
  `okvCustomAttrAddInteger` adds an integer data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddInterval
  `okvCustomAttrAddInterval` adds an interval data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddLongInteger
  `okvCustomAttrAddLongInteger` adds a long integer data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddStructure
  `okvCustomAttrAddStructure` adds a `TTLV` structure to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrAddTextString
  `okvCustomAttrAddTextString` adds a text string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrGet
  `okvCustomAttrGet` finds any custom attribute at or after element index `elem_ind`.
- okvCustomAttrGetBoolean
  `okvCustomAttrGetBoolean` fetches a Boolean data type found at the element index `elem_ind`.
- okvCustomAttrGetByName
  `okvCustomAttrGetByName` finds custom attribute with name `name` at or after the element index `elem_ind`.
- okvCustomAttrGetByteString
  `okvCustomAttrGetByteString` returns the byte string data type found at the element index `elem_ind`.
- okvCustomAttrGetByteStringLen
  `okvCustomAttrGetByteStringLen` returns the length of the byte string data type found at the element index `elem_ind`.
- okvCustomAttrGetByType
  `okvCustomAttrGetByType` finds any custom attribute of type `typ` at or after the element index `elem_ind`, and returns the actual element index and length of the name of the custom attribute.
- okvCustomAttrGetDateTime
  `okvCustomAttrGetDateTime` returns a date time data type found at the element index `elem_ind`.
- okvCustomAttrGetEnum
  `okvCustomAttrGetEnum` returns an enumeration data type found at the element index `elem_ind`.
- okvCustomAttrGetInteger
  Returns an integer data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.
- okvCustomAttrGetInterval
  `okvCustomAttrGetInterval` returns an interval data type found at the element index `elem_ind`.

- okvCustomAttrGetLongInteger
  `okvCustomAttrGetLongInteger` returns a long integer data type found at the element index `elem_ind`.
- okvCustomAttrGetStructure
  `okvCustomAttrGetStructure` returns a `TTLV` structure data type found at element index `elem_ind`.
- okvCustomAttrGetTextString
  `okvCustomAttrGetTextString` returns the text string data type found at the element index `elem_ind`.
- okvCustomAttrGetTextStringLen
  `okvCustomAttrGetTextStringLen` returns the length of the text string data type found at the element index `elem_ind`.

11.2.1 About the KMIP Custom Attributes API

The Oracle Key Vault KMIP custom attributes follow a set of strict guidelines.

The custom attributes can have structures but the structures cannot have structures within them.

All client KMIP custom attribute names can begin with `x-` only. Any Oracle Key Vault custom attribute not starting with this prefix will be rejected.

Functions that return the `OKVTTLV` object have the following interpretation:

- On success, a valid `OKVTTLV` object for the attribute is returned.
- On failure, a `NULL` pointer is returned.

Functions that return the length of the retrieved object have the following interpretation:

- On success, a length of the buffer is returned.
- On failure, zero (0) is returned.

The following Oracle Key Vault functions add and retrieve custom attributes of a specific KMIP Data type from the `OKVTTLV` object. The functions have the following construction in general:

- `okvCustomAttrAdd`*`Type`* adds the custom attribute name and value to the `OKVTTLV` parent object.
- `okvCustomAttrGet`*`Type`*`Len` gets the length of the custom attribute value from the `OKVTTLV` parent object for KMIP Data Types that don't have fixed length.
- `okvCustomAttrGet`*`Type`* gets the custom attribute name and value from the `OKVTTLV` parent object.

The attribute index and element index used in the Oracle Key Vault functions have the same meaning as the standard Oracle Key Vault KMIP attribute APIs, but with the following exception:

Custom attributes have iterator functions because the endpoint may not know the type of the custom attribute with a given name. This makes it necessary to iterate over all types of custom attributes. But when a custom attribute with a given name is located, then the element index points that attribute for certain. Therefore, the get functions for

the custom attributes do not have to look for the attribute at all and do not have to update the element index passed in.

Related Topics

- About the Oracle Key Vault KMIP Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.

11.2.2 okvCustomAttrAddBoolean

`okvCustomAttrAddBoolean` adds a Boolean data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddBoolean` adds a Boolean data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddBoolean(OKVEnv *env, OKVTTLV *ttlv,
                                  oratext *name, ub4 namel,
                                  ub8 bool_val, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`bool_val`	`IN`	Boolean value to be added to the custom attribute
`attr_index`	`IN`	Attribute index of the Boolean custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with Boolean data type added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddBoolean(env, req, "x-My Boolean",
                                  strlen("x-My Boolean"),
                                  00000001, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetBoolean
  `okvCustomAttrGetBoolean` fetches a Boolean data type found at the element index `elem_ind`.

11.2.3 okvCustomAttrAddByteString

`okvCustomAttrAddByteString` adds a byte string data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddByteString` adds a byte string data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvCustomAttrAddByteString(OKVEnv *env, OKVTTLV *ttlv,
                                    oratext *name, ub4 namel,
                                    ub1 *byte, ub4 bytel, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`byte`	`IN`	Byte string
`bytel`	`IN`	Length of byte string
`attr_index`	`IN`	Attribute index of the byte string custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with byte string added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
ub1 *byte = "ByteOne";
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddByteString(env, req, "x-My Byte String",
                                     strlen("x-My Byte String"),
                                     byte, strlen(byte), 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetByteStringLen
  `okvCustomAttrGetByteStringLen` returns the length of the byte string data type found at the element index `elem_ind`.
- okvCustomAttrGetByteString
  `okvCustomAttrGetByteString` returns the byte string data type found at the element index `elem_ind`.

11.2.4 okvCustomAttrAddDateTime

`okvCustomAttrAddDateTime` adds a date time data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddDateTime` adds a date time data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddDateTime(OKVEnv *env, OKVTTLV *ttlv,
                                   oratext *name, ub4 namel,
                                   ub8 date_time, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`date_time`	`IN`	Date time value to be added to the custom attribute
`attr_index`	`IN`	Attribute index of the date time custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with date time custom attribute added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
time_t   current_time = time((time_t *)NULL);
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddDateTime(env, req, "x-My Date Time",
                                   strlen("x-My Date Time"),
                                   (ub8) current_time, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetDateTime
  `okvCustomAttrGetDateTime` returns a date time data type found at the element index `elem_ind`.

11.2.5 okvCustomAttrAddEnum

`okvCustomAttrAddEnum` adds an enumeration data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddEnum` adds an enumeration data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddEnum(OKVEnv *env, OKVTTLV *ttlv,
                               oratext *name, ub4 namel,
                               ub4 enumval, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`enumval`	`IN`	Enum value to be added to the custom attribute
`attr_index`	`IN`	Attribute index of enumeration custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with enumeration values added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddEnum(env, req, "x-My Enumeration",
                               strlen("x-My Enumeration"),
                               CRYPTO_ALG_DES, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetEnum
  `okvCustomAttrGetEnum` returns an enumeration data type found at the element index `elem_ind`.

11.2.6 okvCustomAttrAddInteger

`okvCustomAttrAddInteger` adds an integer data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddInteger` adds an integer data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddInteger(OKVEnv *env, OKVTTLV *ttlv,
                                  oratext *name, ub4 namel,
                                  ub4 integer, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`integer`	`IN`	Integer value to be added to the custom attribute
`attr_ind`	`IN`	Attribute index of the integer custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with integer data type added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddInteger(env, req, "x-My Integer",
```

```
                                    strlen("x-My Integer"), 32, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetInteger
  Returns an integer data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

11.2.7 okvCustomAttrAddInterval

`okvCustomAttrAddInterval` adds an interval data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddInterval` adds an interval data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddInterval(OKVEnv *env, OKVTTLV *ttlv,
                                   oratext *name, ub4 namel,
                                   ub4 interval, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`name`	`IN`	Name of custom attribute
`namel`	`IN`	Length of name of custom attribute
`interval`	`IN`	Interval value to be added to the custom attribute
`attr_ind`	`IN`	Attribute index of the Interval custom attribute

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with interval data type added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddInterval(env, req, "x-My Interval",
                                        strlen("x-My Interval"),
                                        5, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetInterval
  `okvCustomAttrGetInterval` returns an interval data type found at the element index `elem_ind`.

11.2.8 okvCustomAttrAddLongInteger

`okvCustomAttrAddLongInteger` adds a long integer data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddLongInteger` adds a long integer data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddLongInteger(OKVEnv *env, OKVTTLV *ttlv,
                                      oratext *name, ub4 namel,
                                      ub8 long_integer, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`name`	`IN`	Name of custom attribute.
`namel`	`IN`	Length of name of custom attribute.
`long_integer`	`IN`	Long integer value to be added to the custom attribute.
`attr_index`	`IN`	Attribute index of long integer custom attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with long integer data type added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddLongInteger(env, req, "x-My Long Integer",
                                      strlen("x-My Long Integer"),
                                      (ub8) 32, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetLongInteger
  `okvCustomAttrGetLongInteger` returns a long integer data type found at the element index `elem_ind`.

11.2.9 okvCustomAttrAddStructure

`okvCustomAttrAddStructure` adds a `TTLV` structure to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddStructure` adds a TTLV structure to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV * okvCustomAttrAddStructure(OKVEnv *env, OKVTTLV *ttlv,
                                    oratext *name, ub4 namel,
                                    OKVTTLV *structure, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.

Parameter	IN/OUT	Description
`ttlv`	`IN`	TTLV parent object.
`name`	`IN`	Name of custom attribute.
`namel`	`IN`	Length of name of custom attribute.
`structure`	`IN`	TTLV structure to be added to the custom attribute.
`attr_index`	`IN`	Attribute index of the structure custom attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with TTLV structure added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
/* Construct the structure to be added */
OKVTTLV *req2 = okvEnvGetOpRequestObj(env);
okvAttrAddUniqueID(env, req2, "Unique_ID354", sizeof("Unique_ID354"));

/* Add the structure */
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddStructure(env, req, "x-My Structure",
                                    strlen("x-My Structure"),
                                    req2, 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetStructure
  `okvCustomAttrGetStructure` returns a `TTLV` structure data type found at element index `elem_ind`.

11.2.10 okvCustomAttrAddTextString

`okvCustomAttrAddTextString` adds a text string data type to the custom attribute specified by the OKVTTLV parent object.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrAddTextString` adds a text string data type to the custom attribute specified by the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvCustomAttrAddTextString(OKVEnv *env, OKVTTLV *ttlv,
                                    oratext *name, ub4 namel,
                                    oratext *text, ub4 textl, ub4 attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`name`	`IN`	Name of custom attribute.
`namel`	`IN`	Length of name of custom attribute.
`text`	`IN`	Text string.
`textl`	`IN`	Length of text string.
`attr_index`	`IN`	Attribute index of the text string custom attribute.

Return Values

Return Value	Description
`OKVTTLV *`	Pointer to the OKVTTLV object. **Success:** Pointer to OKVTTLV object with text string data type added. **Failure:** `NULL`.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
oratext *text = "Text Message";
...
req = okvEnvGetOpRequestObj(env);
attr_in = okvCustomAttrAddTextString(env, req, "x-My Text String", strlen("x-My
Text String"),
                                     text, strlen(text), 0);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvCustomAttrGetTextString
  `okvCustomAttrGetTextString` returns the text string data type found at the element index `elem_ind`.

- okvCustomAttrGetTextStringLen
  `okvCustomAttrGetTextStringLen` returns the length of the text string data type found at the element index `elem_ind`.

11.2.11 okvCustomAttrGet

`okvCustomAttrGet` finds any custom attribute at or after element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGet` finds any custom attribute at or after element index `elem_ind`. The actual element index and length of the name of the custom attribute are also returned.

Syntax

```
OKVErrNo okvCustomAttrGet(OKVEnv *env, OKVTTLV *ttlv,
                          ub4 *elem_ind, OKVType *typ, ub4 *name_len);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN/OUT`	Element index.
`typ`	`IN`	Type of the custom attribute.
`name_len`	`OUT`	Length of the name of the custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Custom Attribute";
ub4 elem_ind = 0;
ub1 typ = 0;
ub4 name_len = 0;
...
```

```
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);

/* We need the type of the custom attribute */
okvCustomAttrGetByName(env, ttlv, &elem_ind, attr_name_list[0],
                       strlen((char *) attr_name_list[0]), &typ);
okvCustomAttrGet(env, ttlv, &elem_ind, &typ, &name_len);
printf ("%d %d", elem_ind, name_len);
```

Related Topics

- okvCustomAttrGetByName
  `okvCustomAttrGetByName` finds custom attribute with name `name` at or after the element index `elem_ind`.
- okvCustomAttrGetByType
  `okvCustomAttrGetByType` finds any custom attribute of type `typ` at or after the element index `elem_ind`, and returns the actual element index and length of the name of the custom attribute.

11.2.12 okvCustomAttrGetBoolean

`okvCustomAttrGetBoolean` fetches a Boolean data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetBoolean` fetches a Boolean data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetBoolean(OKVEnv *env, OKVTTLV *ttlv,
                                 ub4 elem_ind, oratext *name, ub4 *namel,
                                 ub8 *bool_val, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.
`bool_val`	`OUT`	Boolean value in the custom attribute at `elem_ind`.
`attr_index`	`OUT`	Attribute index of the Boolean custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Boolean";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub8 bool_val = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetBoolean(env, ttlv, elem_ind, name,
                        &namel, &bool_val, &attr_index);
printf ("%d", bool_val);
...
free(name);
```

Related Topics

- okvCustomAttrAddBoolean
  `okvCustomAttrAddBoolean` adds a Boolean data type to the custom attribute specified by the OKVTTLV parent object.

11.2.13 okvCustomAttrGetByName

`okvCustomAttrGetByName` finds custom attribute with name `name` at or after the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetByName` finds custom attribute with name `name` at or after element index `elem_ind`. The actual element index and type of the custom attribute are also returned.

Syntax

```
OKVErrNo okvCustomAttrGetByName(OKVEnv *env, OKVTTLV *ttlv,
                                ub4 *elem_ind, oratext *name, ub4 name_len,
                                OKVType *typ);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle
ttlv	IN	TTLV parent object
elem_ind	IN/OUT	Element index
name	IN	Name of custom attribute
name_len	IN	Length of name of custom attribute
typ	OUT	Type of the custom attribute

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Custom Attribute";
ub4 elem_ind = 0;
ub1 typ = 0;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
okvCustomAttrGetByName(env, ttlv, &elem_ind, attr_name_list[0], strlen((char *)
attr_name_list[0]), &typ);
printf ("%d %d", elem_ind, typ);
```

Related Topics

- okvCustomAttrGet
  okvCustomAttrGet finds any custom attribute at or after element index elem_ind.
- okvCustomAttrGetByType
  okvCustomAttrGetByType finds any custom attribute of type typ at or after the element index elem_ind, and returns the actual element index and length of the name of the custom attribute.

11.2.14 okvCustomAttrGetByteString

`okvCustomAttrGetByteString` returns the byte string data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetByteString` returns the byte string data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetByteString(OKVEnv *env, OKVTTLV *ttlv,
                                    ub4 elem_ind, oratext *name, ub4 *namel,
                                    ub1 *byte, ub4 *bytel, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.
`byte`	`OUT`	Byte string.
`bytel`	`IN/OUT`	Length of byte string.
`attr_index`	`OUT`	Attribute index of the byte string custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Memory for the buffer for the byte string data type should be pre-allocated and the size of memory passed in as `bytel`.

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Byte String";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub4 bytel = 16;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
ub1 *byte = (ub1 *) malloc(bytel * (sizeof(ub1)));
memset((void *) byte, 0, bytel);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetByteString(env, ttlv, elem_ind, name,
                           &namel, byte,
                           &bytel, &attr_index);
printf ("%s", byte);
...
free(name);
free(byte);
```

Related Topics

- okvCustomAttrAddByteString
  `okvCustomAttrAddByteString` adds a byte string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrGetByteStringLen
  `okvCustomAttrGetByteStringLen` returns the length of the byte string data type found at the element index `elem_ind`.

11.2.15 okvCustomAttrGetByteStringLen

`okvCustomAttrGetByteStringLen` returns the length of the byte string data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetByteStringLen` returns the length of the byte string data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object.

Syntax

```
ub4 okvCustomAttrGetByteStringLen(OKVEnv *env, OKVTTLV *ttlv,
                                  ub4 elem_ind);
```

Parameters

Parameters	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
ttlv	IN	TTLV parent object.
elem_ind	IN	Element index.

Return Values

Return Value	Description
ub4	Length of the attribute value. **Success:** Length of the byte string data type. **Failure:** 0.

Comments

None.

Example

```
OKVTTLV *attrs = (OKVTTLV *)NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Byte String";
ub4 len = 0, elem_ind = 0;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &attrs);
len = okvCustomAttrGetByteStringLen(env, attrs, elem_ind);
```

Related Topics

- okvCustomAttrAddByteString
  okvCustomAttrAddByteString adds a byte string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrGetByteString
  okvCustomAttrGetByteString returns the byte string data type found at the element index elem_ind.

11.2.16 okvCustomAttrGetByType

okvCustomAttrGetByType finds any custom attribute of type typ at or after the element index elem_ind, and returns the actual element index and length of the name of the custom attribute.

Category

KMIP custom attributes API

Purpose

okvCustomAttrGetByType finds any custom attribute of type typ at or after element index elem_ind. The actual element index and length of the name of the custom attribute are also returned.

Syntax

```
OKVErrNo okvCustomAttrGetByType(OKVEnv *env, OKVTTLV *ttlv,
                                ub4 *elem_ind, OKVType typ, ub4 *name_len);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle
ttlv	IN	TTLV parent object
elem_ind	IN/OUT	Element index
typ	IN	Type of the custom attribute
name_len	OUT	Length of the name of the custom attribute

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Custom Attribute";
ub4 elem_ind = 0;
ub1 typ = 0;
ub4 name_len = 0;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);

/* We need the type of the custom attribute */
okvCustomAttrGetByName(env, ttlv, &elem_ind, attr_name_list[0],
                       strlen((char *) attr_name_list[0]), &typ);
okvCustomAttrGetByType(env, ttlv, &elem_ind, typ, &name_len);
printf ("%d %d", elem_ind, name_len);
```

Related Topics

- okvCustomAttrGet
  okvCustomAttrGet finds any custom attribute at or after element index elem_ind.
- okvCustomAttrGetByName
  okvCustomAttrGetByName finds custom attribute with name name at or after the element index elem_ind.

11.2.17 okvCustomAttrGetDateTime

`okvCustomAttrGetDateTime` returns a date time data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetDateTime` returns a date time data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetDateTime(OKVEnv *env, OKVTTLV *ttlv,
                                  ub4 elem_ind, oratext *name, ub4 *namel,
                                  ub8 *date_time, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.
`date_time`	`OUT`	Date-time value in the custom attribute at element index `elem_ind`.
`attr_index`	`OUT`	Attribute index of the date-time custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Date Time";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub8 date_time = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetDateTime(env, ttlv, elem_ind, name,
                         &namel, &date_time, &attr_index);
printf ("%d", date_time);
...
free(name);
```

Related Topics

- okvCustomAttrAddDateTime
  `okvCustomAttrAddDateTime` adds a date time data type to the custom attribute specified by the OKVTTLV parent object.

11.2.18 okvCustomAttrGetEnum

`okvCustomAttrGetEnum` returns an enumeration data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetEnum` returns an enumeration data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetEnum(OKVEnv *env, OKVTTLV *ttlv,
                              ub4 elem_ind, oratext *name, ub4 *namel,
                              ub4 *enumval, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.

Parameter	IN/OUT	Description
enumval	OUT	enum value in the custom attribute at element index elem_ind.
attr_index	OUT	Attribute index of enumeration custom attribute.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as namel, or both name and namel should be NULL. Just one of them cannot be NULL.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Enumeration";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub4 enumval = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetEnum(env, ttlv, elem_ind, name,
                     &namel, &enumval, &attr_index);
printf ("%d", enumval);
...
free(name);
```

Related Topics

- okvCustomAttrAddEnum
  okvCustomAttrAddEnum adds an enumeration data type to the custom attribute specified by the OKVTTLV parent object.

11.2.19 okvCustomAttrGetInteger

Returns an integer data type found at element index elem_ind in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Catetory

KMIP Custom Attributes API

Purpose

Returns an integer data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetInteger(OKVEnv *env, OKVTTLV *ttlv,
                                 ub4 elem_ind, oratext *name, ub4 *namel,
                                 ub4 *integer, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of the custom attribute.
`namel`	`IN/OUT`	Length of the name of the custom attribute.
`integer`	`OUT`	Integer value in the custom attribute at `elem_ind`.
`attr_index`	`OUT`	Attribute index of the integer custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Integer";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub4 integer = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetInteger(env, ttlv, elem_ind, name,
```

```
                        &namel, &integer, &attr_index);
printf ("%d", integer);
...
free(name);
```

Related Topics

- okvCustomAttrAddInteger
  `okvCustomAttrAddInteger` adds an integer data type to the custom attribute specified by the OKVTTLV parent object.

11.2.20 okvCustomAttrGetInterval

`okvCustomAttrGetInterval` returns an interval data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetInterval` returns an interval data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetInterval(OKVEnv *env, OKVTTLV *ttlv,
                                  ub4 elem_ind, oratext *name, ub4 *namel,
                                  ub4 *interval, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.
`interval`	`OUT`	Interval value in the custom attribute at element index `elem_ind`.
`attr_index`	`OUT`	Attribute index of the Interval custom attribute.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Interval";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub4 interval = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetInterval(env, ttlv, elem_ind, name,
                         &namel, &interval, &attr_index);
printf ("%d", interval);
...
free(name);
```

Related Topics

- okvCustomAttrAddInterval
  `okvCustomAttrAddInterval` adds an interval data type to the custom attribute specified by the OKVTTLV parent object.

11.2.21 okvCustomAttrGetLongInteger

`okvCustomAttrGetLongInteger` returns a long integer data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetLongInteger` returns a long integer data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetLongInteger(OKVEnv *env, OKVTTLV *ttlv,
                                     ub4 elem_ind, oratext *name, ub4 *namel,
                                     ub8 *long_integer, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	TTLV parent object.
`elem_ind`	`IN`	Element index.
`name`	`OUT`	Name of custom attribute.
`namel`	`IN/OUT`	Length of name of custom attribute.
`long_integer`	`OUT`	Long integer value in the custom attribute at element index `elem_ind`.
`attr_index`	`OUT`	Attribute index of the long integer custom attribute.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Long Integer";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub8 long_integer = 0;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetLongInteger(env, ttlv, elem_ind, name,
                            &namel, &long_integer, &attr_index);
printf ("%ld", long_integer);
...
free(name);
```

Related Topics

- okvCustomAttrAddLongInteger
  `okvCustomAttrAddLongInteger` adds a long integer data type to the custom attribute specified by the OKVTTLV parent object.

11.2.22 okvCustomAttrGetStructure

`okvCustomAttrGetStructure` returns a `TTLV` structure data type found at element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetStructure` returns a TTLV structure data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetStructure(OKVEnv *env, OKVTTLV *ttlv,
                                   ub4 elem_ind, oratext *name,
                                   ub4 *namel, OKVTTLV **structure,
                                   ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`elem_ind`	`IN`	Element index
`name`	`OUT`	Name of custom attribute
`namel`	`IN/OUT`	Length of name of custom attribute
`structure`	`OUT`	TTLV structure of the custom attribute
`attr_index`	`OUT`	Attribute index of the structure custom attribute

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be `NULL`. Just one of them cannot be `NULL`.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Structure";
ub4 attr_index = 0;
ub4 elem_ind = 0;
OKVTTLV *structure = (OKVTTLV *) NULL;
ub4 namel = 20;
...

/* Retrieve the structure */
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetStructure(env, ttlv, elem_ind, name,
                          &namel, &structure, &attr_index);

/* Parse the retrieved structure */
ub4 len = 0;
OKVTTLV *cttlv = okvTTLVGetChild(env, structure, 0, (OKVTag *) 0,
                                 (OKVType *) 0, &len);
oratext *cttlv_val = (oratext *)okvTTLVGetValue(cttlv);
printf ("%s", cttlv_val);
...
free(name);
```

Related Topics

- okvCustomAttrAddStructure
  `okvCustomAttrAddStructure` adds a `TTLV` structure to the custom attribute specified by the OKVTTLV parent object.

11.2.23 okvCustomAttrGetTextString

`okvCustomAttrGetTextString` returns the text string data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetTextString` returns the text string data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object along with the attribute index.

Syntax

```
OKVErrNo okvCustomAttrGetTextString(OKVEnv *env, OKVTTLV *ttlv,
                                    ub4 elem_ind, oratext *name,
```

```
                                        ub4 *namel, oratext *text,
                                        ub4 *textl, ub4 *attr_index);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle
ttlv	IN	TTLV parent object
elem_ind	IN	Element index.
name	OUT	Name of the custom attribute.
namel	IN/OUT	Length of the name of the custom attribute.
text	OUT	Text string.
textl	IN/OUT	Length of the text string.
attr_index	OUT	Attribute index of text string custom attribute.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number. **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

Memory for the buffer for the text string data type should be pre-allocated and the size of memory passed in as `textl`.

Either memory for the buffer for the name of the custom attribute should be pre-allocated and the size of memory passed in as `namel`, or both `name` and `namel` should be NULL.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Text String";
ub4 attr_index = 0;
ub4 elem_ind = 0;
ub4 textl = 16;
ub4 namel = 20;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &ttlv);
oratext *text = (oratext *) malloc(textl * (sizeof(oratext)));
memset((void *) text, 0 , textl);
oratext *name = (oratext *) malloc(namel * (sizeof(oratext)));
memset((void *) name, 0, namel);
okvCustomAttrGetTextString(env, ttlv, elem_ind, name,
      &namel, text,
      &textl, &attr_index);
printf ("%s", text);
...
```

```
free(name);
free(text);
```

Related Topics

- okvCustomAttrAddTextString
  `okvCustomAttrAddTextString` adds a text string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrGetTextStringLen
  `okvCustomAttrGetTextStringLen` returns the length of the text string data type found at the element index `elem_ind`.

11.2.24 okvCustomAttrGetTextStringLen

`okvCustomAttrGetTextStringLen` returns the length of the text string data type found at the element index `elem_ind`.

Category

KMIP custom attributes API

Purpose

`okvCustomAttrGetTextStringLen` returns the length of the text string data type found at element index `elem_ind` in the custom attribute specified by the OKVTTLV parent object.

Syntax

```
ub4 okvCustomAttrGetTextStringLen(OKVEnv *env, OKVTTLV *ttlv,
                                  ub4 elem_ind);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ttlv`	`IN`	TTLV parent object
`elem_ind`	`IN`	Element index

Return Values

Return Value	Description
`ub4`	Length of the attribute value. **Success:** Length of the text string data type attribute value. **Failure:** `0`.

Comments

None.

Example

```
OKVTTLV *attrs = (OKVTTLV *)NULL;
oratext *attr_name_list[1];
attr_name_list[0] = "x-My Text String";
ub4 len = 0, elem_ind = 0;
...
okvGetAttributes(env, uid, 1, (oratext **) attr_name_list, &attrs);
len = okvCustomAttrGetTextStringLen(env, attrs, elem_ind);
```

Related Topics

- okvCustomAttrAddTextString
  `okvCustomAttrAddTextString` adds a text string data type to the custom attribute specified by the OKVTTLV parent object.
- okvCustomAttrGetTextString
  `okvCustomAttrGetTextString` returns the text string data type found at the element index `elem_ind`.

12

Oracle Key Vault Extension Operation Management APIs

Oracle Key Vault provides operations used to execute custom KMIP requests.

12.1 About the Oracle Key Vault Client SDK Extension Operation Management APIs

This section describes the interfaces for Oracle Key Vault operations used to execute custom KMIP requests.

12.2 okvOpsCreate

`okvOpsCreate` creates the Oracle Key Vault operation handle that will be used to execute custom KMIP operations.

Category

KMIP extension operation management API

Purpose

`okvOpsCreate` is used to create the Oracle Key Vault operation handle which will be used to execute custom KMIP operations.

Syntax

```
OKVOps *okvOpsCreate(OKVEnv *env, OKVOpsNo ops);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle

Parameter	IN/OUT	Description
ops	IN	KMIP operation

Return Values

Return Value	Description
OKVOps*	Oracle Key Vault operation handle. **Success:** A valid pointer to the Oracle Key Vault operation handle is returned. **Failure:** A NULL pointer is returned.

Comments

Oracle Key Vault KMIP extension APIs can be used to create custom OKVTTLV request packets that can be sent to the Oracle Key Vault server to get a KMIP response packet. An operation with custom OKVTTLV request packet is a custom KMIP operation.

Oracle Key Vault KMIP extention APIs can also parse the KMIP response. Once the operation is executed, Oracle Key Vault operation handle will also hold the OKVTTLV response packet from the Oracle Key Vault server.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
OKVOps *op = okvOpsCreate(env, OKVOpAddAttribute);
req = okvTTLVGetRequest(env, op);
attr_in = okvAddAttributeObject(env, req, OKVAttrName, 0);
okvAttrAddName(env, attr_in, "XYZ", strlen("XYZ"), 1);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvOpsExecuteOp
  okvOpsExecuteOp executes one or more custom KMIP operations.
- okvOpsFree
  okvOpsFree frees the Oracle Key Vault operation handle.

12.3 okvOpsExecuteOp

okvOpsExecuteOp executes one or more custom KMIP operations.

Category

KMIP extension operation management API

Purpose

okvOpsExecuteOp is used to execute one or more custom KMIP operations. The operations are batched and executed.

Syntax

```
OKVErrNo okvOpsExecuteOp(OKVEnv *env, OKVOps **opsr, ub4 ops_cnt);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
opsr	IN	Oracle Key Vault operation handle.
ops_cnt	IN	Count of Oracle Key Vault operation handle.

Return Values

Return Value	Description
OKVErrNo	Oracle Key Vault error number **Success:** OKV_SUCCESS (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

opsr is an array of operations to be batched and executed. ops_cnt is the count of the operations batched.

The error handle will hold the error returned by the Oracle Key Vault server. But this error need not be for all the operations in the operation array. Even if there is an error, the Oracle Key Vault operation handle should be checked for valid response packets.

There is no order to interpret the operation array. The result of the third operation can be processed before that of the first one.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
OKVTTLV *template;
OKVOps *ops[2];
...
/* First Batch Operation */
ops[0]  = okvOpsCreate(env, OKVOpCreate);
req  = okvTTLVGetRequest(env, ops[0]);
okvAttrAddObjectType(env, req, OKVObjSymmetric);
template = okvTTLVAddToObject(env, req, OKVDEF_TAG_TEMPLATE_ATTR_ST,
                              OKVDEF_ITEM_TYPE_STRUCT, (void *) NULL,
                              (ub4) 0);
attr_in = okvAddAttributeObject(env, template, OKVAttrCryptoAlg, 0);
okvAttrAddCryptoAlgo(env, attr_in, (ub4) CRYPTO_ALG_AES);
attr_in = okvAddAttributeObject(env, template, OKVAttrCryptoLen, 0);
okvAttrAddCryptoLen(env, attr_in, (ub4) 128);
attr_in = okvAddAttributeObject(env, template, OKVAttrCryptoUsageMask, 0);
okvAttrAddCryptoUsageMask(env, attr_in, (ub4) 12);
```

```
/* Second Batch Operation */
ops[1]  = okvOpsCreate(env, OKVOpActivate);
req = okvTTLVGetRequest(env, ops[1]);

/* Execute Batch Operation */
okvOpsExecuteOp(env, ops, 2);
```

Related Topics

- okvOpsCreate
  `okvOpsCreate` creates the Oracle Key Vault operation handle that will be used to execute custom KMIP operations.
- okvOpsFree
  `okvOpsFree` frees the Oracle Key Vault operation handle.

12.4 okvOpsFree

`okvOpsFree` frees the Oracle Key Vault operation handle.

Category

KMIP extension operation management API

Purpose

`okvOpsFree` is used to free the Oracle Key Vault operation handle.

Syntax

```
void okvOpsFree(OKVEnv *env, OKVOps **ops);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ops`	`IN`	Oracle Key Vault operation handle

Return Values

None.

Comments

None.

Example

```
OKVOps ops  = okvOpsCreate(env, OKVOpCreate);
...
okvOpsFree(env, &ops);
```

Related Topics

- okvOpsCreate
  `okvOpsCreate` creates the Oracle Key Vault operation handle that will be used to execute custom KMIP operations.
- okvOpsExecuteOp
  `okvOpsExecuteOp` executes one or more custom KMIP operations.

13

Oracle Key Vault Client SDK TTLV Object APIs

The SDK `TTLV` object APIs enable you to perform activities such as getting the child of an `0KVTTLV` object.

- About the Oracle Key Vault Client SDK TTLV Object APIs
  Oracle Key Vault provides interfaces for the Oracle Key Vault KMIP parser and builder, which help build the OKVTTLV objects and help parse them.
- okvTTLVAddToObject
  `okvTTLVAddToObject` creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.
- okvTTLVAddToObjectByTag
  `okvTTLVAddToObjectByTag` creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.
- okvTTLVGetChild
  `okvTTLVGetChild` retrieves a child OKVTTLV object from the OKVTTLV object at the given element index.
- okvTTLVGetChildByTag
  `okvTTLVGetChildByTag` retrieves a child OKVTTLV object from the OKVTTLV object with the specified tag.
- okvTTLVGetChildCount
  `okvTTLVGetChildCount` returns the number of child OKVTTLV objects of the specified OKVTTLV parent object.
- okvTTLVGetChildCountByTag
  `okvTTLVGetChildCountByTag` will return the number of child OKVTTLV objects of the specified OKVTTLV parent object that have the given tag.
- okvTTLVGetFirstChildByTag
  `okvTTLVGetFirstChildByTag` retrieves the first child of the OKVTTLV object that has the specified tag.
- okvTTLVGetLen
  `okvTTLVGetLen` returns the length of the value of the OKVTTLV object.
- okvTTLVGetRequest
  `okvTTLVGetRequest` returns the OKVTTLV request object of the operation.
- okvTTLVGetResponse
  `okvTTLVGetResponse` returns the OKVTTLV response object of the custom KMIP operation.
- okvTTLVGetTag
  `okvTTLVGetTag` returns the tag value of the OKVTTLV object.
- okvTTLVGetType
  `okvTTLVGetType` returns the type value of the OKVTTLV object.

- okvTTLVGetValue
  `okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.
- okvTTLVGetValueCopy
  `okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

13.1 About the Oracle Key Vault Client SDK TTLV Object APIs

Oracle Key Vault provides interfaces for the Oracle Key Vault KMIP parser and builder, which help build the OKVTTLV objects and help parse them.

TTLV stands for tag type length value. The element and attribute indexes used in the Oracle Key Vault SDK TTLV object APIs have same meaning as the Oracle Key Vault KMIP attribute APIs.

Related Topics

- About the Oracle Key Vault KMIP Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.

13.2 okvTTLVAddToObject

`okvTTLVAddToObject` creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVAddToObject` will create an OKVTTLV object and make the newly created OKVTTLV object a child object of the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvTTLVAddToObject(OKVEnv *env, OKVTTLV *ttlv,
                           OKVTag tag, OKVType typ, void *val, ub4 len);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	OKVTTLV parent object.
`tag`	`IN`	Tag of the OKVTTLV object.
`typ`	`IN`	Type of the OKVTTLV object.
`val`	`IN`	Value of the OKVTTLV object.
`len`	`IN`	Length of value of the OKVTTLV object.

Return Values

Return Value	Description
OKVTTLV*	OKVTTLV object. **Success:** A valid pointer to the created OKVTTLV object is returned. **Failure:** A NULL pointer is returned.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *template;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
req  = okvTTLVGetRequest(env, ops);
...
template = okvTTLVAddToObject(env, req, OKVDEF_TAG_TEMPLATE_ATTR_ST,
                              OKVDEF_ITEM_TYPE_STRUCT, (void *) NULL,
                              (ub4) 0);
```

Related Topics

- okvTTLVAddToObjectByTag
  okvTTLVAddToObjectByTag creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.

13.3 okvTTLVAddToObjectByTag

okvTTLVAddToObjectByTag creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.

Category

KMIP extension TTLV object API

Purpose

okvTTLVAddToObjectByTag will create an OKVTTLV object and make the newly created OKVTTLV object a child object of the OKVTTLV parent object.

Syntax

```
OKVTTLV *okvTTLVAddToObjectByTag(OKVEnv *env, OKVTTLV *ttlv,
                                 OKVTag tag, void *val, ub4 len);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle

Parameter	IN/OUT	Description
ttlv	IN	OKVTTLV parent object
tag	IN	Tag of the OKVTTLV object
val	IN	Value of the OKVTTLV object
len	IN	Length of the value of the OKVTTLV object

Return Values

Return Value	Description
OKVTTLV*	OKVTTLV object **Success:** A valid pointer to the created OKVTTLV object is returned. **Failure:** A NULL pointer is returned.

Comments

The difference between okvTTLVAddToObjectByTag and okvTTLVAddToObject is that okvTTLVAddToObjectByTag will try to interpret the type from the tag. This can be done most of the times but not always.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *template;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
req  = okvTTLVGetRequest(env, ops);
...
template = okvTTLVAddToObjectByTag(env, req,
                                   OKVDEF_TAG_TEMPLATE_ATTR_ST,
                                   (void *) NULL, (ub4) 0);
```

Related Topics

- okvTTLVAddToObject
  okvTTLVAddToObject creates an OKVTTLV object and makes this object a child object of the OKVTTLV parent object.

13.4 okvTTLVGetChild

okvTTLVGetChild retrieves a child OKVTTLV object from the OKVTTLV object at the given element index.

Category

KMIP extension TTLV object API

Purpose

okvTTLVGetChild retrieves a child OKVTTLV object from the OKVTTLV object at the given element index. The tag and type of the OKVTTLV object and length of the value of the OKVTTLV object are also returned.

Syntax

```
OKVTTLV *okvTTLVGetChild(OKVEnv *env, OKVTTLV *ttlv,
                         ub4 elem_index,
                         OKVTag *tag, OKVType *typ, ub4 *len);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle
ttlv	IN	OKVTTLV parent object
elem_index	IN	Element index of the child OKVTTLV object
tag	OUT	Tag of the child OKVTTLV object being retrieved
typ	OUT	Type of the OKVTTLV object being retrieved
len	OUT	Length of the value of the OKVTTLV object

Return Values

Return Value	Description
OKVTTLV*	Child OKVTTLV object at element index elem_index. **Success:** A valid pointer to the child OKVTTLV object at element index elem_index is returned. **Failure:** A NULL pointer is returned..

Comments

- If elem_index exceeds the number of children, then NULL is returned.
- The tag of the value of OKVTTLV object is also returned if tag is not NULL.
- The type of the value of OKVTTLV object is also returned if typ is not NULL.
- The length of the value of OKVTTLV object is also returned if len is not NULL.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVTag ctag;
OKVType ctyp;
ub4 clen;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetChild(env, resp, 1, &ctag, &ctyp, &clen);
```

Related Topics

- okvTTLVGetChildByTag
  `okvTTLVGetChildByTag` retrieves a child OKVTTLV object from the OKVTTLV object with the specified tag.
- okvTTLVGetFirstChildByTag
  `okvTTLVGetFirstChildByTag` retrieves the first child of the OKVTTLV object that has the specified tag.

13.5 okvTTLVGetChildByTag

`okvTTLVGetChildByTag` retrieves a child OKVTTLV object from the OKVTTLV object with the specified tag.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetChildByTag` is used to retrieve a child OKVTTLV object from the OKVTTLV object with the specified tag at or after element index `elem_index`. The actual element index is also returned.

Syntax

```
OKVTTLV *okvTTLVGetChildByTag(OKVEnv *env, OKVTTLV *ttlv,
                              OKVTag tag, ub4 *elem_index);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ttlv`	`IN`	OKVTTLV parent object.
`tag`	`IN`	Tag of the OKVTTLV object being retrieved.
`elem_index`	`IN/OUT`	Element index of the child OKVTTLV object.

Return Values

Return Value	Description
`OKVTTLV*`	OKVTTLV object with the given tag. **Success:** A valid pointer to the child OKVTTLV object with the given tag at or after element index `elem_index` is returned. **Failure:** A `NULL` pointer is returned.

Comments

If `elem_index` exceeds the number of children, then `NULL` is returned.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVTag ctag;
OKVType ctyp;
ub4 clen;
ub4 elem_index = 0;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetChildByTag(env, resp, OKVDEF_TAG_ID, &elem_index);
ctag  = okvTTLVGetTag(tagid);
ctyp  = okvTTLVGetType(tagid);
clen  = okvTTLVGetLen(tagid);
```

Related Topics

- okvTTLVGetChild
  `okvTTLVGetChild` retrieves a child OKVTTLV object from the OKVTTLV object at the given element index.
- okvTTLVGetFirstChildByTag
  `okvTTLVGetFirstChildByTag` retrieves the first child of the OKVTTLV object that has the specified tag.

13.6 okvTTLVGetChildCount

`okvTTLVGetChildCount` returns the number of child OKVTTLV objects of the specified OKVTTLV parent object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVVGetChildCount` will return the number of child OKVTTLV objects of the specified OKVTTLV parent object.

Syntax

```
ub4 okvTTLVGetChildCount(OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object

Return Values

Return Value	Description
ub4	Number of child OKVTTLV objects of the specified OKVTTLV object. **Success:** Count of child OKVTTLV objects of the specified OKVTTLV parent object. **Failure:** A zero is returned on error or if the OKVTTLV parent object is not a STRUCTURE.

Comments

If the specified OKVTTLV object is not a STRUCTURE, then zero is returned. If the specified OKVTTLV object is a STRUCTURE and there was an error, zero is returned.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
ub4 child_count;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp = okvTTLVGetResponse(env, ops);
child_count = okvTTLVGetChildCount(resp);
```

Related Topics

- okvTTLVGetChildCountByTag
  okvTTLVGetChildCountByTag will return the number of child OKVTTLV objects of the specified OKVTTLV parent object that have the given tag.

13.7 okvTTLVGetChildCountByTag

okvTTLVGetChildCountByTag will return the number of child OKVTTLV objects of the specified OKVTTLV parent object that have the given tag.

Category

KMIP extension TTLV object API

Purpose

okvTTLVGetChildCountByTag will return the number of child OKVTTLV objects of the specified OKVTTLV parent object that have the given tag.

Syntax

```
ub4 okvTTLVGetChildCountByTag(OKVTTLV *ttlv, OKVTag tag);
```

Parameters

Parameter	IN/OUT	Description
ttlv	IN	OKVTTLV object
tag	IN	Tag of the child OKVTTLV objects being counted

Return Values

Return Value	Description
ub4	Number of child OKVTTLV Objects of the specified OKVTTLV parent object with the given tag. **Success:** Count of the child OKVTTLV objects of the specified OKVTTLV parent object with the given tag. **Failure:** A zero is returned on error or if the OKVTTLV parent object is not a STRUCTURE.

Comments

If the specified OKVTTLV object is not a STRUCTURE, then zero is returned. If the specified OKVTTLV object is a STRUCTURE and there was an error, zero is returned.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
ub4 unique_id_count;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp = okvTTLVGetResponse(env, ops);
unique_id_count = okvTTLVGetChildCountByTag(resp, OKVDEF_TAG_ID);
```

Related Topics

- okvTTLVGetChildCount
  okvTTLVGetChildCount returns the number of child OKVTTLV objects of the specified OKVTTLV parent object.

13.8 okvTTLVGetFirstChildByTag

okvTTLVGetFirstChildByTag retrieves the first child of the OKVTTLV object that has the specified tag.

Category

KMIP extension TTLV object API

Purpose

okvTTLVGetFirstChildByTag is used to retrieve the first child of the OKVTTLV object that has the specified tag.

Syntax

```
OKVTTLV *okvTTLVGetFirstChildByTag(OKVEnv *env,
                                   OKVTTLV *ttlv, OKVTag tag);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle
ttlv	IN	OKVTTLV parent object
tag	IN	Tag of the OKVTTLV object being retrieved

Return Values

Return Value	Description
OKVTTLV*	OKVTTLV object with the given tag. **Success:** A valid pointer to the child OKVTTLV object with the given tag is returned. **Failure:** A NULL pointer is returned.

Comments

None.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVTag ctag;
OKVType ctyp;
ub4 clen;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp = okvTTLVGetResponse(env, ops);
tagid =  okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
ctag = okvTTLVGetTag(tagid);
ctyp = okvTTLVGetType(tagid);
clen = okvTTLVGetLen(tagid);
```

Related Topics

- okvTTLVGetChild
  okvTTLVGetChild retrieves a child OKVTTLV object from the OKVTTLV object at the given element index.
- okvTTLVGetChildByTag
  okvTTLVGetChildByTag retrieves a child OKVTTLV object from the OKVTTLV object with the specified tag.

13.9 okvTTLVGetLen

okvTTLVGetLen returns the length of the value of the OKVTTLV object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetLen` will return the length of the value of the OKVTTLV object.

Syntax

```
ub4 okvTTLVGetLen(OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object

Return Values

Return Value	Description
`ub4`	Length of the OKVTTLV object value **Success:** Length of the value of the OKVTTLV object is returned. **Failure:** A zero is returned if the OKVTTLV object is a `STRUCTURE` or if there was an error.

Comments

If length returned is zero, then it implies an error except when the OKVTTLV type is a `STRUCTURE`.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
ub4 clen;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp = okvTTLVGetResponse(env, ops);
tagid =  okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
clen = okvTTLVGetLen(tagid);
```

Related Topics

- okvTTLVGetTag
  `okvTTLVGetTag` returns the tag value of the OKVTTLV object.
- okvTTLVGetType
  `okvTTLVGetType` returns the type value of the OKVTTLV object.
- okvTTLVGetValue
  `okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.
- okvTTLVGetValueCopy
  `okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

13.10 okvTTLVGetRequest

`okvTTLVGetRequest` returns the OKVTTLV request object of the operation.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetRequest` will return the OKVTTLV request object of the operation. This object will be the root OKVTTLV object used to build the KMIP Request. All OKVTTLV objects used in the KMIP request will fall under the OKVTTLV request object.

Syntax

```
OKVTTLV *okvTTLVGetRequest(OKVEnv *env, OKVOps *ops);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle
`ops`	`IN`	Oracle Key Vault operation handle

Return Values

Return Value	Description
`OKVTTLV*`	OKVTTLV request object for the operation. **Success:** A valid pointer to the OKVTTLV request object is returned. **Failure:** A `NULL` pointer is returned.

Comments

None.

Example

```
OKVTTLV *req = (OKVTTLV *) NULL;
OKVTTLV *attr_in = (OKVTTLV *)NULL;
...
OKVOps *op = okvOpsCreate(env, OKVOpAddAttribute);
req = okvTTLVGetRequest(env, op);
attr_in = okvAddAttributeObject(env, req, OKVAttrName, 0);
okvAttrAddName(env, attr_in, "XYZ", strlen("XYZ"), 1);
okvAddAttribute(env, uid, &attr_in);
```

Related Topics

- okvTTLVGetResponse
  `okvTTLVGetResponse` returns the OKVTTLV response object of the custom KMIP operation.

13.11 okvTTLVGetResponse

`okvTTLVGetResponse` returns the OKVTTLV response object of the custom KMIP operation.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetResponse` will return the OKVTTLV response object of the custom KMIP operation. This object will be used to parse the KMIP response from the Oracle Key Vault server. All OKVTTLV objects of the KMIP response have to be extracted and processed from the OKVTTLV response object.

Syntax

```
OKVTTLV *okvTTLVGetResponse(OKVEnv *env, OKVOps *ops);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`ops`	`IN`	Oracle Key Vault operation handle.

Return Values

Return Value	Description
`OKVTTLV*`	OKVTTLV response object of the operation. **Success:** A valid pointer to the OKVTTLV Response object is returned. **Failure:** A `NULL` pointer is returned.

Comments

None.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVTag ctag, rtag;
OKVType ctyp, rtyp;
oratext cval[OKV_UNIQUE_ID_MAXLEN + 1], rval[OKV_UNIQUE_ID_MAXLEN + 1];
ub4 clen, rlen;
OKVOps *ops[2];
...
/* Get the result of the First Batch Operation */
ops[0] = okvOpsCreate(env, OKVOpCreate);
resp = okvTTLVGetResponse(env, ops[0]);
```

```
/* Retrieve the Unique Identifier */
tagid = okvTTLVGetChild(env, resp, 1, &ctag, &ctyp, &clen);
printf("\n%d %d %d ", ctag, ctyp, clen);
okvTTLVGetValueCopy(tagid, (void *)cval, clen);
cval[clen] = 0;
printf("%s\n", cval); /* Print the Unique Identifier */

/* Get the result of the Second Batch Operation */
ops[1] = okvOpsCreate(env, OKVOpActivate);
resp = okvTTLVGetResponse(env, ops[1]);

/* Retrieve the Unique Identifier */
tagid =  okvTTLVGetChild(env, resp, 0, &rtag, &rtyp, &rlen);
printf("\n%d %d %d ", rtag, rtyp, rlen);
okvTTLVGetValueCopy(tagid, (void *)rval, rlen);
rval[rlen] = 0;
printf("%s\n", rval); /* Print the Unique Identifier */
```

Related Topics

- okvTTLVGetRequest
  `okvTTLVGetRequest` returns the OKVTTLV request object of the operation.

13.12 okvTTLVGetTag

`okvTTLVGetTag` returns the tag value of the OKVTTLV object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetTag` will return the tag value of the OKVTTLV object.

Syntax

```
OKVTag okvTTLVGetTag(OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object

Return Values

Return Value	Description
`OKVTag`	Tag of the OKVTTLV object. **Success:** A valid tag value of the OKVTTLV object is returned. **Failure:** A zero is returned.

Comments

None.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVTag ctag;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
ctag  = okvTTLVGetTag(tagid);
```

Related Topics

- okvTTLVGetType
  `okvTTLVGetType` returns the type value of the OKVTTLV object.
- okvTTLVGetLen
  `okvTTLVGetLen` returns the length of the value of the OKVTTLV object.
- okvTTLVGetValue
  `okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.
- okvTTLVGetValueCopy
  `okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

13.13 okvTTLVGetType

`okvTTLVGetType` returns the type value of the OKVTTLV object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetType` will return the type value of the OKVTTLV object.

Syntax

```
OKVType okvTTLVGetType(OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object

Return Values

Return Value	Description
`OKVType`	Type of the OKVTTLV object. **Success:** A valid Type value of the OKVTTLV object is returned. **Failure:** A zero is returned if there was an error.

Comments

None.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
OKVType ctyp;
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
ctyp  = okvTTLVGetType(tagid);
```

Related Topics

- okvTTLVGetTag
  `okvTTLVGetTag` returns the tag value of the OKVTTLV object.
- okvTTLVGetLen
  `okvTTLVGetLen` returns the length of the value of the OKVTTLV object.
- okvTTLVGetValue
  `okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.
- okvTTLVGetValueCopy
  `okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

13.14 okvTTLVGetValue

`okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.

Syntax

```
void *okvTTLVGetValue(OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object

Return Values

Return Value	Description
void*	OKVTTLV object value. **Success:** Pointer to the value of the OKVTTLV object is returned. **Failure:** A NULL pointer is returned if there is an error.

Comments

None.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
ub4 clen;
oratext *cval;
...
OKVOps *ops = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
clen  = okvTTLVGetLen(tagid);
cval  = (oratext *) okvTTLVGetValue(tagid);
cval[clen] = 0;
printf("%s", cval);
```

Related Topics

- okvTTLVGetTag
  `okvTTLVGetTag` returns the tag value of the OKVTTLV object.
- okvTTLVGetType
  `okvTTLVGetType` returns the type value of the OKVTTLV object.
- okvTTLVGetLen
  `okvTTLVGetLen` returns the length of the value of the OKVTTLV object.
- okvTTLVGetValueCopy
  `okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

13.15 okvTTLVGetValueCopy

`okvTTLVGetValueCopy` copies the value of the OKVTTLV object into the supplied buffer.

Category

KMIP extension TTLV object API

Purpose

`okvTTLVGetValueCopy` will copy the value of the OKVTTLV object into the supplied buffer.

Syntax

```
ub4 okvTTLVGetValueCopy(OKVTTLV *ttlv, void *val, ub4 len);
```

Parameters

Parameter	IN/OUT	Description
`ttlv`	`IN`	OKVTTLV object
`val`	`OUT`	Buffer for the OKVTTLV object value
`len`	`IN`	Length of the OKVTTLV object value

Return Values

Return Value	Description
`ub4`	Length of the OKVTTLV object value. **Success:** Length of the value of the OKVTTLV object is returned. **Failure:** Zero is returned.

Comments

The memory for the value has to be pre-allocated before making a call to this function.

Depending on the type of the OKVTTLV object, the value can be interpreted differently. For example, if the type of the OKVTTLV object is `INTEGER`, then the value can be stored in a ub4 variable. However, if the OKVTTLV object is a `BYTE STRING`, the value is a ub1 array of length `len`.

This function should not be used for OKVTTLV objects that are of type `STRUCTURE`.

Example

```
OKVTTLV *resp = (OKVTTLV *) NULL;
OKVTTLV *tagid = (OKVTTLV *) NULL;
ub4 clen;
oratext cval[OKV_UNIQUE_ID_MAXLEN + 1];
...
OKVOps *ops  = okvOpsCreate(env, OKVOpCreate);
resp  = okvTTLVGetResponse(env, ops);
tagid = okvTTLVGetFirstChildByTag(env, resp, OKVDEF_TAG_ID);
clen  = okvTTLVGetLen(tagid);
okvTTLVGetValueCopy(tagid, (void *)cval, clen);
cval[clen] = 0;
printf("%s", cval);
```

Related Topics

- okvTTLVGetTag
  `okvTTLVGetTag` returns the tag value of the OKVTTLV object.
- okvTTLVGetType
  `okvTTLVGetType` returns the type value of the OKVTTLV object.

- okvTTLVGetLen
  `okvTTLVGetLen` returns the length of the value of the OKVTTLV object.
- okvTTLVGetValue
  `okvTTLVGetValue` returns the pointer to the value of the OKVTTLV object.

14

Oracle Key Vault Client SDK Utility APIs

You can use the Oracle Key Vault client SDK utility APIs with other Oracle Key Vault functions to simplify common operations.

The element and attribute index used in the Oracle Key Vault utility functions have the same meaning as described in section Attribute Index and Element Index.

- About the Oracle Key Vault Client SDK Utility APIs
  You can use the Oracle Key Vault SDK utility APIs with other Oracle Key Vault APIs to simplify commonly used operations.
- okvAttrExtractTTLV
  `okvAttrExtractTTLV` converts the attributes that are child OKVTTLV objects of the specified OKVTTLV parent object into `OKVAttr`.
- okvAttrMakeTTLV
  `okvAttrMakeTTLV` converts the `OKVAttr` into an OKVTTLV structure.
- okvGetTextForAttributeNum
  `okvGetTextForAttributeNum` returns the attribute name for a given Oracle Key Vault attribute number.
- okvGetTextForTag
  `okvGetTextForTag` returns the name of the valid KMIP tag.
- okvGetTextForTagEnum
  `okvGetTextForTagEnum` returns the name of the enumerated value for a valid KMIP tag of `ENUMERATION` type.
- okvGetTextForTagType
  `okvGetTextForTagType` returns the name of the valid KMIP type.
- okvGetTextLenForAttributeNum
  `okvGetTextLenForAttributeNum` returns the length of the name of the attribute for a given Oracle Key Vault attribute number.
- okvObjGetAttrNo
  `okvObjGetAttrNo` will return the Oracle Key Vault attribute number for a given TTLV object.

14.1 About the Oracle Key Vault Client SDK Utility APIs

You can use the Oracle Key Vault SDK utility APIs with other Oracle Key Vault APIs to simplify commonly used operations.

The client SDK utility API element and attribute indexes have the same structure as the Oracle Key Vault KMIP attribute APIs.

Related Topics

- About the Oracle Key Vault KMIP Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.

14.2 okvAttrExtractTTLV

`okvAttrExtractTTLV` converts the attributes that are child OKVTTLV objects of the specified OKVTTLV parent object into `OKVAttr`.

Category

KMIP utility API

Purpose

`okvAttrExtractTTLV` will convert the attributes that are child OKVTTLV objects of the specified OKVTTLV parent object into `OKVAttr`.

Syntax

```
OKVErrNo okvAttrExtractTTLV(OKVEnv *env, OKVTTLV *attr_ttlv,
                            OKVAttr *attr_col);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`attr_ttlv`	`IN`	TTLV parent object containing attributes.
`attr_col`	`OUT`	Collection of attributes.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

None.

Example

```
OKVAttr attr_struct;
ub4 index = 0;
memset((void *)&attr_struct, 0, sizeof(attr_struct));
...
okvAttrExtractTTLV(env, ttlv, &attr_struct);
printf("\nAttributes Extracted:");
printf("\nName: %s ", attr_struct.name[index].name);
printf("\nName Length: %d ", attr_struct.name[index].namel);

/* Null terminate the unique identifier */
attr_struct.unique_identifier.id[attr_struct.unique_identifier.idl] = 0;
```

```
printf("\nUnique ID: %s ", attr_struct.unique_identifier.id);
printf("\nUnique ID Length: %d ", attr_struct.unique_identifier.idl);
printf("\nCryptographic Algorithm: %d ", attr_struct.crypto_algorithm);
```

Related Topics

- okvAttrMakeTTLV
  `okvAttrMakeTTLV` converts the `OKVAttr` into an OKVTTLV structure.

14.3 okvAttrMakeTTLV

`okvAttrMakeTTLV` converts the `OKVAttr` into an OKVTTLV structure.

Category

KMIP utility API

Purpose

`okvAttrMakeTTLV` will convert the `OKVAttr` into an OKVTTLV structure and all the new attributes in the OKVTTLV will be collected under the specified OKVTTLV parent object.

Syntax

```
OKVErrNo okvAttrMakeTTLV(OKVEnv *env, OKVAttr *attr_col
                         OKVTTLV *attr_ttlv);
```

Parameters

Parameter	IN/OUT	Description
`env`	`IN`	Oracle Key Vault environment handle.
`attr_col`	`IN`	Collection of attributes.
`attr_ttlv`	`OUT`	TTLV parent object containing attributes.

Return Values

Return Value	Description
`OKVErrNo`	Oracle Key Vault error number. **Success:** `OKV_SUCCESS` (0) is returned. **Failure:** A valid error number is returned for the error on top of the error stack.

Comments

For multi-instance attributes, the count of the attribute must be set. For example, `name` is a multi-instance attribute. Hence, if `name` is being added, `name_count` must also be specified.

Example

```
OKVTTLV *ttlv = (OKVTTLV *)NULL;
OKVOps *ops;
OKVAttr attrs;
memset((void *)&attrs, 0, sizeof(attrs));
...

/* Adding name attribute to the structure */
ub4 index = 0;
attrs.name_count = 1;
attrs.name[index].name = "attr_name";
attrs.name[index].namel = strlen(attrs.name[index].name);
attrs.name[index].type = 1;

/* Adding cryptographic algo attribute to the structure */
attrs.crypto_algorithm = CRYPTO_ALG_AES;

/* Adding unique identifier attribute to the structure */
attrs.unique_identifier.id = uid;
attrs.unique_identifier.idl = strlen(attrs.unique_identifier.id);
ops = okvOpsCreate(env, OKVOpAddAttribute);
ttlv = okvTTLVGetRequest(env, ops);

/* Convert OKVAttr structure to TTLV Attributes Object */
okvAttrMakeTTLV(env, &attrs, ttlv);
```

Related Topics

- okvAttrExtractTTLV
  `okvAttrExtractTTLV` converts the attributes that are child OKVTTLV objects of the specified OKVTTLV parent object into `OKVAttr`.

14.4 okvGetTextForAttributeNum

`okvGetTextForAttributeNum` returns the attribute name for a given Oracle Key Vault attribute number.

Catetory

KMIP Utility API

Purpose

`okvGetTextForAttributeNum` returns the attribute name for a given Oracle Key Vault attribute number.

Syntax

```
oratext *okvGetTextForAttributeNum(OKVAttrNo attrno);
```

Parameters

Parameter	IN/OUT	Description
`attrno`	`IN`	Oracle Key Vault attribute number of the KMIP attribute.

Return Values

Return Value	Description
`oratext*`	Name of KMIP attribute. **Success:** A valid pointer to the name of the KMIP attribute. **Failure:** A NULL pointer is returned.

Comments

None.

Example

```
printf("%s", okvGetTextForAttributeNum(OKVAttrActivationDate));
```

Related Topics

- okvGetTextForTag
  `okvGetTextForTag` returns the name of the valid KMIP tag.
- okvGetTextForTagEnum
  `okvGetTextForTagEnum` returns the name of the enumerated value for a valid KMIP tag of `ENUMERATION` type.
- okvGetTextForTagType
  `okvGetTextForTagType` returns the name of the valid KMIP type.
- okvGetTextLenForAttributeNum
  `okvGetTextLenForAttributeNum` returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

14.5 okvGetTextForTag

`okvGetTextForTag` returns the name of the valid KMIP tag.

Category

KMIP utility API

Purpose

`okvGetTextForTag` returns the name of the valid KMIP tag. A `NULL` value is returned for invalid KMIP tag.

Syntax

```
oratext *okvGetTextForTag(OKVTag tag);
```

Parameters

Parameter	IN/OUT	Description
`tag`	`IN`	KMIP tag

Return Values

Return Value	Description
`oratext*`	KMIP tag name. **Success:** A valid pointer to the KMIP tag name is returned. **Failure:** A `NULL` pointer is returned.

Comments

None.

Example

```
printf("%s", okvGetTextForTag(OKVDEF_TAG_ID));
```

Related Topics

- okvGetTextForAttributeNum
  `okvGetTextForAttributeNum` returns the attribute name for a given Oracle Key Vault attribute number.
- okvGetTextForTagEnum
  `okvGetTextForTagEnum` returns the name of the enumerated value for a valid KMIP tag of `ENUMERATION` type.
- okvGetTextForTagType
  `okvGetTextForTagType` returns the name of the valid KMIP type.
- okvGetTextLenForAttributeNum
  `okvGetTextLenForAttributeNum` returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

14.6 okvGetTextForTagEnum

`okvGetTextForTagEnum` returns the name of the enumerated value for a valid KMIP tag of `ENUMERATION` type.

Category

KMIP utility API

Purpose

`okvGetTextForTagEnum` will return the name of the enumerated value for a valid KMIP tag of `EMUMERATION` type.

Syntax

```
oratext *okvGetTextForTagEnum(OKVTag tag, ub4 val);
```

Parameters

Parameter	IN/OUT	Description
`tag`	`IN`	KMIP enum tag

Parameter	IN/OUT	Description
val	IN	KMIP enum tag value

Return Values

Return Value	Description
oratext*	Name of KMIP enum tag value. **Success:** A valid pointer to the name of the KMIP enum tag value is returned. **Failure:** A NULL pointer is returned.

Comments

A NULL value is returned for invalid KMIP tag or invalid value of the KMIP tag or if the KMIP tag is not of ENUMERATION type.

Example

```
printf("%s", okvGetTextForTagEnum(OKVDEF_TAG_STATE, OKVDEF_STATE_ACTIVE));
```

Related Topics

- okvGetTextForAttributeNum
  okvGetTextForAttributeNum returns the attribute name for a given Oracle Key Vault attribute number.
- okvGetTextForTag
  okvGetTextForTag returns the name of the valid KMIP tag.
- okvGetTextForTagType
  okvGetTextForTagType returns the name of the valid KMIP type.
- okvGetTextLenForAttributeNum
  okvGetTextLenForAttributeNum returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

14.7 okvGetTextForTagType

okvGetTextForTagType returns the name of the valid KMIP type.

Category

KMIP utility API

Purpose

okvGetTextForTagType returns the name of the valid KMIP type. A NULL value is returned for invalid KMIP type.

Syntax

```
oratext *okvGetTextForTagType(OKVType typ);
```

Parameters

Parameter	IN/OUT	Description
typ	IN	KMIP type

Return Values

Return Value	Description
oratext*	KMIP type name. **Success:** A valid pointer to the KMIP type name is returned. **Failure:** A NULL pointer is returned.

Comments

None.

Example

```
printf("%s", okvGetTextForTagType(OKVDEF_ITEM_TYPE_INT));
```

Related Topics

- okvGetTextForTag
  okvGetTextForTag returns the name of the valid KMIP tag.
- okvGetTextForTagEnum
  okvGetTextForTagEnum returns the name of the enumerated value for a valid KMIP tag of ENUMERATION type.
- okvGetTextForAttributeNum
  okvGetTextForAttributeNum returns the attribute name for a given Oracle Key Vault attribute number.
- okvGetTextLenForAttributeNum
  okvGetTextLenForAttributeNum returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

14.8 okvGetTextLenForAttributeNum

okvGetTextLenForAttributeNum returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

Catetory

KMIP Utility API

Purpose

okvGetTextLenForAttributeNum returns the length of the name of the attribute for a given Oracle Key Vault attribute number.

Syntax

```
ub4 okvGetTextLenForAttributeNum(OKVAttrNo attrno);
```

Parameters

Parameter	IN/OUT	Description
attrno	IN	Oracle Key Vault attribute number of the KMIP attribute.

Return Values

Return Value	Description
ub4	Length of the attribute name. **Success:** Length of the buffer is returned. **Failure:** 0.

Comments

None.

Example

```
printf("%d", okvGetTextLenForAttributeNum(OKVAttrActivationDate));
```

Related Topics

- okvGetTextForAttributeNum
  okvGetTextForAttributeNum returns the attribute name for a given Oracle Key Vault attribute number.
- okvGetTextForTag
  okvGetTextForTag returns the name of the valid KMIP tag.
- okvGetTextForTagEnum
  okvGetTextForTagEnum returns the name of the enumerated value for a valid KMIP tag of ENUMERATION type.
- okvGetTextForTagType
  okvGetTextForTagType returns the name of the valid KMIP type.

14.9 okvObjGetAttrNo

okvObjGetAttrNo will return the Oracle Key Vault attribute number for a given TTLV object.

Catetory

KMIP Utility API

Purpose

okvObjGetAttrNo will return the Oracle Key Vault attribute number for a given TTLV object.

Syntax

```
OKVAttrNo okvObjGetAttrNo(OKVEnv *env, OKVTTLV *ttlv);
```

Parameters

Parameter	IN/OUT	Description
env	IN	Oracle Key Vault environment handle.
ttlv	IN	OKVTTLV object.

Return Values

Return Value	Description
OKVAttrNo	Oracle Key Vault attribute number for the TTLV object. **Success:** Oracle Key Vault attribute number for the given TTLV object is returned. **Failure:** OKVAttrInvalid is returned.

Comments

None.

Example

```
OKVTTLV *ttlv = (OKVTTLV *) NULL;
...
switch (okvObjGetAttrNo(env, ttlv))
{
   ...
   case OKVAttrObjType:
      ...
   case OKVAttrCryptoAlg:
      ...
   case OKVAttrCryptoLen:
      ...
   case OKVAttrCryptoUsageMask:
      ...
}
```

Part III

Oracle Key Vault Client Java SDK API Reference

Part III provides information about the Oracle Key Vault Java SDK packages and its APIs.

- Oracle Key Vault Java SDK Packages
  The Oracle Key Vault Java SDK provides four packages.
- Oracle Key Vault Java SDK APIs
  The Oracle Key Vault Java SDK provides APIs similar to C SDK APIs.

15
Oracle Key Vault Java SDK Packages

The Oracle Key Vault Java SDK provides four packages.

Table 15-1 Oracle Key Vault Java SDK Packages

Package	Description
`oracle.okv.exception`	This package contains all the exceptions thrown by the Java SDK Oracle Key Vault interface.
`oracle.okv.kmip`	This package contains `enum` class for KMIP tags, tag enumerations, and types.
`oracle.okv.response`	This package contains all the response classes whose objects are the return values of various high-level Oracle Key Vault Java SDK APIs.
`oracle.okv.service`	This package contains the `OKVService` class through which all the high level Java SDK APIs are exposed.

- oracle.okv.exception Java Package
  The `oracle.okv.exception` package contains all the exceptions thrown by the Java SDK Oracle Key Vault interface.
- oracle.okv.kmip Java Package
  The `oracle.okv.kmip` package contains `enum` class for KMIP tags, tag enumerations, and types.
- oracle.okv.response Java Package
  The `oracle.okv.response` package contains all the response classes whose objects are the return values of various high-level Oracle Key Vault Java SDK APIs.
- oracle.okv.service Java Package
  The `oracle.okv.service` package contains the `OKVService` class through which all the high level Java SDK APIs are exposed.

15.1 oracle.okv.exception Java Package

The `oracle.okv.exception` package contains all the exceptions thrown by the Java SDK Oracle Key Vault interface.

Exceptions

- OKVAppNamespaceNotSuppException
- OKVAuthNotSuccessfulException
- OKVBatchResponseException
- OKVConfigurationException

- OKVConnectionException
- OKVCryptographicFailureException
- OKVEncodingOptionException
- OKVException
- OKVFeatureNotSuppException
- OKVGeneralFailureException
- OKVIllegalOperationException
- OKVIndexOutOfBoundsException
- OKVInsufficientGrpPrivException
- OKVInvalidArgumentException
- OKVInvalidAttrValueException
- OKVInvalidFieldException
- OKVInvalidGroupException
- OKVInvalidMessageException
- OKVInvalidTTLVException
- OKVItemNotFoundException
- OKVKeyCompressionTypeNotSuppException
- OKVKeyFormatTypeNotSuppException
- OKVMissingDataException
- OKVNoReasonException
- OKVObjectArchivedException
- OKVOperationCancelledException
- OKVOperationNotSuppException
- OKVPermissionDeniedException
- OKVResponseException
- OKVResponseTooLargeException
- OKVRestrictedModeException
- OKVWrongObjectTypeException

15.2 oracle.okv.kmip Java Package

The `oracle.okv.kmip` package contains `enum` class for KMIP tags, tag enumerations, and types.

Interfaces

- OKVPrimitiveConnection

Classes

- OKVConnection
- OKVSSLConnection

- OKVTTLV

Enums

- OKVTag
- OKVTagEnum
- OKVType

15.3 oracle.okv.response Java Package

The `oracle.okv.response` package contains all the response classes whose objects are the return values of various high-level Oracle Key Vault Java SDK APIs.

Interfaces

- OKVAppSpecificInfo
- OKVAttribute
- OKVAttrListResponse
- OKVAttrsResponse
- OKVBatchResponse
- OKVCertResponse
- OKVCryptoDomainParams
- OKVCryptoParams
- OKVDigest
- OKVKeyResponse
- OKVLink
- OKVName
- OKVOpaqueDataResponse
- OKVQueryResponse
- OKVResponse
- OKVRevocationReason
- OKVSecretDataResponse
- OKVTemplateResponse
- OKVUidListResponse
- OKVUidResponse
- OKVUsageLimits
- OKVX509CertId

Classes

- OKVServerInformation

15.4 oracle.okv.service Java Package

The `oracle.okv.service` package contains the `OKVService` class through which all the high level Java SDK APIs are exposed.

Interfaces

- OKVTransporter

Classes

- OKVService
- OKVTransporterImpl

16

Oracle Key Vault Java SDK APIs

The Oracle Key Vault Java SDK provides APIs similar to C SDK APIs.

- Java SDK Management APIs
  This section describes the interfaces for Oracle Key Vault program environment.
- Java SDK Connection Management APIs
  This section describes the interfaces for Oracle Key Vault connection management.
- Java SDK KMIP APIs
  The Java SDK KMIP APIs provide functions for creating keys, activating keys, adding attributes to keys, destroying keys, and other operations.
- Java SDK KMIP Batch APIs
  This section describes the interfaces for the Oracle Key Vault KMIP Batch functions.
- Java SDK KMIP Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.
- Java SDK KMIP Custom Attribute APIs
  This section describes the interfaces that help create and interpret Oracle Key Vault KMIP custom attributes.
- Java SDK KMIP Extension Operation Management APIs
  This section describes the interfaces for Oracle Key Vault operations used to execute custom KMIP requests.
- Java SDK KMIP Extension TTLV Object APIs
  This section describes the interfaces for Oracle Key Vault KMIP parser/builder.

16.1 Java SDK Management APIs

This section describes the interfaces for Oracle Key Vault program environment.

Management APIs

- okvEnvGetOpRequestObj
- okvEnvSetConfig

16.2 Java SDK Connection Management APIs

This section describes the interfaces for Oracle Key Vault connection management.

Connection Management APIs

- okvConnect
- okvConnSendRecvBytes
- okvDisconnect

16.3 Java SDK KMIP APIs

The Java SDK KMIP APIs provide functions for creating keys, activating keys, adding attributes to keys, destroying keys, and other operations.

KMIP APIs

- okvActivate
- okvAddAttribute
- okvCreateKey
- okvDeleteAttribute
- okvDestroy
- okvGetAttributeList
- okvGetAttributes
- okvGetKey
- okvGetOpaqueData
- okvGetSecretData
- okvGetTemplate
- okvLocate
- okvModifyAttribute
- okvQueryCapability
- okvRegKey
- okvRegOpaqueData
- okvRegSecretData
- okvRegTemplate
- okvRekey
- okvRevoke

16.4 Java SDK KMIP Batch APIs

This section describes the interfaces for the Oracle Key Vault KMIP Batch functions.

KMIP Batch APIs

- okvBatchExecute

16.5 Java SDK KMIP Attribute APIs

This section describes the interfaces that help create and interpret Oracle Key Vault KMIP attributes.

KMIP Attribute APIs

- okvAddAttributeObject

- okvAttrAddActivationDate
- okvAttrAddCompromiseDate
- okvAttrAddCompromiseOccurrenceDate
- okvAttrAddContactInfo
- okvAttrAddCryptoAlgo
- okvAttrAddCryptoLen
- okvAttrAddCryptoParams
- okvAttrAddCryptoUsageMask
- okvAttrAddDeactivationDate
- okvAttrAddDestroyDate
- okvAttrAddDigest
- okvAttrAddFresh
- okvAttrAddLeaseTime
- okvAttrAddName
- okvAttrAddObjectGroup
- okvAttrAddObjectType
- okvAttrAddProcessStartDate
- okvAttrAddProtectStopDate
- okvAttrAddRevocationReason
- okvAttrAddUniqueID
- okvAttrAddUsageLimits
- okvAttrGetActivationDate
- okvAttrGetArchiveDate
- okvAttrGetCompromiseDate
- okvAttrGetCompromiseOccurrenceDate
- okvAttrGetContactInfo
- okvAttrGetCryptoAlgo
- okvAttrGetCryptoLen
- okvAttrGetCryptoParams
- okvAttrGetCryptoUsageMask
- okvAttrGetDeactivationDate
- okvAttrGetDestroyDate
- okvAttrGetDigest
- okvAttrGetFresh
- okvAttrGetInitialDate
- okvAttrGetLastChangeDate
- okvAttrGetLeaseTime

- okvAttrGetName
- okvAttrGetObjectGroup
- okvAttrGetObjectType
- okvAttrGetProcessStartDate
- okvAttrGetProtectStopDate
- okvAttrGetRevocationReason
- okvAttrGetState
- okvAttrGetUniqueID
- okvAttrGetUsageLimits
- okvGetAttributeObject

16.6 Java SDK KMIP Custom Attribute APIs

This section describes the interfaces that help create and interpret Oracle Key Vault KMIP custom attributes.

KMIP Custom Attribute APIs

- okvCustomAttrAddBoolean
- okvCustomAttrAddByteString
- okvCustomAttrAddDateTime
- okvCustomAttrAddEnum
- okvCustomAttrAddInteger
- okvCustomAttrAddInterval
- okvCustomAttrAddLongInteger
- okvCustomAttrAddStructure
- okvCustomAttrAddTextString
- okvCustomAttrGetBoolean
- okvCustomAttrGetByName
- okvCustomAttrGetByteString
- okvCustomAttrGetByType
- okvCustomAttrGetDateTime
- okvCustomAttrGetEnum
- okvCustomAttrGetInteger
- okvCustomAttrGetInterval
- okvCustomAttrGetLongInteger
- okvCustomAttrGetStructure
- okvCustomAttrGetTextString

16.7 Java SDK KMIP Extension Operation Management APIs

This section describes the interfaces for Oracle Key Vault operations used to execute custom KMIP requests.

Extension Operation Management APIs

- okvOpsExecuteOp

16.8 Java SDK KMIP Extension TTLV Object APIs

This section describes the interfaces for Oracle Key Vault KMIP parser/builder.

KMIP Extension TTLV Object APIs

- okvTTLVGetChild
- okvTTLVGetChildByTag
- okvTTLVGetFirstChildByTag
- okvTTLVGetLen
- okvTTLVGetTag
- okvTTLVGetType
- okvTTLVGetValue

Part IV

Oracle Key Vault Client SDK Troubleshooting

Part VII describes troubleshooting issues with using the Oracle Key Vault Client SDK.

- Troubleshooting
  Oracle Key Vault SDK programs can be debugged using the tracing infrastructure which is included as a part of Oracle Key Vault SDK deliverable.

17
Troubleshooting

Oracle Key Vault SDK programs can be debugged using the tracing infrastructure which is included as a part of Oracle Key Vault SDK deliverable.

Refer the content below for using the tracing infrastructure for C SDK and Java SDK programs.

- **C SDK Tracing:** Before turning on the tracing, identify the version of C SDK software being used on your system as shown below. If you have deployed C SDK under $OKV_HOME, then execute the following commands:

  ```
  cd $OKV_HOME/csdk/lib
  strings libokvcsdk.so | grep "Oracle Key Vault C SDK Version"
  ```

  The Oracle Key Vault C SDK supports an extensive tracing infrastructure in order to debug errors encountered during execution of a C SDK endpoint program. The tracing can be enabled for a C SDK program using the `okvEnvSetTrace` API. The Oracle Key Vault C SDK supports different trace levels to filter trace messages generated for a C SDK endpoint program. Refer to section okvEnvSetTrace to find information on various trace levels and refer the example in same section to enable tracing for a C SDK endpoint program. The trace files can be found in the location provided as input to `okvEnvSetTrace` API call. The trace files have the format `okv_csdk_<Process_ID_of_CSDK_Program>.trc`.

- **Java SDK Tracing:** Before turning on the tracing, identify the version of Java SDK software being used on your system as shown below. If you have deployed Java SDK under $OKV_HOME, then execute the following commands:

  ```
  cd $OKV_HOME/jsdk/lib
  unzip -p okvjsdk.jar | strings | grep "Oracle Key Vault Java SDK Version"
  ```

  Tracing for the Oracle Key Vault Java SDK is supported at the standard `java.util.logging` trace levels: OFF, SEVERE, WARNING, INFO, FINE, FINER, and ALL. Tracing can be turned on by setting the trace level in configuration file `logging.properties`. This configuration file is provided as part of the Oracle Key Vault Java SDK deliverable. Please make sure to turn on the global trace level `.level` and `java.util.logging.FileHandler.level` in the configuration file so that the Java SDK traces are generated in a file. Also you may change the directory in which you want the trace file to be generated by editing the property `java.util.logging.FileHandler.pattern` in the configuration file.

  For more information on trace levels, refer to https://docs.oracle.com/javase/7/docs/api/java/util/logging/Level.html. For more information on `java.util.logging` FileHandler, refer to https://docs.oracle.com/javase/7/docs/api/java/util/logging/FileHandler.html.

Provide the SDK version information and the trace logs when raising SDK issues to support team. Note that no sensitive information is written to a trace file during program execution.

Related Topics

- okvEnvSetTrace
  `okvEnvSetTrace` sets up the Oracle Key Vault trace configuration context.

Index

A

B

C

D

E

F

J

K

M

N

O

P

Q

R

S

T

U