Release

4.0

CORE® API

Vitech corporation

CORE®: Product and Process Engineering Solutions

Vitech Corporation

2070 Chain Bridge Road, Suite 320

Vienna, Virginia 22182-2536

(703) 883-2270   FAX: (703) 883-1860

Support E-mail: support@vtcorp.com

Web: http://www.vtcorp.com

Table Of Contents

1. INTRODUCTION 1

1.1 FUNCTIONAL OVERVIEW 1

1.1.1 API Server 1

1.1.2 Server Settings 1

1.1.3 API Library 2

2. APPLICATION 2

2.1 CORE API OPERANDS STRUCTURE 2

3. SESSION MANAGEMENT 4

3.1 LOGIN 4

3.2 LOGOUT 5

3.3 GET ENTITY NAME 5

3.4 GET HANDLE AT INDEX 6

3.5 GET VALUE AT INDEX 7

3.6 GET HANDLES 8

3.7 SET TEXT RETURN FORMAT 9

3.8 GET LAST API ERROR 9

3.9 GET LAST SESSION ERROR 10

3.10 RELEASE HANDLES 11

3.11 RELEASE OPERAND NAMED 11

4. PROJECT MANAGEMENT 13

4.1 PROJECT NAME 13

4.2 PROJECTS 14

4.3 SELECT ACTIVE PROJECT 14

4.4 PROJECT RTF HEADER 15

4.5 DATABASE CLASSES 16

4.6 SCHEMA CLASSES 17

5. FOLDER 18

5.1 ALL FOLDER ELEMENTS 18

5.2 ALL PARENT FOLDERS 19

5.3 ALL SUBFOLDERS 20

5.4 FOLDER ELEMENTS 21

5.5 FOLDER NAME 22

5.6 NUMBER OF LOCAL FOLDER ELEMENTS 23

5.7 FOLDER PATH NAME 23

5.8 ROOT FOLDER 24

5.9 SUBFOLDERS 25

5.10 TOTAL FOLDER ELEMENTS 26

6. CLASS 28

6.1 CLASS 28

6.2 ATTRIBUTE DEFINITION PROPERTY 29

6.3 CLASS ATTRIBUTES 29

6.4 CLASS NAME 30

6.5 CLASS PROPERTY 31

.

6.6 CLASS RELATIONS 32
6.7 CLASS TARGET CLASSES 33

7. ELEMENT 35

7.1 ELEMENT ATTRIBUTE VALUE 35
7.2 ELEMENT CHILDREN 36
7.3 ELEMENT CLASS 37
7.4 ELEMENT NAME 38
7.5 ELEMENT NUMBER 39
7.6 ELEMENT PARENTS 40
7.7 ELEMENT RELATIONSHIPS 41
7.8 ELEMENT RELATIONSHIPS TARGET CLASS 42
7.9 ELEMENTS 43
7.10 ELEMENT TARGETS 44
7.11 ELEMENT TARGETS TARGET CLASS 45
7.12 ELEMENT TARGETS TARGET CLASSES 46
7.13 RELATIONSHIPS 47

8. FACILITY 49

8.1 FACILITIES 49
8.2 FACILITY DATABASE CLASSES 49
8.3 FACILITY NAME 50
8.4 FACILITYPROPERTY 51
8.5 FACILITY SCHEMA CLASSES 52

9. RELATION 54

9.1 RELATIONS 54
9.2 RELATION ATTRIBUTES 54
9.3 RELATIONNAME 55
9.4 RELATION PROPERTY 56

10. RELATIONSHIP 58

10.1 COMPLEMENT RELATIVE TO 58
10.2 DEFINITION RELATIVE TO 59
10.3 RELATIONSHIP ATTRIBUTE 60
10.4 TRAVERSE 61

.

1. INTRODUCTION

The CORE API is designed to provide application developers read access to CORE project, schema and database information. The API makes available to the application developer the same database information available via the COREscript language. The API is designed around a message based client-server architecture. The client component is implemented in C++ and provides an ANSI C interface. The server implements a communication layer on top of existing CORE services.

1.1 Functional Overview

1.1.1 API Server

The CORE API server is designed to provide database services to some finite number of client applications. Access to database services through the API is limited to valid CORE users. The server uses the existing security services in CORE to enforce this protocol. Upon completing the login process for a client application, the server establishes a session to process the client's requests for service. This session maintains resources (object references) on behalf of the client application. The resources allocated for the session are reclaimed by the system in the event the client fails to maintain an active session. By an active session, we mean that the client must submit transactions to the server in intervals not less than the session time out period. The server administrator can set the session timeout period to whatever value considered appropriate for his/her environment.

1.1.2 Server Settings

- **Session Timeout**

  The session timeout defines the interval (in minutes) that the server will wait for a request from a client application before it considers the session orphaned. If a client application does not submit a request within the timeout period the server terminates the session and logs a timeout.
- **Maximum Sessions**

  The value of the maximum sessions setting defines the number of concurrent sessions that the server will accept. Note that a client process can have multiple concurrent sessions and that one client application could use up all the available sessions.
- **Maximum Login Attempts**

  The maximum login attempts define the number of failed login attempts that the server will record before login is disabled. The value of this setting is for the server as a whole and in not on a per user basis. The effect of this is that a user may not be able to login after a single failed attempt if that failed attempt brings the total number of failures above the value of this setting.
- **Logging Level**

  The logging level setting determines the parts of a transaction that are logged. At the lowest level, the default, only session startups and termination are logged. At the highest level, every part of a transaction is logged; the contents of the client's request, the server's response and session information.

- **Error Log**
  The error log specifies the file to which server side errors are written
- **Transaction Log**
  The transaction log specifies the file in which server records client transactions.
- **Login Port**

  The login port is the port through which client applications log into the API server. The login port number cannot be changed dynamically. Should there be a need to change the login port, the server must be shutdown and started on a new port number.
- **First Data Port**

  As currently configured, the API server requires a contiguous range of data port numbers. The first data port is the first port number that the server will assign to a client application for submitting data requests.
- **Last Data Port**

  The last data port is the last port number that the server can assign to a client application for submitting data requests.

1.1.3 API Library

The API Library provides an ANSI C interface to the client application and relays the client's request to the server using a messaging protocol with transport provided by TCP sockets. The library provides the client process a facility to establish any number of sessions with the API server. The library manages the sessions for the client. As part of this session management, the library allocates memory for result sets that are returned in response to a client application's requests. The library maintains a list of allocated memory and provides a facility to return the allocated memory to the system. However, it is incumbent on the application developer to use this facility to reduce the memory load of his/her executing application. To establish a session with the API server, a client application submits a login request to the server on a dedicated login channel. The login channel port number may be available from the server administrator or from the API Monitor server settings pane. The result of a successful login request is a session ID to which the client application can connect to submit requests for database services.

2. APPLICATION

2.1 CORE API Operands Structure

The **CoreApiOperands** provides a structure to encapsulate data transfers between the API library and a client application. This structure provides a mechanism through which a client application can make an API call that results in arbitrary size result sets. When used as an output parameter, the client application creates an instance of a CoreApiOperands and passes it as an argument in a function call to the API library. The API library assigns the size of the result set to the *handleCount* member. The API library allocates a block of memory into which it loads the object handles and assigns the base address of the memory block to the *handles* pointer. When

.

used as an input parameter, the client application creates an instance and assigns the address of an array of handles to the handles member and the number of handles to the handleCount member.

```
struct   CoreApiOperands
{
     short       handleCount;
     ULONG       *handles;
};
```

- ***handleCount*** **-** Specifies the number of handles in the handles array**.**
- ***handles*** **-** Pointer to an array of object handles**.**

3. SESSION MANAGEMENT

3.1 Login

The **Login** function creates a session for a client application and returns the handle to the session.

```
VTAPIRSLT Login (
    LPSTR       userName,
    LPSTR       password,
    LPSTR       hostIpAddress,
    UINT        loginPort,
    LPVTSHND    sessionHandle
);
```

Parameters

userName
A CORE account user name.

password
A CORE account user password.

hostIpAddress
The IP address of the machine on which the CORE API Server is running.

loginPort
The port number on which the API server is listening for logins. The login port number can be obtained from the server administrator or from the server settings pane on the API Monitor.

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

Return Values

If the function succeeds the return value is zero, and the <*session handle*> contains the handle to the session created on the server on behalf of the client. A non-zero return code is an indication that the function failed. To get extended error information call the **GetLastApiError** function. This is an application level call and it returns error information on the last application level function call. The following errors apply to this function:

• LOGIN_ACCESS_DENIED	The user does not have access to the CORE database.
• LOGIN_DISABLED	The server has disabled logins.
• API_COMMUNICATION_ERROR	Network error on client machine or server is not running.
• API_CONNECT_FAIL	Network error on client machine or server is not running.

.

Remarks

The function call fails if any of the input parameters is *NULL*. The call also fails if login is disabled on the server, or if all available port numbers are in use.

3.2 Logout

The **Logout** function terminates the session specified by the <*session handle*>. The *Logout* call will release all client side and server side resources maintained for the session.

```
VTAPIRSLT Logout (
        VTSHND sessionHandle
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. Whether the return code of this call indicates success or failure, once the *Logout* call is issued, references to any handles associated with the logged out session is unpredictable. To get extended error information call the **GetLastApiError** function. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The session handle argument is not defined in the session table.
• LOGOUT_FAILURE	The server was unable to complete the logout process.

Remarks

The function call fails if the <*session handle*> in invalid. This would be the case if the client issues the *Logout* call with a handle other than a session handle obtained from the login process. If the function returns with an INVALID_SESSION_HANDLE error code, then the session has not been terminated, and the client can continue to make references to handles retrieved in the context of that session. However, references to API handles after this call is not recommended since it may result in unpredicatible behavior.

3.3 Get Entity Name

The **GetEntityName** function returns the name of a CORE entity specified by the <*entity handle*> argument. The client application has to preallocate a sufficiently large buffer to accommodate the name of the entity. If however the name requires more space than has been allocated, the function will return that part of the name that can fit in the allocated buffer.

```
VTAPIRSLT GetEntityName (
        VTSHND  sessionHandle,
        VTOHND  entityHandle,
        LPSTR   entityNameBuffer,
        SHORT   bufferSize
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

entityHandle

Handle to any CORE entity (element, facility, folder, etc.)

entityNameBuffer

The <*entity name buffer*> argument is a pointer to a C string buffer allocated by the client application to receive the name of the entity. This buffer should be large enough to hold the largest entity name expected otherwise long names are be truncated.

bufferSize

The <*buffer size*> contains the size of the <*entity name buffer*>.

Return Values

If the function succeeds the return value is the number of bytes copied into the <*entity name buffer*>. The function call fails with a return value of zero indicating that no information was copied into the <*entity name buffer*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target entity.
• DEFUNCT_OPERAND	The entity specified by the <*entity handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*entity handle*> is invalid.

3.4 Get Handle At Index

The **GetHandleAtIndex** function returns the handle specified by the <*index*> within the handles array. This is a utility function to provide access to client environments that do not support C style pointers, e.g. Visual Basic. The client application can use this call to retrieve a handle from the collection of handles resulting from other API calls.

```
long GetHandleAtIndex (
        VTSHND          sessionHandle,
        CoreApiOperands *handles,
        SHORT           index
);
```

Parameters

sessionHandle

.

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

handles

A pointer to an array of handles

index

The <*index*> argument is the position of the handle within the handles array.

Return Values

If the function succeeds the return value is the handle of the object at the index position within the handles array.

Remarks

The function fails if either the <*session handle*> or the <*entity handle*> is invalid.

3.5 Get Value at Index

The **GetValueAtIndex** function returns the string representation of a return value specified by the <*index*> within the handles array. This is a utility function to provide access to client environments that do not support C style pointers, e.g. Visual Basic.

```
long GetValueAtIndex (
       VTSHND                sessionHandle,
       CoreApiOperands       *handles,
       SHORT                 index,
       LPSTR                 buffer,
       SHORT                 bufferSize
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

handles

A pointer to an array of handles

index

The <*index*> argument is the position of the value within the return array.

buffer

The <*buffer*> argument is a string large enough to receive the value.

bufferSize

The <*bufferSize*> argument determines the maximum number of characters that will be copied into the buffer.

Return Values

If the function succeeds the return value is the number of bytes copied into the string buffer.

Remarks

The function fails if either the <*session handle*> or < *handles* > is invalid. The function will also fail if the index is outside the range of the result set.

3.6 Get Handles

The **GetHandles** function provides a facility that allows a client application to make API calls resulting in arbitrary size result sets. An application can check the handle count of the CoreApiOperands structure and call the GetHandles function with an array large enough to hold the result set. This function is useful in environments without support for variable size collections.

```
VTAPIRSLT GetHandles (
        VTSHND  sessionHandle,
        CoreApiOperands *coreApiOperands,
        long handles[]
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

coreApiOperands

A pointer to a *CoreApiOperands* structure containing an array of object handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function allocates memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

handles

The handles parameter is an array large enough to hold the number of handles specified by the **coreApiOperands** handle count member.

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

.

Remarks

The function fails if either the <*session handle*> or the <*coreApiOperands*> is invalid.

3.7 Set Text Return Format

The **SetTextReturnFormat** function sets the format in which string result sets are returned. Once set, the format stays in effect until it is changed by another **SetTextReturnFormat** call with a different format.

```
VTAPIRSLT SetTextReturnFormat (
        VTSHND  sessionHandle,
        UINT    textFormat
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client on the server. The server creates a session on behalf of the client upon a successful login and returns the session handle to the client. The session provides the context for the function call. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

textFormat

The text format parameter is an integer in the range <1..5> that specifies the desired format for string results. The following formats are available:

1. ASCII
2. RTF – Includes the body of theRTF without header information. Takes on the project's font and color properties.
3. Fully Qualified RTF – Includes all formatting information required to render the result set as RTF.
4. HTML - Includes the body of theRTF without header information. Takes on the project's font and color properties.
5. Fully Qualified HTML – Includes all formatting information required to render the text as HTML.

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if either the <*session handle*> invalid or the <*textFormat*> is out of range.

3.8 Get Last API Error

The **GetLastApiError** function returns the error description of the last application level API error. This call is used before a session is established; for example, if the login call fails, the client application calls this function to get error information on the login call.

```
VTAPIRSLT GetLastApiError (
        LPSTR   messageBuffer,
```

.

```
        SHORT   bufferSize
);
```

Parameters

messageBuffer

The <*message buffer*> argument is a pointer to a C string buffer allocated by the client application to receive the error description. The error description is truncated if the buffer is not large enough to hold the string.

bufferSize

The <*buffer size*> argument is the size of the < *message buffer*>.

Return Values

The function returns the number of bytes copied into the <*message buffer*>. The function call fails with a return value of zero indicating that no information was copied into the <*message buffer*>.

Remarks

3.9 Get Last Session Error

The **GetLastSessionError** function returns the error description of the last error resulting from a function call in the context of the session specified by the <*session handle*>.

```
VTAPIRSLT GetLastSessionError (
        VTSHND  sessionHandle,
        LPSTR   messageBuffer,
        SHORT   bufferSize
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client on the server. The server creates a session on behalf of the client upon a successful login and returns the session handle to the client. The session provides the context for the function call. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

messageBuffer

The <*message buffer*> argument is a pointer to a C string buffer allocated by the client application to receive the error description. The error description is truncated if the buffer is not large enough to hold the string.

bufferSize

The <*buffer size*> argument is the size of the < *message buffer*>.

Return Values

The function returns the number of bytes copied into the <*message buffer*>. The function call fails with a return value of zero indicating that no information was copied into the <*message buffer*>.

Remarks

The function call fails if the <*session handle*> is invalid.

3.10 Release Handles

The **ReleaseHandles** function frees all resources allocated for the handles in the <*handles* > argument. The memory allocated for the handles in the API library is freed and all object references cached on the server are released.

```
VTAPIRSLT ReleaseHandles (
        VTSHND          sessionHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

handles

A pointer to a *CoreApiOperands* structure containing an array of object handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function allocates memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if the <*session handle*> is invalid.

3.11 Release Operand Named

The **ReleaseOperandNamed** function frees the memory allocated for the operand (a CORE entity) specified by the <*operandName* > argument. This is a local call and servers as a way for the client application to deallocate resources that are no longer needed.

```
VTAPIRSLT ReleaseHandles (
        VTSHND  sessionHandle,
        LPSTR  *operandName
);
```

Parameters

sessionHandle

.

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

operandName

The name of an entity received from any of the API function calls. When a client application makes a call through the API library, the API library caches the return values. The client application may release resources any time with this or the related call **ReleaseHandles.**

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if the <*session handle*> is invalid.

.

4. PROJECT MANAGEMENT

4.1 Project Name

The **ProjectName** function returns the name of the project specified by the <*project handle*> argument.

```
VTAPIRSLT ProjectName (
        VTSHND   sessionHandle,
        VTOHND   projectHandle,
        LPVTRSTR projectName
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

projectHandle

Handle to the project for which the name is desired. Project handles can be obtained through the API **Projects** function.

projectName

The <*project name*> argument is a pointer to an address to which the element name is assigned. The function allocates memory for the name of the element and returns the address to the caller via the <*project name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*project name*> pointer contains the address of the element name. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_PROJECT_HANDLEThe <*element handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> or the <*project handle*> is invalid. The memory for the project name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the project name.

.

4.2 Projects

The **Projects** function returns handles to all the projects in the CORE repository.

```
VTAPIRSLT Projects (
        VTSHND          sessionHandle,
        CoreApiOperands *projecthandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

projectHandles
A pointer to a *CoreApiOperands* structure containing an array of project handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function allocates memory to accommodate the handles returned from the server and assign the base address of the memory to the <*project handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*project handles*> pointer has the address of the structure containing the project handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if either the <*session handle*> or the <*project handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

4.3 Select Active Project

The **SelectActiveProject** function selects the project that becomes the context for all subsequent API function calls in the session specified by <*session handle*>.

```
VTAPIRSLT SelectActiveProject (
        VTSHND  sessionHandle,
        VTOHND  projectHandle
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent

function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

projectHandle

Handle to the CORE project that is to become the active project.

Return Values

If the function succeeds the return value is zero. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if either the <*session handle*> or the <*project handle*> is invalid.

4.4 Project RTF Header

The **ProjectRTFHeader** function returns a string that defines the RTF header for the selected project. The RTF header includes a font and a color table. This includes information on all formatted text in the project. When the text return format is set to RFT (using **SetTextReturnFormat**) all attribute values are returned as RTF strings. The returned strings contain references into the project RTF header font and/or color tables.

```
VTAPIRSLT ProjectRTFHeader (
        VTSHND  sessionHandle,
        VTOHND  projectHandle,
        LPVTRSTR * rtfHeader
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

projectHandle

Handle to the project for which the RTF header is desired. Project handles can be obtained through the **Projects** function.

rtfHeader

The rtfHeader parameter is a pointer to a string with the address of the return RTF header.

Return Values

If the function succeeds the return value is zero, and the < *rtfHeader* > pointer contains the address of the RTF header string. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

.

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_PROJECT_HANDLE The <*project handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> or the <*project handle*> is invalid.

4.5 Database Classes

The **DatabaseClasses** function returns handles to all concrete classes in the schema.

```
VTAPIRSLT DatabaseClasses (
        VTSHND  sessionHandle,
        VTOHND  projectHandle,
        CoreApiOperands *classHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

projectHandle
Handle to the project for which the name is desired. Project handles can be obtained through the API **Projects** function.

classHandles
A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handles*> pointer has the address of the structure containing the class handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_PROJECT_HANDLE The <*project handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> or the <*project handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

4.6 Schema Classes

The **SchemaClasses** function returns handles to all classes in the schema.

```
VTAPIRSLT SchemaClasses (
        VTSHND          sessionHandle,
        CoreApiOperands *classHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandles

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handles*> pointer has the address of the structure containing the class handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.

Remarks

The function fails if the <*session handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

5. FOLDER

Folder (previously known as Category) is a hierarchical grouping of CORE elements within a database class.

5.1 All Folder Elements

The **AllFolderElements** function returns handles to all the elements of the folder specified by the <*folder handle*> argument and all its subfolders.

```
VTAPIRSLT AllFolderElements (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle

Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

handles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handles*> pointer has the address of the structure containing the folder element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.2 All Parent Folders

The **AllParentFolders** function returns handles to all folder elements above (ancestors of) the folder specified by the <*folder handle*> argument. The return collection is ordered hierarchically starting at the root folder.

```
VTAPIRSLT AllParentFolders (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle
: Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle
: Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

handles
: A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handles*> pointer has the address of the structure containing the folder element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.3 All Subfolders

The **AllSubfolders** function returns handles to all folder elements below the folder specified by the <*folder handle*> argument. The return collection is ordered according to the depth of the folder element in the hierarchy. Elements at the same level are ordered alphabetically.

```
VTAPIRSLT AllSubfolders (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle
Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

handles
A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handles*> pointer has the address of the structure containing the folder element handles. A non-zero return code is an indication that the function failed. To get extended error information call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.4 Folder Elements

The **FolderElements** function returns handles to the elements of the folder specified by the <*folder handle*> argument.

```
VTAPIRSLT FolderElements (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle
: Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle
: Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

handles
: A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handles*> pointer has the address of the structure containing the folder element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.5 Folder Name

The **FolderName** function returns the name of the folder element specified by the <*folder handle*> argument.

```
VTAPIRSLT FolderName (
        VTSHND   sessionHandle,
        VTOHND   folderHandle,
        LPVTRSTR folderName
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle
Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

folderName
The <*folder name*> argument is a pointer to the address to which the folder element name is assigned. The function allocates memory for the name of the folder element and returns the address to the caller via the <*folder name*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*folder name*> pointer contains the address of the folder element name. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FOLDER_HANDLE	The <*folder handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The memory for the folder element name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform

.

the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the folder element name.

5.6 Number of Local Folder Elements

The **NumberOfLocalFolderElements** function returns the number of elements in the folder specified by the <*folder handle*> argument.

```
VTAPIRSLT NumberOfLocalFolderElements (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        USHORT  *numberOfElements
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle
Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

numberOfElements
The parameter to which the resulting number of elements is assigned.

Return Values

If the function succeeds the return value is zero, and the <*numberOfElements*>. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FOLDER_HANDLE	The <*folder handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid.

5.7 Folder Path Name

The **FolderPathName** function returns the path name of the folder element specified by the <*folder handle*> argument.

```
VTAPIRSLT FolderPathName (
        VTSHND  sessionHandle,
```

.

```
        VTOHND  folderHandle,
        LPSTR   pathName
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle

Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

pathName

The <*path name*> argument is a pointer to an address to which the path name is assigned. The function allocates memory for path name and returns the address to the caller via the <*path name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*path Name*> pointer has the address of the string containing the folder path name. The path name is a string delimited with a single slash of the names of all the ancestors of the folder element specified by the <folder handle>. The names are concaternated in hierarchical order starting at the root folder. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FOLDER_HANDLE	The <*folder handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.8 Root Folder

The **RootFolder** function returns the handle to the folder root element specified by the <*class handle*> argument.

```
VTAPIRSLT RootFolder (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        CoreApiOperands *handle
```

.

```
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandle

Handle to the class that is the target of the request.

handle

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handle*> pointer has the address of the structure containing the folder element handle. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_CLASS_HANDLE	The <*class handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The folder specified by the <*class handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.9 Subfolders

The **Subfolders** function returns handles to the folder elements directly below the folder specified by the <*folder handle*> argument.

```
VTAPIRSLT Subfolders (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle

Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

handles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*handles*> pointer has the address of the structure containing the folder element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*folder handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

5.10 Total Folder Elements

The **TotalFolderElements** function returns the total number of elements in the hierarchy of the folder specified by the <*folder handle*> argument.

```
VTAPIRSLT TotalFolderElements (
        VTSHND  sessionHandle,
        VTOHND  folderHandle,
        USHORT  *numberOfElements
);
```

Parameters

sessionHandle

.

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

folderHandle

Handle to the folder object that is the target of the request. Folder handles can be obtained through the **RootFolder** function.

numberOfElements

A pointer to an unsigned short; receive the total number of elements in the result set.

Return Values

If the function succeeds the return value is zero, and the <*numberOfElements*>. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_FOLDER_HANDLE The <*folder handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The folder specified by the <*folder handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the *folder handle* is invalid.

6. CLASS

6.1 Class

The **Class** function returns the handle to the class specified by <*class Name*>.

```
VTAPIRSLT Class (
        VTSHND  sessionHandle,
        LPSTR   className,
        CoreApiOperands *classHandle
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

className

Name of the class for which the element definition is being requested.

classHandle

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class* handle> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handle*> pointer has the address of the structure containing the class handle. Note that the function returns a single class handle and that the *CoreApiOperands* will contain a single value in the handles array. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• UNDEFINED_ELEMENT	There is no class defined by the name in the <*class name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*class name*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

6.2 Attribute Definition Property

The **AttributeDefinitionProperty** function returns handles to all attribute definitions for the attribute specified by the <*attribute handle*> argument and the property specified by the <*property Name*> argument.

```
VTAPIRSLT AttributeDefinitionProperty (
        VTSHND  sessionHandle,
        VTOHND  attributeHandle,
        LPSTR   propertyName,
        CoreApiOperands *handles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

attributeHandle

A pointer to a *CoreApiOperands* structure containing an array of attribute handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <h*andles*> pointer.

propertyName

The name of the property for which the value is desired.

handles

A pointer to a *CoreApiOperands* structure containing an array of object handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function allocates memory to accommodate the handles returned from the server and assign the base address of the memory to the <*handles*> pointer.

6.3 Class Attributes

The **ClassAttributes** function returns handles to all attribute definitions for the class specified by the <*class handle*> argument.

```
VTAPIRSLT ClassAttributes (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        CoreApiOperands *attributeHandles
);
```

Parameters

sessionHandle

.

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandle

Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

attributeHandles

A pointer to a *CoreApiOperands* structure containing an array of attribute handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*attribute* handles> pointer.

Return Values

If the function succeeds the return value is zero, and the <*attribute Handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_CLASS_HANDLE	The <*class handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> *or the* <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

6.4 Class Name

The **ClassName** function returns handles to all attribute definitions for the class specified by the <*class handle*> argument.

```
VTAPIRSLT ClassName (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        LPVTRSTR className
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests

.

submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandle
: Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

claseName
: The <*class name*> argument is a pointer to an address to which the class name is assigned. The function allocates memory for the name of the class and returns the address to the caller via the <*class name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*class name*> pointer contains the address of the class name. A non-zero return code is an indication that the call failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

•	INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
•	INVALID_CLASS_HANDLE	The <*class handle*> argument is undefined or is NULL.
•	INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
•	DEFUNCT_OPERAND	The class specified by the <*class handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> *or the* <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

6.5 Class Property

The **ClassProperty** function returns the handle to the property specified by the <*property name*> of the class specified by the <*class handle*> argument.

```
VTAPIRSLT ClassProperty (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        LPSTR   propertyName,
        CoreApiOperands *propertyHandle
);
```

Parameters

sessionHandle
: Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

.

classHandle

Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

propertyName

The name of the property for which the value is desired.

attributeHandles

A pointer to a *CoreApiOperands* structure containing an array of property handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*property handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*property handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_CLASS_HANDLE The <*class handle*> argument is undefined or is NULL.
- UNDEFINED_ELEMENT There is no property defined by the name in the <*property name*> argument.

Remarks

The function fails if either the <*session handle> or the <class handle*> is invalid or the property name is not defined for the specified class. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

6.6 Class Relations

The **ClassRealtions** function returns handles to all the relations of class specified by the <*class handle*> argument.

```
VTAPIRSLT ClassRelations (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        CoreApiOperands *relationHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

.

classHandle
Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

relationHandles
A pointer to a *CoreApiOperands* structure containing an array of relation handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relation handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relation Handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_CLASS_HANDLE	The <*class handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> or the <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.
Class Target Classes

6.7 Class Target Classes

The **ClassTargetClasses** function returns handles to all classes defined in the CORE schema as targets of the relation specified by the <*relation handle*> argument.

```
VTAPIRSLT ClassTargetClasses (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        VTOHND  relationHandle,
        CoreApiOperands *classHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandle
Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

relationHandle

A valid relation of the class specified by the <*class handle*> argument.

classHandles

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*class handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target class.
• DEFUNCT_OPERAND	The class specified by the <*class handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

Remarks

The function fails if either the <*session handle*>, <*class handle*> or <*relation handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

7. ELEMENT

7.1 Element Attribute Value

The **ElementAttributeValue** function returns handles to all attribute values for the attribute specified by the <*attribute name*> argument, and are defined for the element specified by <*element handle*>.

```
VTAPIRSLT ElementAttributeValue (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   attributeName,
        CoreApiOperands *attributeHandles
);
```

Parameters

sessionHandle
: Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
: Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

attributeName
: The name of the attribute for which values are desired, e.g., “description.”

attributeHandles
: A pointer to a *CoreApiOperands* structure containing an array of attribute handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*attribute handles*> pointer.

Return Values

If the function succeeds, the return value is zero and the <*attribute handles*> pointer has the address of the structure containing the attribute value handles. The result set of this function is dependent on the type of the attribute. If the attribute is of type collection then the result set is a collection; if the attribute type is text then the result is of type string. See the COREscript Expression Language Reference Guide for additional information on other types of results. A non-zero return code is an indication that the call failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.

• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no attribute defined by the name in the <*attribute name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed.

7.2 Element Children

The **ElementChildren** function returns handles to all the elements that are targets of child-parent relationships defined on the element specified by <*element Handle*>.

```
VTAPIRSLT ElementChildren (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        CoreApiOperands *childrenHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle

Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

childrenHandles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*children handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*children handles*> pointer has the address of the structure containing the element handles. The returned children handles may not all be of the same class as the element against which the call was issued. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.

.

• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.3 Element Class

The **ElementClass** function returns the handle to the (class) element definition for the element specified by <*element handle*>.

```
VTAPIRSLT ElementClasss (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        CoreApiOperands *classHandle
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle

Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

classHandle

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handle*> pointer has the address of the structure containing the class handle. Note that the function returns a single class handle and that the *CoreApiOperands* will contain a single value in the handles array. A non-zero return code is an indication that the call failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.

.

• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.4 Element Name

The **ElementName** function returns the name of the element specified by the <*element handle*> argument.

```
VTAPIRSLT ElementName (
        VTSHND   sessionHandle,
        VTOHND   elementHandle,
        LPVTRSTR elementName
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project; requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

elementName
The <*element name*> argument is a pointer to an address to which the element name is assigned. The function allocates memory for the name of the element and returns the address to the caller via the <*element name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*element name*> pointer contains the address of the element name. A non-zero return code is an indication that the call failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The memory for the element name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the element name.

7.5 Element Number

The **ElementNumber** function returns the hierarchical number of the element specified by the <*element handle*> argument.

```
VTAPIRSLT ElementNumber (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        USHORT  *hierarchyNumber
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

hierarchyNumber
The parameter to which the hierarchical element number is assigned.

Return Values

If the function succeeds the return value is zero, and the <*hierarchy number*>. A non-zero return code is an indication that the call failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid.

.

7.6 Element Parents

The **ElementParents** function returns handles to all the elements that are targets of child-parent relationships defined on the element specified by <*element handle*>.

```
VTAPIRSLT ElementParents (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        CoreApiOperands *parentHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle

Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

parentHandles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*parent handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*parent handles*> pointer has the address of the structure containing the parent handles. The returned parent handles may not all be of the same class as the element against which the call was issued. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

7.7 Element Relationships

The **ElementRealtionships** function returns handles to all the relationships of the relation specified by the <*relation name*> argument that have been defined for the element <*element handle*>.

```
VTAPIRSLT ElementRelationships (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        CoreApiOperands *relationshipHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle

Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName

The name of a valid relation for the element specified by the <*element handle*> argument.

relationshipHandles

A pointer to a *CoreApiOperands* structure containing an array of relationship handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relationship handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relationship handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.8 Element Relationships Target Class

The **ElementRelationshipsTargetClass** function returns handles to all the relationship objects specified by the <*relation name*> that are defined for the element <*element handle*> as the source and an element of the class specified by <*target class handle*>.

```
VTAPIRSLT ElementRelationshipsTargetClass (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        VTOHND  targetClassHandle,
        CoreApiOperands *relationshipHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName
The name of a valid relation for the element specified by the <*element handle*> argument.

targetClassHandle
A valid target class defined for the relation specified by the <*relation name*> argument.

relationshipHandles
A pointer to a *CoreApiOperands* structure containing an array of relationship handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relationship handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relationship handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.

.

• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INVALID_CLASS_HANDLE	The <*target class handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target relationship.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.9 Elements

The **Elements** function returns handles to the elements specified by the <*class handle*> argument.

```
VTAPIRSLT Elements (
        VTSHND  sessionHandle,
        VTOHND  classHandle,
        CoreApiOperands *elementHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

classHandle

Handle to the class that is the target of the request. Class handles can be obtained through any of the following functions: **Class**, **DatabaseClasses**, or **FacilityDatabaseClasses**.

elementHandles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*element handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*element handles*> pointer has the address of the structure containing the element handles. A non-zero return code is an indication

.

that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
INVALID_CLASS_HANDLE	The <*class handle*> argument is undefined or is NULL.
INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.

Remarks

The function fails if either the <*session handle*> or the <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.10 Element Targets

The **ElementTargets** function returns handles to all the elements that are targets of the relationship specified by the <*relation name*> argument.

```
VTAPIRSLT ElementTargets (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        CoreApiOperands *elementHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName
The name of a valid relation for the element specified by the <*element handle*> argument.

elementHandles
A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*element handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*element handles*> pointer has the address of the structure containing the element handles. A non-zero return code is an indication

that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.11 Element Targets Target Class

The **ElementTargetsTargetClass** function returns handles to all the elements that are of the class specified by the <*target class handle*> and are the targets of the relationship specified by the <*relation name*> argument.

```
VTAPIRSLT ElementTargetsTargetClass (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        VTOHND  targetClassHandle,
        CoreApiOperands *elementHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName
The name of a valid relation for the element specified by the <*element handle*> argument.

targetClassHandle
A valid target class defined for the relation specified by the <*relation name*> argument.

elementHandles
A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to

accommodate the handles returned from the server and assign the base address of the memory to the <*element handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*element handles*> pointer has the address of the structure containing the element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INVALID_CLASS_HANDLE	The <*target class handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.12 Element Targets Target Classes

The **ElementTargetsTargetClasses** function returns handles to all the elements that are of the class specified by the <*target class handles*> and are the targets of the relationship specified by the <*relation name*> argument.

```
VTAPIRSLT ElementTargetsTargetClass (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        CoreApiOperands *targetClassHandles,
        CoreApiOperands *elementHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName

.

The name of a valid relation for the element specified by the <*element handle*> argument.

targetClassHandles

An array of valid target class handles defined for the relation specified by the <*relation name*> argument.

elementHandles

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*element handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*element handles*> pointer has the address of the structure containing the element handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INVALID_CLASS_HANDLE	The <*target class handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

Remarks

The function fails if either the <*session handle*> or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

7.13 Relationships

The **Relationships** function returns handles to all the relationships in the CORE database that are of the relation specified by the <*relation name*> argument and have the element <*element handle*> as their source.

```
VTAPIRSLT Relationships (
        VTSHND  sessionHandle,
        VTOHND  elementHandle,
        LPSTR   relationName,
        CoreApiOperands *relationshipHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent

.

function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

elementHandle

Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationName

The name of the relation for which a value is desired.

relationshipHandles

A pointer to a *CoreApiOperands* structure containing an array of relationship handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relationship handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relationship handles*> pointer has the address of the structure containing the property handle. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>.

Remarks

The function fails if either the <*session handle*> *or the* <*element handle*> is invalid or the relation name is not defined for the specified element. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no relation defined by the name in the <*relation name*> argument.

.

8. FACILITY

8.1 Facilities

The **Facilities** function returns handles to all the facilities in the CORE schema.

```
VTAPIRSLT Facilities (
        VTSHND          sessionHandle,
        CoreApiOperands *facilityHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

facilityHandles

A pointer to a *CoreApiOperands* structure containing an array of facility handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*facility handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*facility handles*> pointer has the address of the structure containing the facility handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.

Remarks

The function fails if either the <*session handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

8.2 Facility Database Classes

The **FacilityDatabaseClasses** function returns handles to all the classes in the specified facility.

```
VTAPIRSLT FacilityDatabaseClasses (
        VTSHND  sessionHandle,
        VTOHND  facilityHandle,
        CoreApiOperands *classHandles
);
```

.

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

facilityHandle

Handle to a facility in the CORE schema against which function calls are issued. Facility handles can be obtained through a call the **Facilities** function.

classHandles

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handles*> pointer has the address of the structure containing the class handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FACILITY_HANDLE	The <*facility handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.

Remarks

The function fails if either the <*session handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

8.3 Facility Name

The **FacilityName** function returns the name of the facility specified by the <*facility handle*> argument.

```
VTAPIRSLT FacilityName (
        VTSHND  sessionHandle,
        VTOHND  facilityHandle,
        LPVTRSTR facilityName
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

facilityHandle

Handle to a facility in the CORE schema against which function calls are issued. Facility handles can be obtained through a call the **Facilities** function.

facilityName

The <*facility name*> argument is a pointer to an address to which the facility name is assigned. The function allocates memory for the name of the element and returns the address to the caller via the <*facility name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*facility name*> pointer contains the address of the facility name. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FACILITY_HANDLE	The <*facility handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target facility.
• DEFUNCT_OPERAND	The element specified by the <*facility handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*facility handle*> is invalid. The memory for the element name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the facility name.

8.4 FacilityProperty

The **FacilityProperty** function returns a collection of property handles for the facility specified by specified by the <*facility handle*> and <*propertyName*> argument.

```
VTAPIRSLT FacilityProperty (
        VTSHND  sessionHandle,
        VTOHND  facilityHandle,
        LPSTR propertyName,
        CoreApiOperands *propertyHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

facilityHandle

Handle to a facility in the CORE schema against which function calls are issued. Facility handles can be obtained through a call the **Facilities** function.

propertyName

The <*property name*> argument is the name of the property.

propertyHandles

A pointer to a *CoreApiOperands* structure containing an array of property handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*property handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*property handles*> pointer contains the address of the properties. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FACILITY_HANDLE	The <*facility handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target facility.
• DEFUNCT_OPERAND	The element specified by the <*facility handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*facility handle*> is invalid. The memory for the element name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the facility name.

8.5 Facility Schema Classes

The **FacilitySchemaClasses** function returns handles to all abstract classes defined in the CORE schema as belonging to the facility specified by the <*facility handle*> argument.

```
VTAPIRSLT FacilitySchemaClasses (
        VTSHND  sessionHandle,
        VTOHND  facilityHandle,
        CoreApiOperands *classHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

facilityHandle

Handle to a facility in the CORE schema against which function calls are issued. Facility handles can be obtained through a call the **Facilities** function.

classHandles

A pointer to a *CoreApiOperands* structure containing an array of class handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*class handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*class handles*> pointer has the address of the structure containing the facility handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_FACILITY_HANDLE	The <*facility handle*> argument is undefined or is NULL.

Remarks

The function fails if either the <*session handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

9. RELATION

9.1 Relations

The **Relations** function returns handles to all relations in the CORE schema.

```
VTAPIRSLT Relations (
        VTSHND  sessionHandle,
        CoreApiOperands *relationHandles
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationHandles

A pointer to a *CoreApiOperands* structure containing an array of relation handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relation handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relation handles*> pointer has the address of the structure containing the facility handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.

Remarks

The function fails if either the <*session handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

9.2 Relation Attributes

The **RelationAttributes** function returns handles to all attribute definitions of the relation specified by the <*relation handle*> argument.

```
VTAPIRSLT RelationAttributes (
        VTSHND  sessionHandle,
        VTOHND  relationHandle,
        CoreApiOperands *attributeHandles
);
```

.

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationHandle

A handle to a relation in the CORE schema for which an attribute definition is desired. Relation handles can be obtained through a call to the **Relations** function.

attributeHandles

A pointer to a *CoreApiOperands* structure containing an array of attribute handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*attribute handles*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*attribute handles*> pointer has the address of the structure containing the attribute handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_RELATION_HANDLE	The <*relation handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.

Remarks

The function fails if either the <*session handle*> *or the* <*class handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

9.3 RelationName

The **RelationName** function returns the name of the relation specified by the <*relation handle*> argument.

```
VTAPIRSLT RelationName (
        VTSHND  sessionHandle,
        VTOHND  relationHandle,
        LPVTRSTR relationName
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationHandle

Handle to the relation whose name is desired.

relationName

The <*relation name*> argument is a pointer to an address to which the relation name is assigned. The function allocates memory for the name of the relation and returns the address to the caller via the <*relation name*>pointer.

Return Values

If the function succeeds the return value is zero, and the <*relation name*> pointer contains the address of the relation name. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_RELATION_HANDLE	The <*relation handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target facility.
• DEFUNCT_OPERAND	The element specified by the <*relation handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> or the <*relation handle*> is invalid. The memory for the relation name is allocated from the heap. The client can release that memory with a call to **ReleaseHandles** when the handles are no longer needed**.** If the client wishes to perform the memory management then the client can call the related function **GetEntityName** with a buffer large enough to accommodate the relation name.

9.4 Relation Property

The **RelationProperty** function returns the handle to the property specified by the <*property name*> for the relation specified by the <*relation handle*> argument.

```
VTAPIRSLT RelationProperty (
        VTSHND  sessionHandle,
        VTOHND  relationHandle,
        LPSTR   propertyName,
        CoreApiOperands *propertyHandle
);
```

Parameters

sessionHandle

Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationHandle

A handle to a relation in the CORE schema for which an attribute definition is desired. Relation handles can be obtained through a call to the **Relations** function.

propertyName

The name of the property for which a value is desired.

propertyHandles

A pointer to a *CoreApiOperands* structure containing an array of property handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*property handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*property handle*> pointer has the address of the structure containing the property handle. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_RELATION_HANDLE	The <*relation handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target property.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.
• UNDEFINED_ELEMENT	There is no property defined by the name in the <*property name*> argument.

Remarks

The function fails if either the <*session handle*> *or the* <*relation handle*> is invalid or the property name is not defined for the specified class. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.

10. RELATIONSHIP

10.1 Complement Relative To

The **ComplementRelativeTo** function returns the handle to the complement relation definition for the relationship specified by the <*relationship handle*> where <*element handle*> is the subject element.

```
VTAPIRSLT ComplementRelativeTo (
        VTSHND  sessionHandle,
        VTOHND  relationshipHandle,
        VTOHND  elementHandle,
        CoreApiOperands *relationHandle
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationshipHandle
Handle to the relationship definition for which the complement relative is desired. A relationship handle can be obtained through a call to the **Relationships** function with a relationship name argument.

elementHandle
Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationHandle
A pointer to a *CoreApiOperands* structure containing an array of relation handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relation handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relation handle*> pointer has the address of the structure containing the property handle. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_RELATIONSHIP_HANDLE The <*relationship handle*> argument is undefined or is NULL.
- INVALID_ELEMENT_HANDLE The <*element handle*> argument is undefined or is NULL.

.

• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*>, <*relationship handle*>, or the <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

10.2 Definition Relative To

The **DefinitionRelativeTo** function returns the handle to the relation definition for the relationship specified by the <*relationship handle*> having the direction specified by the <*element handle*> argument.

```
VTAPIRSLT DefinitionRelativeTo (
        VTSHND  sessionHandle,
        VTOHND  relationshipHandle,
        VTOHND  elementHandle,
        CoreApiOperands *relationHandle
);
```

Parameters

sessionHandle
: Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationshipHandle
: Handle to the relationship definition for which the definition relative is desired. A relationship handle can be obtained through a call to the **Relationships** function with a relationship name argument.

elementHandle
: Handle to the element against which the function is executed. Element handles can be obtained through a call to the **Elements** function.

relationHandle
: A pointer to a *CoreApiOperands* structure containing an array of relation handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*relation handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*relation handle*> pointer has the address of the structure containing the property handle. A non-zero return code is an indication

.

that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

• INVALID_SESSION_HANDLE	The <*session handle*> specified does not map to a valid session.
• INVALID_RELATIONSHIP_HANDLE	The <*relationship handle*> argument is undefined or is NULL.
• INVALID_ELEMENT_HANDLE	The <*element handle*> argument is undefined or is NULL.
• INSUFFICIENT_PRIVILEGE	The user does not have read permission on a target element.
• DEFUNCT_OPERAND	The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*>, <*relationship handle*>, *or the* <*element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

10.3 Relationship Attribute

The **RelationshipAttribute** function returns the attribute handles for the relationship specified by the <*relationship handle*> and having the name specified by the <*attribute Name*> argument.

```
VTAPIRSLT RelationshipAttribute (
        VTSHND  sessionHandle,
        VTOHND  relationshipHandle,
        LPSTR   attributeName,
        CoreApiOperands *attributeHandles
);
```

Parameters

sessionHandle
Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationshipHandle
Handle to the relationship definition for which the attribute is desired. A relationship handle can be obtained through a call to the **Relationships** function with a relationship name argument.

attributeName
The name of the attribute for which the relationship value is desired..

attributeHandles
A pointer to a *CoreApiOperands* structure containing an array of attribute handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*attribute handles*> pointer.

.

Return Values

If the function succeeds the return value is zero, and the <*attribute Handles*> pointer has the address of the structure containing the relationship handles. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_RELATIONSHIP_HANDLE The <*relationship handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The relationship specified by the <*relationship handle*> argument has been deleted.

Remarks

The function fails if either the <*session handle*> *or the* <*relationship handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

10.4 Traverse

The **Traverse** function returns the handle to the destination element of a relationship given the source element handle and the relationship handle. The notion of source and destination in this context is arbitrary. Either element of a relationship can be the source or destination. The idea here is that given a relationship and one element of that relationship, traverse will return the other element.

```
VTAPIRSLT Traverse (
        VTSHND  sessionHandle,
        VTOHND  relationshipHandle,
        VTOHND  sourceElementHandle,
        CoreApiOperands *targetElementHandle
);
```

Parameters

sessionHandle
  Handle to the server session that maintains resources for the client application. Upon a successful login, the server creates a session for the client application and returns the session handle to the client application. This session will be the context in which all-subsequent function calls are executed. For example, a session maintains an active project and requests submitted to the server are executed against this project. The API supports multiple sessions within a client process and the <*session handle*> provides the facility to select the context in which function calls are executed.

relationshipHandle
  Handle to the relationship definition. A relationship handle can be obtained through a call to the **Relationships** function with a relationship name argument.

sourceElementHandle
  Handle to the class that is the target of the request.

targetElementHandle

.

A pointer to a *CoreApiOperands* structure containing an array of element handles and the number of handles contained in the array. The client application creates a *CoreApiOperands* structure and passes a reference in the function call. The function will allocate memory to accommodate the handles returned from the server and assign the base address of the memory to the <*target element handle*> pointer.

Return Values

If the function succeeds the return value is zero, and the <*target element handle*> pointer has the address of the structure containing the element handle. A non-zero return code is an indication that the function failed. To get extended error information, call **GetLastSessionError** with the <*session handle*>. The following errors apply to this function:

- INVALID_SESSION_HANDLE The <*session handle*> specified does not map to a valid session.
- INVALID_RELATIONSHIP_HANDLE The <*relationship handle*> argument is undefined or is NULL.
- INVALID_ELEMENT_HANDLE The <*element handle*> argument is undefined or is NULL.
- INSUFFICIENT_PRIVILEGE The user does not have read permission on a target element.
- DEFUNCT_OPERAND The element specified by the <*element handle*> argument has been deleted.

Remarks

The function fails if the <*session handle*>, the <*relationship handle*>, or the <*source element handle*> is invalid. The handles are allocated from the heap. The client application can release the handles with a call to **ReleaseHandles** when the handles are no longer needed.

.