TIBCO Data Virtualization®

Application Programming Interface Guide

Version 8.5.0
Last Updated: October 26, 2021

Contents

Introduction

This topic contains an alphabetical listing of all operations available in TIBCO® Data Virtualization (TDV), and separate descriptions of structures that occur in multiple places among operation elements.

Purpose of the Web Services Operations

Operations let Web Services clients perform all activities to create, update, maintain and destroy resources in TDV and execute SQL, scripts and procedures, as though the clients were local to TDV.

Note: Web Service operations have sometimes been called, collectively, the "Admin API."

Groups of Operations

Operations fall into two main groups:

Administrative Operations

Administrative operations can be found in two places in the resource tree:

- Desktop/Data Services/Web Services/system/admin/
- localhost/services/webservices/system/admin/

Administrative operations are grouped into five branches:

The archive Branch

The archive branch of administrative operations lets Web Services clients monitor and manage archives.

The operations in this branch are:

The execute Branch

The execute branch of administrative operations lets Web Services clients execute SQL, SQL scripts and procedures, and get plans and results.

The operations in this branch are:

- getResourcePlan, page 148
- getResultSetPlan, page 155
- getSqlPlan, page 159
- getTabularResult, page 161
- parseSqlQuery, page 176

The resource Branch

The resource branch of administrative operations lets Web Services clients monitor and manage all TDV resources.

The operations in this branch are grouped by resource category and (in the case of data sources and resources) subcategory:

- Caching
  - clearResourceCache, page 54
  - getCachedResourceStatisticsConfig, page 110
  - getResourceCacheConfig, page 146
  - getResourceStatsSummary, page 152
  - refreshResourceCache, page 184
  - updateCachedResourceStatisticsConfig, page 201
  - updateResourceCacheConfig, page 225
- Connectors
  - createConnector, page 64
  - destroyConnector, page 81
  - getConnectorGroup, page 113
  - getConnectorGroupNames, page 113
  - getConnectors, page 114
  - testDataSourceConnection, page 193
  - updateConnector, page 204
- Data services
  - updateDataServicePort, page 205
  - updateImplementationContainer, page 220

- Data sources
  - createDataSource, page 65
  - getUsedDataSources, page 163
  - updateDataSource, page 207
  - updateDataSourcePort, page 211

  Attribute definitions
  - getDataSourceAttributeDefs, page 115

  Custom types
  - createCustomDataSourceType, page 64
  - destroyCustomDataSourceType, page 81
  - updateCustomDataSourceType, page 205

  Location and Dependencies
  - getDataSourceChildResources, page 115 (deprecated)
  - updateDataSourceChildInfos, page 208 (deprecated)
  - updateDataSourceChildInfosWithFilter, page 209

  Statistics
  - getDataSourceStatisticsConfig, page 118
  - updateDataSourceStatisticsConfig, page 213

  Types
  - getDataSourceTypeAttributeDefs, page 119
  - getDataSourceTypeCustomCapabilities, page 119
  - getDataSourceTypes, page 120
  - getExtendableDataSourceTypes, page 125
  - getParentDataSourceType, page 141
  - updateDataSourceTypeCustomCapabilities, page 214
- Definition sets
  - updateDefinitionSet, page 215

- Introspection
  - clearIntrospectableResourceIdCache, page 53
  - getIntrospectableResourceIdsResult, page 128
  - getIntrospectableResourceIdsTask, page 130
  - getIntrospectedResourceIdsResult, page 132
  - getIntrospectedResourceIdsTask, page 134
  - getIntrospectionAttributeDefs, page 135
  - getIntrospectionAttributes, page 136
  - getMostRecentIntrospectionStatus, page 139
  - introspectResourcesResult, page 167
  - introspectResourcesTask, page 169
- Links
  - createLink, page 74
  - createLinksRecursively, page 76
  - updateLink, page 221
- Reintrospection
  - cancelDataSourceReintrospect, page 48 (deprecated)
  - getDataSourceReintrospectResult, page 117
  - reintrospectDataSource, page 185 (deprecated)

- Resources
  - copyResource, page 58
  - copyResources, page 61
  - createResource, page 78
  - destroyResource, page 84
  - destroyResources, page 85
  - getResource, page 145
  - getResources, page 150
  - getResourceUpdates, page 153
  - getUsedResources, page 164
  - moveResource, page 173
  - moveResources, page 175
  - rebindResources, page 183
  - renameResource, page 190
  - resourceExists, page 192
  - updateResourceAnnotation, page 224
  - updateResourceEnabled, page 228
  - updateResources, page 230
  - updateResourcePrivileges, page 229

  Location and Dependencies
  - getAllResourcesByPath, page 101
  - getAncestorResources, page 101
  - getChildResources, page 111
  - getDependentResources, page 120
  - getParentResource, page 141

Locks

— getLockedResources, page 137

— lockResource, page 172

— lockResources, page 173

— unlockResource, page 194

— unlockResources, page 195

— updateUserLockState, page 241

Owner

— changeResourceOwner, page 51

Privileges

— copyResourcePrivileges, page 60

— getResourcePrivileges, page 149

— updateResourcePrivileges, page 229

Statistics

— cancelResourceStatistics, page 49

— clearResourceStatistics, page 55

— getResourceStatisticsConfig, page 151

— refreshResourceStatistics, page 185

— updateResourceStatisticsConfig, page 231

- SQL and SQL Script
  - — updateExternalSqlProcedure, page 217
  - — updateSqlScriptProcedure, page 234
  - — updateSqlTable, page 235
- Transformation
  - — getTransformFunctions, page 163
  - — updateBasicTransformProcedure, page 200
  - — updateStreamTransformProcedure, page 236
  - — updateTransformProcedure, page 237
  - — updateXQueryTransformProcedure, page 242
  - — updateXsltTransformProcedure, page 245

- Triggers
  - — updateTrigger, page 238
- XQuery
  - — updateXQueryProcedure, page 241
  - — updateXQueryTransformProcedure, page 242
- XSLT
  - — updateXSLTProcedure, page 244
  - — updateXsltTransformProcedure, page 245

The server Branch

The server branch of administrative operations lets Web Services clients manage licenses, clusters, server actions and attributes, and other server-level activities.

The operations in this branch are:

- addLicenses, page 39
- createCluster, page 63
- createDBHealthMonitorTable, page 67
- getClusterConfig, page 112
- getCreateDBHealthMonitorTableSQL, page 114
- getLicenses, page 137
- getServerActions, page 156
- getServerAttributeDefChildren, page 156
- getServerAttributeDefs, page 157
- getServerAttributes, page 158
- getServerName, page 159
- joinCluster, page 171
- performServerAction, page 178
- removeFromCluster, page 187
- removeLicenses, page 187
- repairCluster, page 191
- updateClusterName, page 203
- updateServerAttributes, page 233

- updateServerName, page 233

The user Branch

The branch of administrative operations lets Web Services clients monitor and manage users, groups, and domains.

The operations in this branch are:

- addUsersToGroup, page 40
- addUserToGroups, page 41
- cancelCreateDomain, page 47
- createDomain, page 68
- createGroup, page 72
- createUser, page 80
- destroyDomain, page 82
- destroyGroup, page 83
- destroyUser, page 86
- getDomainGroups, page 121
- getDomains, page 122
- getDomainTypeAttributeDefs, page 123
- getDomainTypes, page 124
- getDomainUsers, page 124
- getGroups, page 127
- getGroupsByUser, page 127
- getUser, page 165
- getUsers, page 165
- getUsersByGroup, page 166
- removeUserFromGroups, page 189
- removeUsersFromGroup, page 190
- updateDomain, page 216
- updateGroup, page 219
- updateUser, page 240

Utility Operations

Utility operations can be found in two places in the resource tree:

- Desktop/Data Services/Web Services/system/util/
- localhost/services/webservices/system/util/

Utility operations are grouped into three branches:

- The common Branch, page 24
- The security Branch, page 24
- The session Branch, page 25

The common Branch

The common branch of utility operations lets Web Services cancel a server task:

- cancelServerTask, page 50

The security Branch

The security branch of utility operations lets Web Services clients manage the following aspects of security:

- General settings
  - getGeneralSettings, page 126
  - updateGeneralSettings, page 218
- Login module
  - addLoginModule, page 39
  - getAvailableLoginModuleNames, page 109
  - getLoginModule, page 138
  - getLoginModuleDefaultProperties, page 138
  - getLoginModuleList, page 139
  - removeLoginModule, page 188
  - updateLoginModule, page 222
  - updateLoginModuleList, page 223

- Principal mapping
  - addPrincipalMapping, page 40
  - getPrincipalMapping, page 142
  - getPrincipalMappingList, page 143
  - removePrincipalMapping, page 188
  - updatePrincipalMapping, page 223
- Row-based security
  - rbsAssignFilterPolicy, page 178
  - rbsDeleteFilterPolicy, page 179
  - rbsGetFilterPolicy, page 180
  - rbsGetFilterPolicyList, page 180
  - rbsIsEnabled, page 181
  - rbsSetEnabled, page 181
  - rbsWriteFilterPolicy, page 182

The session Branch

The session branch of utility operations lets Web Services clients manage sessions.

The operations in this branch are:

- beginSession, page 43
- beginTransaction, page 44
- closeSession, page 57
- closeTransaction, page 57
- getServerInfo, page 158

Purpose of the Procedures

The built-in procedures let users with appropriate privileges perform basic server activities, including:

- Set up log messages for debugging
- Create, update, introspect, maintain, and destroy resources
- Execute SQL, scripts and procedures

- Set and clear JMS message properties
- Send JMS map and text messages
- Create and delete elements, write messages to files
- Test connections and server performance
- Execute a basic transformation on XML input

Groups of Procedures

Procedures fall into several groups:

Debug Procedures

The debug procedures are found in the resource tree under /lib/debug. They are:

- Log, page 303, which writes text to the log file with severity level INFO.
- LogError, page 304, which writes text to the log file with severity level ERROR.
- Print, page 308, which writes debug messages to the console when running from Studio.

Deployment Procedures

The deployment procedures are found in the resource tree under /lib/resource (and under /services/databases/system/deployment/ as published resources). They are:

- ExplainAttributes, page 280, which retrieves the list of attributes of the data sources included in a resource set.
- ExplainPrincipals, page 280, which retrieves the list of principals included in a resource set.
- ExplainResources, page 281, which retrieves the list of resources in a resource set.
- GetPrincipalSet, page 293, which retrieves the list of principals included in a principal set.
- GetResourceSet, page 298, which retrieves the list of resources included in a resource set.
- ListAttributes, page 299, which retrieves the attribute list for a specified resource set on a given site.
- ListPrincipals, page 300, which retrieves the list of principals included in a resource set.
- ListResources, page 301, which retrieves the list of resources included in a resource set.
- PreviewResourceSet, page 307, which retrieves the list of resource changes since the last deployment of the specified resource set, by the specified deployment plan.

JMS Procedures

The JMS procedures are found in the resource tree under /lib/jms. They are:

- ClearMessageProperties, page 271, which clears all JMS headers and properties that were set using SetMessageProperties, page 326.
- SendMapMessage, page 319, which sends a JMS map message based on a ROW type variable.
- SendTextMessage, page 321, which sends a JMS text message.
- SetMessageProperties, page 326, which sets JMS headers or properties for the subsequent JMS messages to be sent using SendTextMessage, page 321.
- SetMessageProperty, page 327, which sets a JMS header or property for the subsequent JMS messages to be sent using SendTextMessage, page 321.

Lineage Procedures

The lineage procedures are found in the resource tree under /lib/resource (and under /services/databases/system/lineage/ as published resources). They are:

- GetColumnDependencies, page 284, which retrieves the column dependencies of a view.
- GetColumnReferences, page 288, which retrieves the column references of a view, table or procedure.

Profile Procedures

The profile procedures are found in the resource tree under /services/database/system/profile. They are:

- GetTableProfiles, page 294, which retrieves statistical and data source lineage information about a set of published tables or views.
- GetColumnProfiles, page 286, which retrieves statistical and auxiliary data type information about a set of published table columns.
- GetPartitionClauses, page 292, which retrieves the partition SQL clauses (predicates) for use in defining partition queries against a published table and column.

Resource Procedures

The resource procedures are found in the resource tree under /lib/resource. They are:

- CancelDataSourceReintrospect, page 268, which cancels an in-progress, non-blocking reintrospection process that was started using ReintrospectDataSource, page 313.
- ClearResourceCache, page 271, which clears the cache on a resource.
- ClearResourceStatistics, page 272, which clears the statistics on a resource.
- CopyResource, page 273, which copies a resource into a folder using a new name.
- CreateResourceCacheKey, page 276, which creates a cache key for a given resource.
- ExplainAttributes, page 280, which retrieves the list of attributes of the data sources included in a resource. (This procedure is also listed with Deployment Procedures, page 27.)

- ExplainResources, page 281, which retrieves the list of resources in a resource set. (This procedure is also listed with Deployment Procedures, page 27.)
- GetColumnDependencies, page 284, which retrieves the column dependencies of a view. (This procedure is also listed with Lineage Procedures, page 28.)
- GetColumnReferences, page 288, which retrieves the column references of a view, table or procedure. (This procedure is also listed with Lineage Procedures, page 28.)
- GetDataSourceReintrospectReport, page 289, which gets the report for a reintrospection, if available.
- GetPrincipalSet, page 293, which retrieves the list of principals included in a principal set. (This procedure is also listed with Deployment Procedures, page 27.)
- GetResourceSet, page 298, which retrieves the list of resources included in a resource set. (This procedure is also listed with Deployment Procedures, page 27.)
- ListAttributes, page 299, which retrieves the attribute list for a specified resource set on a given site. (This procedure is also listed with Deployment Procedures, page 27.)
- ListResources, page 301, which retrieves the list of resources included in the specified resource set. (This procedure is also listed with Deployment Procedures, page 27.)
- LoadResourceCacheStatus, page 302, which loads the status for a resource cache.
- MoveResource, page 305, which moves a resource into a folder using a new name.
- PreviewResourceSet, page 307, which retrieves the list of resource changes since the last deployment of the specified resource set, by the specified deployment plan. (This procedure is also listed with Deployment Procedures, page 27.)
- RefreshResourceCache, page 310, which refreshes the cache for a view or table resource.
- RefreshResourceCacheSynchronously, page 311, which refreshes the cache on a TABLE resource synchronously.
- RefreshResourceStatistics, page 312, which refreshes the statistics on a resource for use by the cost-based optimizer.
- ReintrospectDataSource, page 313, which performs a reintrospection of a data source.

- RenameResource, page 313, which renames a resource.
- ResourceExists, page 314, which checks to see if a resource exists.
- SendResultsInEMail, page 320, which sends an email message with specified headers and content, and with the results of the given view or procedure as attachments.
- SqlPerf, page 330, which runs a SQL performance test.
- TestAllDataSourceConnections, page 331, which tests all data sources to see if they are operational.
- TestDataSourceConnection, page 331, which tests a specific data source to see if it is operational.
- UpdateResourceCacheEnabled, page 333, which updates the enabled state of a resource cache.
- UpdateResourceCacheKeyStatus, page 334, which updates the cache key for a specified resource of type TABLE, or of type PROCEDURE with zero parameters.
- UpdateResourceEnabled, page 335, which updates the enabled state of a DATA_SOURCE resource.

Service Procedures

The service procedures are found in the resource tree under /lib/services. They are:

- AddUsernameToken, page 267, which adds a WS-Security UsernameToken to a SOAP envelope.
- CreateElement, page 275, which creates a child element in an XML document or element.
- DeleteElement, page 276, which deletes one or more element nodes from an XML document or element.
- LogMessageToFile, page 304, which writes the contents of a message to a file at a specified path.
- SetEnvironmentFromNodeValue, page 326, which evaluates an XPath expression against the envelope, and stores the result in the specified environment variable.
- SetNodeValueFromEnvironment, page 328, which sets an element or attribute value from an environment variable.

Transformation Procedure

The transformation procedure, ExecuteBasicTransform, page 279, performs a basic transformation on the input XML and retrieves metadata information.

User Procedures

The user procedures are found in the resource tree under /lib/users. They are:

- ClearAlternatePrincipal, page 270, which clears all credential data previously stored using the setDataSourceCredentials JDBC method or the SetDataSourceCredentials, page 323 procedure.
- ExplainPrincipals, page 280, which retrieves the list of principals included in a specified resource set. (This procedure is also listed with Deployment Procedures, page 27.)
- ListPrincipals, page 300, which retrieves the list of principals included in the specified resource set. (This procedure is also listed with Deployment Procedures, page 27.)
- SetAlternatePrincipal, page 322, which establishes an alternate identity within the current session, preserving the original identity for afterwards.
- SyncDomain, page 330, which synchronizes the local external domain with the specified external domain server.
- TestUserIdentity, page 332, which allows a SQL script to determine if the current identity matches the one specified.

Utility Procedures

The utility procedures are found in the resource tree under /lib/util. They are:

- ClearAllDataSourceCredentials, page 270, which clears all credential data previously stored using the setDataSourceCredentials JDBC method or the SetDataSourceCredentials, page 323 procedure.
- ClearMessageProperties, page 271, which clears all JMS headers and properties that were set using SetMessageProperties, page 326.
- GenerateEvent, page 282, which generates a custom event with the specified name and value.
- GetEnvironment, page 290, which retrieves values from the environment.
- GetProperty, page 295, which gets the values of system properties.
- Pause, page 306, which causes the current thread of control.

- SendEMail, page 318, which sends an email message with the specified headers and content.
- SendMapMessage, page 319, which sends a JMS map message based on a ROW type variable.
- SendTextMessage, page 321, which sends a JMS text message.
- SetDataSourceCredentials, page 323, which sets a username and password for pass-through authentication with a specific data source.
- SetEnvironment, page 324, which sets an environment variable to a value.
- SetMessageProperties, page 326, which sets JMS headers or properties for the subsequent JMS messages to be sent using SendTextMessage, page 321.
- SetMessageProperty, page 327, which sets a JMS header or property for the subsequent JMS messages to be sent using SendTextMessage, page 321.

Security Features

TDV operations and built-in procedures support the following security features:

- You can specify HTTPS (secure HTTP) for data sources and data services. See updateDataSourcePort, page 211, and updateDataServicePort, page 205.
- You can specify a variety of authentication methods (HTTP BASIC, HTTP DIGEST, WSS user name token, NTLM, or KERBEROS) when you set up a data service (updateDataServicePort, page 205).
- You can specify a password any time you create or update a user in the composite domain (createUser, page 80, and updateUser, page 240), begin a session (beginSession, page 43), join a cluster (joinCluster, page 171), or work with encryption (see next bullet).

Using Web Services Operations

This topic describes how to use operations from the Studio user interface or using a code generator.

Using Operations in Studio

In Studio, operations are prepared and run as procedures. For a description of the procedure editor, see "Procedures and Transformations" in the *TDV User Guide*.

This section has the following topics:

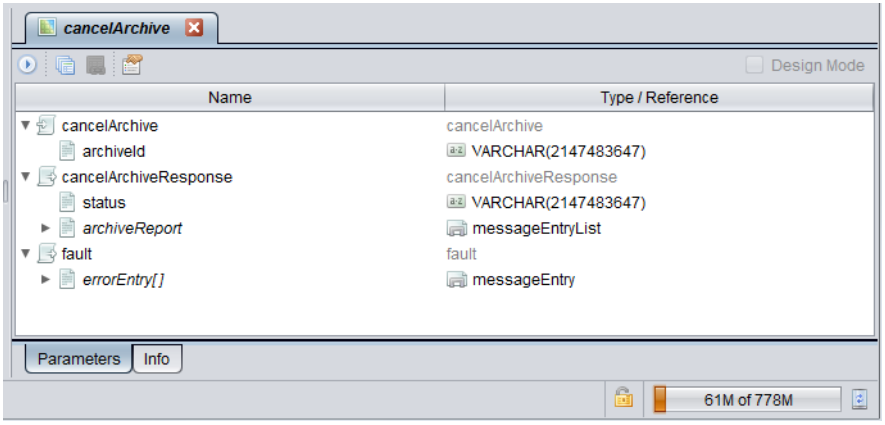
Finding and Opening Operations

If you already know which operator you want to use, expand the /system/admin or /system/util branch of the resource tree and locate the operation. For help finding the operator you want, see the list in Groups of Operations, page 15.

To open an operation

1. Double-click the operation in the Studio resource tree, or select Open from the context menu.

   The operation opens on the Parameters tab of an XML to Tabular Mapping window in the procedure editor.



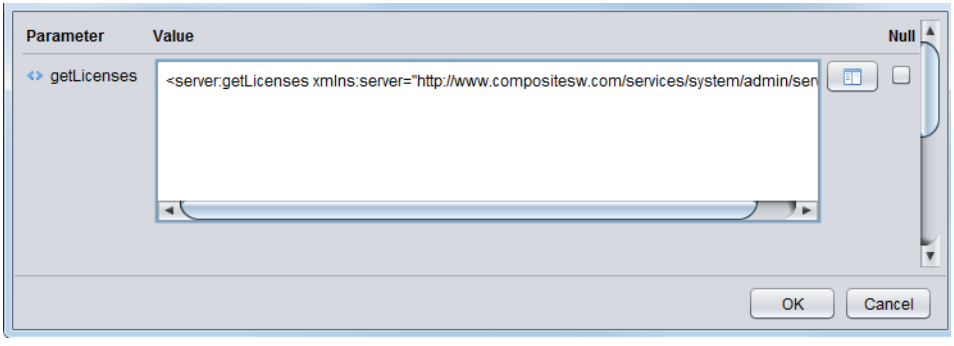
Preparing and Executing an Operation

This section describes how to prepare and execute an operation in Studio.

For a description of the procedure editor, see "Designing Procedures" in the *TDV User Guide*.

To prepare and execute an operation

1. In the procedure editor, click the Execute button on the toolbar. An Input Values window opens for input values (even if the operator takes no request elements).



2. If you want to empty the window of XML text, check the Null checkbox on the right side of this window.
3. If you want to view the operation's XML in formatted, colored text (pretty printing), click the Details button.

   An XML editor window opens, showing the full XML script (if you have not nulled it).
4. Edit the XML as needed to specify values for request elements, either in the XML editor window (click **OK** when you have finished) or in the Input Values window.
5. To continue with execution, click **OK** on the Input Values window.

   When the operation has been executed, a Result tab opens in the lower part of the procedure editor panel, showing the first line of the response XML and the first line of the fault XML.
6. If you want to see the full response XML, click the Details button to the right of the <operation_name> Response field.

   To close that Value window, click **OK**.
7. If you want to see the complete list of faults, click the Details button to the right of the fault field.

   To close that Value window, click **OK**.

Using Operations from a Web Services Client

As a Web services client, you can use any built-in operation by submitting an XML document to TDV using the appropriate Web services port. Response and fault elements are returned using the same port.

This section has the following topics:

Web Services Port

Web services use the HTTP base port, which by default is set to 9400. You can change the value of the base port in the Configuration window, if required, after installation. Navigate to ...Web Services Interface > Communications > HTTP, change the value of Port (On Server Restart), and restart the TDV Server.

You can test the port connection (even if from the local Studio machine) by right-clicking either the admin node or the util node in the resource tree and selecting Test Service.

WSDL Definitions of Operations

All operations are defined in two WSDL files: one for /system/admin operations, and one for /system/util operations. You can view one of the WSDL files by right-clicking either the admin node or the util node in the resource tree and selecting View WSDL.

Note: A View WSDL link is also available in the upper right corner of the URI page that appears when you right-click either the *admin* node or the *util* node in the resource tree and select Test Service. See Web Services Port, page 35.

Run a code generator with the WSDL definitions as input, or hand-code your XML documents.

Web Services Operations

This topic contains an alphabetical listing of all Web services operations available in TDV, and separate descriptions of structures that occur in multiple places among operation elements.

Operations Reference

This section describes all TDV/Studio operations in alphabetical order, along with their resource tree location, request elements, response elements, and faults.

In Studio, *optional* request or response elements are displayed in italic on the Parameters tab. In this documentation, as on the Info tab, these elements are followed by "optional" in parentheses.

Locations are shown as /system/admin/<branch> or /system/util/<branch>. In the resource tree, operations appear in these branches under both Desktop and localhost.

Input Values

To enter the input values for an operation, open the operation and click the Execute button in the upper left corner of its pane (on any tab, even Info). An Input Values window opens, with the operation name listed under Parameter. The Value area contains a hierarchical series of lines in which to type values for the request elements.

For example, if you click the Execute button for the updateResourcePrivileges operation, this window appears.

To assign READ and WRITE privileges to groupABC on datasource NZ 6.0, find the question marks in the syntax and type the values shown in bold, and then click OK:

```
<resource:updateResourcePrivileges
xmlns:resource="http://www.compositesw.com/services/system/admin/r
esource">
  <resource:updateRecursively>false</resource:updateRecursively>

<resource:updateDependenciesRecursively>false</resource:updateDepe
ndenciesRecursively>

<resource:updateDependentsRecursively>false</resource:updateDepend
entsRecursively>
  <resource:privilegeEntries>
    <resource:privilegeEntry>
      <resource:path>/shared/NZload/NZ 6.0</resource:path>
      <resource:type>DATA_SOURCE</resource:type>
      <resource:privileges>
        <resource:privilege>
          <resource:domain>composite</resource:domain>
          <resource:name>groupABC</resource:name>
          <resource:nameType>GROUP</resource:nameType>
          <resource:privs>read write</resource:privs>
          <resource:combinedPrivs>0</resource:combinedPrivs>
          <resource:inheritedPrivs>0</resource:inheritedPrivs>
        </resource:privilege>
      </resource:privileges>
    </resource:privilegeEntry>
  </resource:privilegeEntries>
  <resource:mode>SET_EXACTLY</resource:mode>
</resource:updateResourcePrivileges>
```

Note: If you open the Input Values window and do not replace the question marks with actual values, and then click OK, you get an "internal error" message (in this example, 'Cause: "?" is not a valid resource type.'

addLicenses

Register with the server one or more licenses provided within the license text. Also see getLicenses, page 137 and removeLicenses, page 187.

Location

/services/webservices/system/admin/server/operations/

Request Elements

licenseText: The license text.

Response Elements

licenses: List of licenses that were registered with the server. See Licenses Element, page 254.

Faults

IllegalArgument: If any of the provided licenses are invalid.

Security: The user must have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

addLoginModule

Add a login module to the PAM login sequence.

Location

/services/webservices/system/util/security/operations/

Request Elements

name

group (optional)

enabled (optional)

properties: List of properties for the login module:

Response Elements

id: The ID value of the new login module instance.

Faults

IllegalArgument: If the input XML is not a valid Login Module.

Security: If no implementation exists for the named login module.

Security: If the user is not composite\admin.

addPrincipalMapping

Add a principal mapping to use during user authentication.

Location

/services/webservices/system/util/security/operations/

Request Elements

The principal mapping to use: type, identifier, group.

Response Elements

id: The ID value for the new principal mapping instance.

Faults

Illegal Input: If the input is empty or not a valid principal mapping.

Security: If the user is not composite\admin.

addUsersToGroup

Add one or more users to a domain's group.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

groupName: The group name.

userNames: List of user names (and optionally domains) to add to the group.

Request Example

```
<user:addUsersToGroup
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domainName>composite</user:domainName>
  <user:groupName>production-3</user:groupName>
  <user:userNames>
<user:entry>
      <user:name>jean</user:name>
      <user:domain>composite</user:domain>
    </user:entry>
<user:entry>
      <user:name>kim</user:name>
      <user:domain>composite</user:domain>
    </user:entry>
</user:userNames>
</user:addUsersToGroup>
```

Response Elements

N/A

Faults

NotAllowed: If the group cannot be updated as requested. The group membership may not be updatable, like the composite domain's "all" group.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the group does not exist.

NotFound: If any of the provided users do not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

addUserToGroups

Add a user to one or more groups within a domain.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The user's domain name.

userName: The user name.

groupNames: List of group names (and their domains if needed) to add the user into.

Request Example

In this example, you are adding user composite\jean to the admin group in the composite domain and to the mgrs group in the production domain.

```
<user:addUserToGroups
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domainName>composite</user:domainName>
  <user:userName>jean</user:userName>
  <user:groupNames>
    <user:entry>
      <user:name>admins</user:name>
      <user:domain>composite</user:domain>
    </user:entry>
    <user:entry>
      <user:name>mgrs</user:name>
      <user:domain>production</user:domain>
    </user:entry>
   </user:groupNames>
</user:addUserToGroups>
```

Response Elements

N/A

Faults

NotAllowed: If the user cannot be updated as requested. The group membership may not be updatable, like the composite domain's "all" group.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the user does not exist.

NotFound: If any of the provided groups do not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

beginSession

Start a new session. Web services use a connectionless protocol. This operation allows multiple Web service invocations to appear as if they belong to the same session.

This operation places a cookie on the HTTP connection. If the client supports cookies, no additional actions are required to maintain the session.

As a second option, the client can provide the returned session token in the HTTP basic authentication using `&sessionToken=X` for the user name and providing a blank password, where `X` is the session token returned by this operation.

As a third option, the client can provide the returned session token with a SOAP header element sessionToken like the following:

```
  <soap-env:Envelope
    xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:security="http://www.compositesw.com/services/system/util
/security">
    <soap:Header>
      <security:sessionToken>b9f5d033fb4a81-b02b5b760641c1d6</security:s
essionToken>
    </soap:Header>
    <soap-env:Body>
      <!-- write body element -->
    </soap-env:Body>
  </soap-env:Envelope>
```

If the token is received by the server through more than one of these approaches, precedence is first given to the SOAP header, then to the HTTP authentication, and lastly to the cookie.

While the session is open, all responses for requests made during the session contain a “PUT_SOMETHING_HERE” session token in the SOAP header element.

Location

/services/webservices/system/util/session/operations/

Request Elements

sessionTimeout (optional): The number of seconds to keep an inactive session alive before it is automatically closed.

> To view the default session timeout, select Administration > Configuration from the Studio menu bar and check the value of the Session Timeout parameter under ...> Sessions.

sessionName (optional): The session name. This usually includes the application name as part of the string.

Request Example

```
<session:beginSession
xmlns:session="http://www.compositesw.com/services/system/util/session">
  <session:sessionTimeout>1200</session:sessionTimeout>

<session:sessionName>backup-restore-monthly</session:sessionName>
</session:beginSession>
```

Response Elements

sessionToken: A token that uniquely identifies this session. This should be used within the SessionToken SOAP header for future invocations of operations.

Faults

IllegalArgument: If sessionTimeout is present and is not a positive integer.

NotAllowed: If a session for the current connection is already open.

beginTransaction

Start a transaction on the current session. The transaction continues until closeTransaction, page 57 is invoked. Within a transaction, each operation that is invoked becomes a work unit that is part of the transaction. A transaction must be initiated before calling procedures like createExportArchive, page 69 and createImportArchive, page 73. Later, closeTransaction must be called to COMMIT or ROLLBACK the work units.

When the transaction is closed using a ROLLBACK, all work units within the transaction are rolled back, if the work units support being rolled back.

When the transaction is closed using a COMMIT, all of the work units are committed. If a failure occurs doing the transaction commit, the server attempts to close the transaction using the technique specified by the transaction mode.

The transactionMode can include one of the following options that control compensation behavior:

- BEST_EFFORT: If a failure occurs during commit, log that failure and continue to commit the remaining work units.
- COMPENSATE: If a failure happens during commit, compensate all previously committed work units and rollback all work units that have not yet been committed. (Default value.)
- NO_COMPENSATE: If a failure happens during commit, do nothing to previously committed work units and roll back all work units that have not yet been committed.

The transactionMode can also include one of the following options that controls the behavior if the server goes down:

- FAIL_INTERRUPT: Log enough information so that when the server restarts it can compensate any work units that were committed before the interrupt.
- IGNORE_INTERRUPT: Do no transaction logging. This option yields the best performance. (Default value.)
- LOG_INTERRUPT: Log before the start of the commit and at key points within the commit, so that if an interrupt occurs, it can be reconstructed.

Location

/services/webservices/system/util/session/operations/

Request Elements

transactionMode (optional): A space-separated list of up to two options, one from each of these groups of three:

- BEST_EFFORT, COMPENSATE, or NO_COMPENSATE
- FAIL_INTERRUPT, IGNORE_INTERRUPT, or LOG_INTERRUPT

Request Example

```
<session:beginTransaction
xmlns:session="http://www.compositesw.com/services/system/util/ses
sion">
  <session:transactionMode>BEST_EFFORT
IGNORE_INTERRUPT</session:transactionMode>
</session:beginTransaction>
```

Response Elements

N/A

Faults

IllegalArgument: If transactionMode contains modes that are not listed above or contains conflicting modes.

IllegalState: If a transaction is already open in the current session.

cancelArchive

Cancel an in-progress archive that was started with createExportArchive, page 69 or createImportArchive, page 73. After the operation returns, the archive ID is no longer valid.

The archive ID is only valid during a single transaction, so this operation can only be used within an explicit transaction that also contains either createExportArchive or createImportArchive. It can be used any time after createExportArchive or createImportArchive is called, as long as this is done within the same transaction and the archive has not been closed.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The archive ID provided by the initial createExportArchive or createImportArchive.

Request Example

```
<archive:cancelArchive
xmlns:archive="http://www.compositesw.com/services/system/admin/archive">
  <archive:archiveId>archive_0037</archive:archiveId>
</archive:cancelArchive>
```

Response Elements

status:

- CANCELED: If the archive was successfully canceled.
- SUCCESS or FAIL (as appropriate): If the archive had already completed prior to this operation.

archiveReport (optional): If the status is SUCCESS or FAIL, lists of errors or other messages that occurred during the archive. Otherwise, this element does not exist. For the format of each entry element in archiveReport, see Messages Element, page 255.

Faults

NotFound: The archiveId must still exist at the time of cancelArchive execution. This can occur if the archive was previously canceled using this procedure, if the getExportData returned the last chunk of data, if performArchiveImport, page 177 previously returned a SUCCESS or FAIL status, or if this operation is called on a different transaction.

cancelCreateDomain

Cancel creation of a domain that is currently in the process of being created.

This operation can be used in a separate session or transaction from the one that started the domain creation using createDomain, page 68.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The name of the domain being created that should be canceled.

Request Example

```
<user:cancelCreateDomain
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domainName>production-HA</user:domainName>
</user:cancelCreateDomain>
```

Response Elements

status:

- CANCELED: If domain creation was successfully canceled.
- SUCCESS: If the domain was already created prior to this call and therefore could not be canceled.

Faults

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: Domain does not exist. This may also occur if the domain creation failed.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

cancelDataSourceReintrospect

Deprecated as of API version 6.0. Use cancelServerTask, page 50 instead.

Cancels a non-blocking reintrospect that was started with reintrospectDataSource, page 185 and is still in progress. After this operation returns, the reintrospect ID is no longer valid.

The reintrospect ID is only valid during a single transaction, so this operation can only be used within an explicit transaction that also contains reintrospectDataSource.

Location

/services/webservices/system/admin/resource/operations/

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource and which has *not* been deprecated.

Request Elements

reintrospectId: The reintrospect ID is given by reintrospectDataSource.

Response Elements

status:

- CANCELED: If the reintrospect was successfully canceled.
- SUCCESS or FAIL (as appropriate): If the reintrospect had already completed prior to this operation.

reintrospectReport (optional): If the reintrospect status is SUCCESS or FAIL, list of errors or the changes that occurred during the reintrospect. Otherwise, this element does not exist. For the format of each changeEntry element in reintrospectReport, see Messages Element, page 255.

Faults

NotFound: If the reinstrospectId does not exist. This can occur if the reintrospect was previously canceled using this procedure, if the report was retrieved using getDataSourceReintrospectResult, or if this operation is called on a different transaction.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

cancelResourceStatistics

Cancel a statistics gathering process that is currently in progress. No action is taken if the specified resources is not currently gathering statistics. Does not block until the cancel signal is processed.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: Path to resource.

type: Type of resource. Can only be a relational physical DATA_SOURCE or a TABLE.

Request Example

```
<resource:cancelResourceStatistics
xmlns:resource="http://www.compositesw.com/services/system/admin/r
esource">
  <resource:path>/shared/examples/ds_inventory</resource:path>
  <resource:type>DATA_SOURCE</resource:type>
</resource:cancelResourceStatistics>
```

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: The user must have READ access on all items in path.

Security: The user must have the ACCESS_TOOLS right.

cancelServerTask

Cancel a running server task.

Server tasks are long-running processes maintained by TDV. They are associated with the TDV session that created the task. If the TDV session ends, the task ends.

Operations that create server tasks return a taskId. This taskId can be used to later query task results or cancel the task. Such operations generally have the Task suffix in their name.

Normally, users can only cancel server tasks within the same TDV session used to create the task. Users with the MODIFY_ALL_STATUS right can cancel tasks regardless of which session owns the task.

Location

/services/webservices/system/util/common/operations/

Request Elements

taskId: The server task ID.

Response Elements

N/A

Faults

IllegalArgument: If the taskId is malformed.

NotFound: If there currently exists no task with the given taskId.

Security: If the current TDV session is different from the one originally used to create the task and the user does not have the MODIFY_ALL_STATUS right.

Security: If the user does not have the ACCESS_TOOLS right.

changeResourceOwner

Change the owner of a resource to the user specified by the newOwnerName input attribute. The new resource owner gets only the GRANT privilege, but this allows management of the resource and delegation of resource privileges as appropriate.

This operation can only be called by users with the MODIFY_ALL_RESOURCES right.

You cannot change ownership of resources already owned by system. The reverse is also true: resources cannot change ownership to become a system-owned resource. Additionally, resources cannot be owned by any user in the dynamic domain, or by the anonymous user in the composite domain. Attempts to change resources this way are quietly ignored.

If the recurse element is TRUE (set to 1) and the resource is a container, the ownership change affects all resources within the container, including the container itself.

Changing the ownership of a physical data source implicitly affects all contained resources.

Resources within physical data sources cannot have their ownership directly changed. If recurse is TRUE, the physical data source is changed, but this operation does not recurse into the data structure, because the ownership change of the child resources is implicitly changed to match the data source.

The currentOwnerName attribute restricts change of resource ownership to only those resources that are owned by the currently named owner. When the currentOwnerName feature is used with the recurse feature, the combined effect changes ownership of only those child resources owned by currentOwnerName.

The default value for newOwnerDomain and currentOwnerDomain is composite unless otherwise specified. They are optional input attributes.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to assign to a new owner.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

newOwnerName: The name of the new owner.

newOwnerDomain (optional): The domain name of the new owner. Defaults to composite if not set.

currentOwnerName (optional): The name of the current owner. When user name is set, this restricts ownership changes to those resources currently owned by the specified user.

currentOwnerDomain (optional): The domain name of the current owner. Defaults to composite if not set, and to currentOwnerName if set.

recurse (optional): If set to TRUE (1), the ownership change applies recursively to child resources within the resource container. Default value is 0, meaning no recursion is attempted.

Response Elements

resources: List of the modified resources and resource definitions. See Resources Element, page 257.

Faults

IllegalArgument: Either the path element was malformed or an illegal type or detail was provided.

NotAllowed: If the resource is a child of a physical data source.

NotAllowed: If the resource is currently owned by system.

NotAllowed: If newOwnerName is system in the composite domain.

NotAllowed: If newOwnerName is anonymous in the composite domain.

NotAllowed: If newOwnerDomain is dynamic.

NotAllowed: If currentOwnerName is system in the composite domain.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the resource or any portion of the path to the resource does not exist.

NotFound: If the newOwnerName or currentOwnerName does not exist.

NotFound: If the newOwnerDomain or currentOwnerDomain does not exist.

Security: If the user does not have READ access on all items in the path to the resource.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have ACCESS_TOOLS and MODIFY_ALL_RESOURCES rights.

Note: Separate actions may be needed to make a resource accessible and visible to the new owner. The changeResourceOwner, page 51 operation does not move resources into accessible directories. Use moveResource, page 173 or moveResources, page 175 to place the resource in a directory for which the new owner has read privileges on all parent containers.

clearIntrospectableResourceIdCache

Clear an existing data source resource identifier cache.

Resource identifier caches are created on a data source or reused when getDataSourceResourceIdentifiers is called. The cache contains all known native resources within a data source.

One such cache exists per data source per set per data source user.

The data source user associated with TDV session is used. For non-dynamic data sources, this user is the user specified in the data source connection parameters. For dynamic data source, this is the TDV session user that is passed to the data source.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the data source.

Request Example

```
<resource:clearIntrospectableResourceIdCache
xmlns:resource="http://www.compositesw.com/services/system/admin/r
esource">
  <resource:path>/shared/examples/ds_inventory</resource:path>
</resource:clearIntrospectableResourceIdCache>
```

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed.

NotFound: If the data source or any portion of the path to the data source does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

clearResourceCache

Clear an existing resource cache. Only purposefully configured resources of type TABLE and PROCEDURE support caching. Procedure variants are cleared from the cache along with any cached results.

Use the refreshResourceCache, page 184 procedure to initiate an immediate refresh of a table or cached view, or allow the cache to be refreshed on next use of the resource.

Location

/services/webservices/system/admin/resource/operations/

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Request Elements

path: The path of the resource configured to use caching.

type: The type of the resource. Valid values are TABLE and PROCEDURE.

Request Example

```
<resource:clearResourceCache
xmlns:resource="http://www.compositesw.com/services/system/admin/r
esource">

<resource:path>/shared/examples/ds_inventory/inventorytransactions
</resource:path>
  <resource:type>TABLE</resource:type>
</resource:clearResourceCache>
```

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalState: If the cache is disabled.

NotAllowed: The resource type must support caching. Only TABLE and PROCEDURE resources support caching.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have READ access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

clearResourceStatistics

Clear statistics data on the specified resource. Statistics can be gathered on data sources and tables.

Resource statistics can be refreshed directly by calling the refreshResourceStatistics, page 185 operation. Statistics become available next time a refresh happens.

Location

/services/webservices/system/admin/resource/operations/

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Request Elements

path: The path of the cached resource.

type: The resource type. Valid type values are a relational physical DATA_SOURCE or TABLE.

Request Example

```
<resource:clearResourceStatistics
xmlns:resource="http://www.compositesw.com/services/system/admin/r
esource">

<resource:path>/shared/examples/ds_inventory/inventorytransactions
</resource:path>
  <resource:type>TABLE</resource:type>
```

```
</resource:clearResourceStatistics>
```

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalState: If statistics gathering is disabled.

NotAllowed: If the resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

closeResult

Close the given result associated with the ID. All system resources associated with the result are closed. This includes cursors, query statistics, and anything else tracked by the system for the result. If the given ID is for procedural result, all output cursors are closed.

There is no need to call closeResult for each tabular resultId returned within procedural result. If the execution associated with the ID has not completed, it is terminated.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

id: A tabular or procedural result ID.

Response Elements

N/A

Faults

NotFound: If the result for the given ID does not exist within the current transaction or has already been closed.

closeSession

Close the current session. See beginSession, page 43 for more information.

Location

/services/webservices/system/util/session/operations/

Request Elements

N/A

Request Example

```
<session:closeSession
xmlns:session="http://www.compositesw.com/services/system/util/session"/>
```

Response Elements

N/A

Faults

IllegalState: If the current session is not currently open.

closeTransaction

Close the current transaction by either committing it or rolling it back, depending on the action specified.

- COMMIT commits all changes made in the current transaction. Failures during commit are handled as specified in beginTransaction, page 44.
- ROLLBACK rolls back all changes made in the current transaction.

Location

/services/webservices/system/util/session/operations/

Request Elements

action: COMMIT or ROLLBACK.

Request Example

```
<session:closeSession
xmlns:session="http://www.compositesw.com/services/system/util/session"/>
```

Response Elements

N/A

Faults

IllegalArgument: If the action is not COMMIT or ROLLBACK.

IllegalState: If no transaction is currently open in the current session.

TransactionFailure: If there is an error in committing the transaction.

copyResource

Replicate the specified resource into an existing folder. This procedure allows the resource to be copied into a new location and renamed. Owner and security privileges remain the same in the new copy of the resource.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The TDV source path, which is also known as the Resource Name for copying.

type: The type of the source resource to be copied. Valid TDV types are DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER

targetContainerPath: The path of the target container to copy the resource into.

newName: The new name to call the copied resource.

copyMode: Controls behavior in the case where a resource exists with the same name and type in the container specified by newPath. This element can be one of the following:

- ALTER_NAME_IF_EXISTS: If a resource of the same name and type of the source resource already exists in the target container, avoid conflicts by automatically generating a new name. Names are generated by appending a number to the end of the provided name.
- FAIL_IF_EXISTS: Fails if a resource of the same name and type already exists in the target container. The resource are not copied if this occurs.
- OVERWRITE_MERGE_IF_EXISTS: If a resource of the same name and type as the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, merge the contents of the source container with the corresponding resource in the target. All resources in the source container overwrite those resources with the same name in the target, but child resources in the target with different names are not overwritten and remain unaltered.
- OVERWRITE_REPLACE_IF_EXISTS: If a resource of the same name and type of the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, replace the container within the target container with the source container. This is equivalent to deleting the container in the target before copying the source.

Response Elements

N/A

Faults

DuplicateName: If a resource in the target container exists with the same type as the source, same name as newName, and the copy mode is FAIL_IF_EXISTS.

IllegalArgument: If any of the given paths or types are malformed, or if the copyMode is not one of the legal values.

IllegalState: If the source resource is not allowed to be copied. Resources in */services/databases/system*, */services/webservices/system*, or within any physical data source, cannot be copied.

NotAllowed: If the source resource is not allowed to exist within the target container. Resources cannot be copied into a physical data source. LINK resources can only be copied into a RELATIONAL_DATA_SOURCE, SCHEMA, or a PORT under */services*. Non-LINK resources cannot be copied into any location under */services*.

NotFound: If the source resource or any portion of the new path does not exist.

Security: If the user does not have READ access on all items in the source path.

Security: If the user does not have READ access on the items in targetContainerPath other than the last item.

Security: If the user does not have WRITE access to the last item in targetContainerPath.

Security: If the user does not have WRITE access to a resource that is to be overwritten in one of the overwrite modes.

Security: If the user does not have the ACCESS_TOOLS right.

copyResourcePrivileges

Enable changes to resource privileges for users and groups, by copying privileges from other resources.

Changes can be made to one or more resources with different source resources for one or many users and groups. Resource privileges can be set for a specified set of users and groups without modifying any existing privileges for other users and groups, or the procedure can set resource privileges restrictively to only privileges on the explicit source resources.

Only a user with GRANT privilege on a resource can modify the privileges for that resource. The owner of a resource always has GRANT privilege, as do users with the MODIFY_ALL_RESOURCES right.

When mode is OVERWRITE_APPEND or is not supplied, privileges are applied user by user, so that updating privileges for one user or group does not alter privileges from any other user or group.

The privileges applied for a user or group replace the previous value for that user or group.

When mode is SET_EXACTLY, all privileges on the resource are made to look exactly like the privileges of source resource.

When updateRecursively is FALSE, privileges are applied only to the specified resources. When it is TRUE, privileges are recursively applied into any CONTAINER or DATA_SOURCE resource specified. When recursively applying privileges, the privilege change is ignored for any resource the user lacks owner privileges for. Privileges that are not applicable for a given resource type are automatically stripped down to the set that is legal for each resource:

- TABLE resources support NONE, READ, WRITE, SELECT, INSERT, UPDATE, and DELETE.
- PROCEDURE resources support NONE, READ, WRITE, and EXECUTE.
- All other resource types only support NONE, READ, and WRITE.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

updateRecursively: If TRUE, all children of the given resources are recursively updated with the privileges assigned to their parent.

copyPrivilegeEntries: List of entries containing one source resource and a list of destination resources. Privileges of the source resource are copied over to destination resources in the specified mode.

- copyPrivilegeEntry (optional):
- srcResource: Path and type of source resource.
- dstResource (one or more): Path and type of destination resource.

mode (optional): Determines whether privileges are merged with existing ones:

- OVERWRITE_APPEND (default) merges and does not update privileges for users or groups not mentioned.
- SET_EXACTLY makes privileges look exactly like those provided in the call.

Response Elements

N/A

Faults

IllegalArgument: If any path is malformed, or any type or privilege entry is illegal, or mode is not one of the legal values.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If a path refers to a resource that does not exist.

Security: If for a given entry path the user does not have READ access on any item in a path other than the last item, or does not have GRANT access on the last item.

Security: If the user does not have the ACCESS_TOOLS right.

copyResources

Copy the specified resources into a folder.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of source path-type pairs to copy.

targetContainerPath: The path of the target container to copy the resources into.

copyMode: Controls behavior in the case where a resource exists with the same name and type in the container specified by newPath. It can be one of the following:

- ALTER_NAME_IF_EXISTS: If a resource of the same name and type as the source resource already exists in the target container, avoid conflicts by automatically generating a new name. Names are generated by appending a number to the end of the provided name.
- FAIL_IF_EXISTS: Fail if a resource of the same name and type of the source resource already exists in the target container. No resources are copied if this occurs.
- OVERWRITE_MERGE_IF_EXISTS: If a resource of the same name and type of the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, merge the contents of the source container with the corresponding resource in the target. All resources in the source container overwrite those resources with the same name in the target, but child resources in the target with different names are not overwritten and remain unaltered.
- OVERWRITE_REPLACE_IF_EXISTS: If a resource of the same name and type of the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, replace the container within the target container with the source container. This is equivalent to deleting the container in the target before copying the source.

Response Elements

N/A

Faults

DuplicateName: If a resource in the target container exists with the same name and type as one of the source and the copy mode is FAIL_IF_EXISTS.

IllegalArgument: If any of the given paths or types are malformed, or if the copyMode is not one of the legal values.

IllegalState: If the source resource is not allowed to be copied. Resources in /services/databases/system, /services/webservices/system, or within any physical data source, cannot be copied.

NotAllowed: If any of the source resources are not allowed to exist within the target container. Resources cannot be copied to a physical data source. LINK resources can only be copied into a RELATIONAL_DATA_SOURCE, SCHEMA, or a PORT under /services. Non-LINK resources cannot be copied into any location under /services.

NotFound: If any of the source resources or any portion of the new path does not exist.

Security: If the user does not have READ access on all items in the source paths.

Security: If the user does not have READ access on the items in the newPath other than the last item.

Security: If the user does not have WRITE access to the last item in newPath.

Security: If the user does not have WRITE access to a resource that is to be overwritten in one of the overwrite modes.

Security: If the user does not have the ACCESS_TOOLS right.

createCluster

Create a cluster on this server node.

Location

/services/webservices/system/admin/resource/operations

Request Elements

clusterName: The display name of the cluster to create.

Request Example

```
<server:createCluster
xmlns:server="http://www.compositesw.com/services/system/admin/server">
  <server:clusterName>HQ-cluster-113</server:clusterName>
</server:createCluster>
```

Response Elements

N/A

Faults

IllegalState: If the server is already part of a cluster.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

createConnector

Create a connector.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

connector: The connector to create. See Connector Element, page 249.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

connector: The connector created.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

DuplicateName: If a connector with the same name already exists.

Security: If the user does not have READ access on all items in path except the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

createCustomDataSourceType

Create a custom data source type.

Note: Invoke getDataSourceTypes, page 120 to get a complete list of existing valid data source types.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceType: This element should have a name, type, JDBC URL pattern, driver class name, and parent data source type name. For optional attributes, see Attributes Element, page 248.

Response Elements

resources: List of the newly created resources. See Resources Element, page 257.

dataSourceType: Type of the data source created.

Faults

IllegalArgument: If the name or type is malformed, or the detail or attributes are illegal.

DuplicateName: If a custom data source type with the same name already exists.

createDataSource

Create a data source resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: An existing path to the container where the new data source metadata is to be placed.

name: Name to give to the new data source.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

dataSourceType: The type of data source to create. Invoke the operation getDataSourceTypes, page 120 to get a list of valid data source types. Valid values include, but are not limited to, the following (excluding descriptions in parentheses):

• CustomProc	• Netezza
• DataDirect_Mainframe	• Oracle (for Oracle 9i and 10g (Thin Driver) resources)
• DB2 (for DB2 v7 (Type 4) resources)	• Oracle_Type2 (for Oracle 9i and 10g (OCI Driver) resources)
• DB2_Mainframe (for DB2 z/OS v8 (Type 4) resources)	• SqlServer
• DB2_Type2 (for DB2 v7 (Type 2) resources)	• Sybase
• File	• Teradata
• FileCache	• VirtualRelational
• Informix (for Informix 9.x resources)	• VirtualWsdl
• Ldap	• Wsdl
• MsAccess	• XmlFile
• MsExcel	• XmlHttp
• MySql	

annotation (optional): A description of the resource.

attributes (optional): Connection information and other information as defined by the data source type. Although this is optional, most data sources require that some attributes be provided when created. If a required attribute is missing and the attribute has no default value, a NotAllowed fault is generated. See Attributes Element, page 248.

Response Elements

resources: List of the newly created resource. See Resources Element, page 257.

Faults

DuplicateName: If a resource already exists with the given path and name.

IllegalArgument: If the path is malformed or the detail is not a legal value.

IllegalArgument: If an unsupported attribute is provided.

NotAllowed: If a required attribute is missing.

NotAllowed: If it is not legal to create the resource using the given path and name. Data sources cannot be created inside other data sources. Only COMPOSITE_DATABASE typed sources can be created in /services/databases, and only COMPOSITE_SERVICE typed sources can be created in /services/webservices (or within folders under this location). No other type of data source can be created under /services, nor can these data source types be created at any other location.

NotAllowed: If an attempt is made to create a custom Java procedure or custom data source with an insufficient license.

NotFound: If any portion of path does not exist.

NotFound: If the requested data source type does not exist.

Security: If the user does not have READ access on all items in path except the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

createDBHealthMonitorTable

Create table for DB Health Monitor.

Location

/services/webservices/system/admin/resource/operations

Request Elements

tablePath: Path of data source, in TDV format, in which to create the table.

tableName: Name to give the table.

Response Elements

N/A

Faults

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

createDomain

Create a domain. For some domains this creates a /users/domainName folder resource.

The set of valid domain types can be acquired using getDomainTypes, page 124. The set of required and optional attributes for creating the domain can be acquired using getDomainTypeAttributeDefs, page 123.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "outputs" element is set. A value of FALSE is not supported.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The name of the domain to create.

domainType: The domain type name.

isBlocking: If TRUE, the domain is created when this operation returns. A value of FALSE is not supported.

annotation (optional): A description of the domain.

attributes (optional): List of domain type specific attributes. The required attributes vary by domain type. See Attributes Element, page 248.

Response Elements

status:

- SUCCESS if the domain has been created.
- FAIL if the domain failed to be created.
- INCOMPLETE if the domain is still being created. An INCOMPLETE domain can be canceled using cancelCreateDomain, page 47.

Faults

DuplicateName: If a domain with the same name already exists.

IllegalArgument: If any of the given types or attributes are not valid.

NotAllowed: If additional domains of the given domain type cannot be created.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain type does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

createExportArchive

Create and prepare an archive for export. The archive is defined by its settings. The settings define exactly what is exported.

This operation must be called within an explicit transaction context wrapping its usage and that of other archive operations between beginTransaction, page 44 and closeTransaction, page 57 calls. An export archive is created on the server using this operation. After it is created, the export archive can be further manipulated using getArchiveContents, page 102, getArchiveExportSettings, page 105, and updateArchiveExportSettings, page 196. Multiple calls to these three operations can be made in any order to analyze and redefine what is to be exported. When satisfied, the client then calls getArchiveExportData, page 104 one or more times.

After getArchiveExportData is called, updateArchiveExportSettings can no longer be called.

When the last of the export data is returned by getArchiveExportData, or when cancelArchive, page 46 is called, the server's concept of the archive no longer exists, and further calls to any of these operations results in a NotFound faults because the archiveId is invalid.

The archive contains all resources listed in the resources elements. Each resource has an optional includeChildren element, which defaults to TRUE if unset. If a user exports a resource that contains a child resource that the user does not have READ permission for, the child resource is omitted. (The export succeeds without generating a security fault, but the resource is not exported.) If this element is unset, no resources are exported. The "all" subelement can be used to include all resources on the server.

The archive contains definitions of all the domains, users, and groups identified in the users element. If this element is unset, no domains, users, or groups are exported. The “all” subelement can be used to includes all domains, users, and groups defined by the server. The archive contains all of the server attributes identified in the serverAttributes element. If this element is unset, no server attributes are exported.

The “all” subelement can be used to export all server attributes. See getServerAttributeDefs, page 157 for server attribute definitions and getServerAttributes, page 158 for server attribute values.

A number of additional options can be specified in the exportOptions element. If an option is specified, the associated information is exported. Otherwise it is not. By default none of these options are exported. The valid options are:

- INCLUDE_CACHING: Include caching configurations for resources.
- INCLUDE_CUSTOM_JAVA_JARS: Include custom Java JARs in the export. (ADMIN ONLY)
- INCLUDE_STATISTICS: Include any resources stats known about objects including the table boundaries, and column boundaries.
- INCLUDE_DEPENDENCY: Gather and include all dependent resources for the resources you choose to export.
- INCLUDE_PHYSICAL_SOURCE_INFO: Include sensitive connection information for included physical sources. (OWNER ONLY)
- INCLUDE_REQUIRED_USERS: Include the information about the required users in the export file.
- INCLUDE_SECURITY: Include resource privilege settings. (OWNER ONLY)

If the caller requests an option marked ADMIN ONLY and does not have admin privileges, a Security fault is generated.

If the caller requests an option marked OWNER ONLY, that option is applied only to resources where the caller is the owner. If the caller has admin privileges, the option is applied to all resources regardless of ownership. Messages are generated, but no fault occurs during export.

The importHints element contains information that can be used by the client performing an import of this exported data at a later time. This allows creation of an export archive that has built-in preconceptions of what resources and users should be rebound on import, as well as what resource attributes should be remapped (such as database connection information).

Location

/services/webservices/system/admin/archive/operations/

Request Elements

settings: Description of how much information to export. The settings have the following structure:

- name: The name of the export archive.
- description: A verbose description of the archive.
- type:
  - BACKUP: All information in this archive replaces the server information when imported.
  - ROOT: Resources within the archive cannot be relocated when reimported.
  - PACKAGE: Resources within the archive can be relocated when reimported.
- resources (optional): List of resources to export.
  - all (optional): If set, all resources on the server are exported.
  - resource (0 or more): List of path-type pairs for the individual resources to export:

includeChildren (optional): If TRUE or unset, recursively include all children of this resource in the export. If FALSE, do not include any children.

- users (optional): List of users to export. See Users Element, page 260
- serverAttributes (optional): List of server attributes to be exported.
  - all (optional): If set, all server attributes are exported.
  - attributes (optional): A space-delimited list of names of the server attributes to be exported.
- exportOptions (optional): A space-delimited list of archive options.
- importHints (optional): Hints that can be used during import.
  - rebindResources: List of resources that should be rebound on import.
  - rebindUsers: List of users that should be mapped to other users on import.
  - remapAttributes: List resource attributes that should be mapped during import.
- createInfo (optional): Any setting of this element is ignored.

Response Elements

archiveId: The archive ID. This is used by other archive operations to manipulate this archive.

Faults

IllegalArgument: If the type is malformed.

IllegalArgument: If any of the resource paths or types are malformed.

IllegalArgument: If any of the settings are malformed or contain illegal values.

IllegalArgument: If any of the server attributes are malformed.

IllegalArgument: If any of the export options are malformed.

IllegalArgument: If any of the import hints are malformed.

IllegalState: This operations can only be called within an explicit transaction context. Use beginTransaction and closeTransaction.

NotAllowed: If an explicitly named resource cannot be exported. The inclusion of implicitly identified resources, using includeChildren, that are not allowed to be exported, does not cause this fault.

NotFound: If any portion of any of the resource paths and types does not exist.

NotFound: If any of the domains, users, or groups do not exist.

NotFound: If any of the server attributes do not exist.

NotFound: If any of the resources specified in the importHints are not included in the export archive.

NotFound: If any of the users specified in the importHints are not included in any of the resources, privileges, or user data in the export archive.

Security: If the caller does not have READ access on all items in the explicitly identified resource paths. Paths to resources implicitly included, using includeChildren, that the caller does not have READ on, do not generate this fault.

Security: If the caller does not have admin privileges and attempts to use an export option that is ADMIN ONLY.

Security: If the caller attempts to use an OWNER ONLY export option does not have admin privileges and attempts to use an export option that is ADMIN ONLY.

createGroup

Create a new group within a domain. If the domain is an EXTERNAL domain, the group is a read-only reference to that domain's group. In other words, when the group is destroyed, only the reference to the group is destroyed.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The name of the domain in which to create the group.

groupName: The group name.

explicitRights (optional): A bit mask for a group's rights. For a table of values, see User and Group Rights Mask, page 259.

annotation (optional): A description of the group.

Response Elements

N/A

Faults

DuplicateName: If a group with the same name already exists.

IllegalArgument: If the group name is not valid within the domain.

NotAllowed: If the domain does not allow groups to be created in it.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

Security: If the group does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

createImportArchive

Create and prepare an archive for import.

The data element contains the data aggregated from prior calls to getArchiveExportData, page 104. CreateImportArchive must be called within an explicit transaction context wrapping its usage and that of other archive operations between beginTransaction, page 44 and closeTransaction, page 57 calls.

This operation creates an import archive on the server. After it is created, the import archive can be further manipulated using getArchiveContents, page 102, getArchiveImportSettings, page 108, and updateArchiveImportSettings, page 198. Multiple calls to these four operations can be made in any order to analyze and redefine what portion of the data should be imported. When satisfied, the client then calls performArchiveImport.

If performArchiveImport, page 177 was called without blocking, subsequent calls to getArchiveImportReport, page 106 can be made to determine whether the import has completed or not. After performArchiveImport is called, updateArchiveImportSettings can no longer be called. When a call to performArchiveImport completes with a SUCCESS or FAIL status, or when cancelArchive, page 46 is called, the server's concept of the archive no longer exists, and further calls to any of these operations results in NotFound faults because the archiveId is invalid.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

data: The archive data to import. This is the data that was created by a prior call to getArchiveExportData.

Response Elements

archiveId: The archive ID.

Faults

IllegalArgument: If any of the data is malformed.

NotSupported: If the version of the archive data is not supported.

createLink

Create a link in the /services directory that points to a TABLE or a PROCEDURE resource. Links effectively publish a resource so that it can be consumed by external clients.

A link can be created even if the target resource does not exist. No validation of the existence of the target resource is performed by this procedure. Privileges must also be set appropriately and independently for use of the linked resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: Place where the link is to be created. Links can only be created in the */services* container and within a Data Service, schema, or operations resource.

name: The name of the link.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

targetPath: The fully qualified path of the resource that the new link points to.

targetType: Valid targets specified by targetPath can be either TABLE or PROCEDURE.

annotation (optional): A description of the link.

Response Elements

resources: List of the newly created resources. The resource:impactlevel can be checked when detail is set to SIMPLE or FULL for some target validity information in the response. See Resources Element, page 257.

Faults

DuplicateName: If a resource of any type already exists with the given path and name.

IllegalArgument: If either path or targetPath is malformed, or either detail or targetType is illegal.

NotAllowed: If it is not legal to create the resource using the given path and name. Links can only be created under */services* and within RELATIONAL_DATA_SOURCE, SCHEMA, and OPERATION resources.

NotAllowed: If the targetType is not allowed to be linked. Only TABLE and PROCEDURE resources can be used as the target of a link.

NotFound: If any portion of the path does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

createLinksRecursively

Create links for each of the resources within the source path recursively into the target container. Container resources are not linked; instead, new resources are created within path to match all containers within the source path. The specific type of container create in path depends on its location within path.

Only resources that can be linked are linked. Resources that cannot be linked or are not containers will be linked. If includeRoot is TRUE, the source resource is the one resource that is recursively traversed. If includeRoot is FALSE, the source path must be a container and the contents of this container are used as a list of sources.

If a resource already exists at the targetPath, the behavior of this operation depends on the createMode as follows:

- ALTER_NAME_IF_EXISTS: If a resource in targetPath already exists within path with the same name, avoid conflicts by automatically generating a new name. This only applies to the root if includeRoot is TRUE.
- FAIL_IF_EXISTS: Fail if a resource in the targetPath already exists within path. No resources are created if this occurs. This only applies to the root if includeRoot is TRUE.
- OVERWRITE_MERGE_IF_EXISTS: If a resource in targetPath already exists within path with the same name, recreate the resource within path. If the resource is a container, merge the contents of the container with the corresponding resource in path. Any link resources with conflicting names are recreated.
- OVERWRITE_REPLACE_IF_EXISTS: If a resource in targetPath already exists within path with the same name, recreate the resource within path. If the resource is a container, destroy and recreate the container within path. Only RELATIONAL_DATA_SOURCE resources can be used as the source. The target can be /services/databases if includeRoot is TRUE; it can be a COMPOSITE_DATABASE under /services/databases if includeRoot is FALSE.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A path to the resource to use as a source.

type: The type of the source resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

targetContainerPath: The path of the container resource to create links into.

createMode: One of the following: ALTER_NAME_IF_EXISTS, FAIL_IF_EXISTS, OVERWRITE_MERGE_IF_EXISTS, or OVERWRITE_REPLACE_IF_EXISTS.

includeRoot: If TRUE, create a resource with the same name as the source within the target container. If FALSE, create links to the children of the source container into the target container.

copyAnnotations: If TRUE, all created link resources within path inherit their annotations from the corresponding resource in targetPath. If FALSE, no annotations are provided for created links. Container annotations are not created or modified.

Response Elements

resources: List of the newly created or updated resources. See Resources Element, page 257

Faults

DuplicateName: If a resource already exists with the same path and type as one of the source resources, and createMode is FAIL_IF_EXISTS.

IllegalArgument: If any of the given paths, types, or detail levels are malformed.

NotAllowed: If includeRoot is TRUE and the target container is not */services*, or if includeRoot is FALSE and the target container is not a COMPOSITE_DATABASE under */service/databases*.

IllegalState: If the source is not a RELATIONAL_DATA_SOURCE.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have READ access on the items in the targetContainerPath.

Security: If the user does not have WRITE access to the last item in targetContainerPath.

Security: If the user does not have WRITE access to a resource that is to be recreated when in one of the OVERWRITE modes.

Security: If the user does not have the ACCESS_TOOLS right.

createResource

Create a resource in a default state. Resources cannot be created under /services unless otherwise noted, and cannot be created within a physical data source.

See the table in TDV Resource Types and Subtypes, page 261 for a list of the resource type-subtype element combinations supported by this operation.

- CONTAINER / CATALOG_CONTAINER: Can only be created within a data source under /services/databases.
- CONTAINER / PORT_CONTAINER: Can only be created within a SERVICE under /services/webservices.
- CONTAINER / SCHEMA_CONTAINER: Can only be created within a CATALOG that is under /services/databases.
- CONTAINER / SERVICE_CONTAINER: Can only be created within a TDV Web Services data source that is under /services/webservices.
- CONTAINER / FOLDER_CONTAINER: Cannot be created anywhere under /services except in another FOLDER under /services/webservices.
- DEFINITION_SET / SQL_DEFINITION_SET
- DEFINITION_SET / XML_SCHEMA_DEFINITION_SET
- DEFINITION_SET / WSDL_DEFINITION_SET
- PROCEDURE / BASIC_TRANSFORM_PROCEDURE: Created with no target procedure and no output columns, so it is not runnable.
- PROCEDURE / CUSTOM_PROCEDURE: Created empty, with no associated data source, so it is not runnable.
- PROCEDURE / EXTERNAL_SQL_PROCEDURE: Created with no SQL text, so it is not runnable.
- PROCEDURE / SQL_SCRIPT_PROCEDURE: Created with a simple default script body that is runnable.
- PROCEDURE / XQUERY_PROCEDURE: Created with no XQuery text, so it is not runnable.
- PROCEDURE / XSLT_PROCEDURE: Created with no XSLT text, so it is not runnable.
- PROCEDURE / STREAM_TRANSFORM_PROCEDURE: Created with no target procedure and no output columns, so it is not runnable.
- PROCEDURE / XQUERY_TRANSFORM_PROCEDURE: Created with no target schema and no model, so it is not runnable.

- PROCEDURE / XSLT_TRANSFORM_PROCEDURE: Created with no target procedure and no output columns, so it is not runnable.
- TABLE / SQL_TABLE: Created with no SQL text or model, so it is not runnable.
- TRIGGER / NONE: Created disabled.
- CONNECTOR / JMS: Created with no connection information.
- CONNECTOR / HTTP: Created with no connection information.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A path to the container with which to place the resource.

name: The name of the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

type: The type of resource to create. See the list above.

subtype: The subtype of resource to create. See the above list.

annotation (optional): A description of the resource.

resource (optional): An entire resource.

resourceBundle (optional).

includeOwnership (optional): If resource is supplied, apply the ownership.

Response Elements

resources: List of the newly created resources. See Resources Element, page 257. The amount of detail returned varies depending on the value specified for detail in the request.

Faults

DuplicateName: If a resource already exists with the given path and name.

IllegalArgument: If any of the given paths, types, or detail levels are malformed.

NotAllowed: If it is not legal to create the resource at the specified path.

NotAllowed: If an attempt is made to create a custom Java procedure, SQL script, or a trigger, with an insufficient license.

NotFound: If any portion of path does not exist.

Security: If the user does not have READ access on all items in path except the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

createUser

Create a new user within a domain. For some domains this creates a home folder for the user under /users/domainName/userName. The password element is ignored when creating users in a non-COMPOSITE domain. If the domain is EXTERNAL, a read-only reference to the user is created.

Note: By design, this operation returns a NotAllowed fault if you attempt to use it to create a user in the LDAP domain or any other external domain.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name to create the user in.

userName: The user name.

password (optional): The user's password. This is optional for some domains. This is silently ignored for non-COMPOSITE domains.

explicitRights (optional): A bit mask for a user's rights. For a table of values, see User and Group Rights Mask, page 259.

annotation (optional): A description of the user.

Response Elements

N/A

Faults

DuplicateName: If an user with the same name already exists.

IllegalArgument: If the user name is not valid within the domain.

NotAllowed: If the domain is external, such as an LDAP domain.

NotAllowed: If the domain does not allow users to be created.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

destroyConnector

Destroy a connector.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

name: The name of the connector.

Response Elements

destroyedAll: TRUE if the connector was completely destroyed; otherwise FALSE.

Faults

IllegalState: If the resource is not allowed to be destroyed.

NotFound: If the resource or any portion of the path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the resource.

Security: If the user does not have the ACCESS_TOOLS right.

destroyCustomDataSourceType

Destroy a custom data source type.

Location

/services/webservices/system/admin/resource/operations/

Request Element

dataSourceTypeName: the name of the custom data source type to destroy.

Response Elements

destroyed: Whether or not connector destruction was successful.

Faults

NotFound: If the data source type does not exist.

destroyDomain

Destroy a domain. This destroys all groups and users within the domain, and destroys the /users/domainName folder if it exists.

Location

/services/webservices/system/admin/user/operations/

Request Element

domainName: The domain name.

Request Example

```
<user:destroyDomain
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domainName>sample_domain</user:domainName>
</user:destroyDomain>
```

Response Elements

N/A

Faults

NotAllowed: If the domain cannot be destroyed. For example, the composite domain cannot be destroyed.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

destroyGroup

Destroy a group in a domain. The users within the group no longer belong to the destroyed group. If the group's domain is an EXTERNAL domain, only the read-only reference to the group is deleted. The group still exists within that domain, but it is not visible to the TDV Server.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

groupName: The name of the group to destroy.

Request Example

```
<user:destroyGroup
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domainName>composite</user:domainName>
  <user:groupName>backup_admins</user:groupName>
</user:destroyGroup>
```

Response Elements

N/A

Faults

NotAllowed: If the group cannot be destroyed. For example, the "all" group in the composite domain cannot be destroyed.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the group does not exist.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

destroyResource

Destroy a resource and all resources that might be in that container as well. The force element defines the behavior for containers. If force is FALSE, no resources are destroyed unless all resources within the container can be destroyed. If force is TRUE, as many resources are destroyed as possible, but if any resource cannot be destroyed, its container is not destroyed, either.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A source path of the resource to be destroyed.

type: The type of the source resource to be destroyed.

force: If TRUE, attempt to destroy as many resources as possible. If FALSE, do not destroy any resource if there exists at least one that is not destroyable.

Response Elements

destroyedAll: If TRUE, all of the resources were completely destroyed, including the contents of containers; otherwise FALSE.

Faults

IllegalArgument: If the path or type is malformed.

IllegalState: If the resource is not allowed to be destroyed.

NotFound: If the resource or any portion of the path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the resource.

Security: If force is FALSE and the resource is a container and the user does not have WRITE access to any resource within the container.

Security: If the user does not have the ACCESS_TOOLS right.

destroyResources

Destroy several resources. Containers are recursively destroyed.

The force element defines the behavior for containers. If force is FALSE, no resources are destroyed unless all resources within the container can be destroyed. If force is TRUE, as many resources are destroyed as possible, but if any resource cannot be destroyed, the container that resource is in is not destroyed.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of path-type pairs for the resources to be destroyed.

force: If TRUE, attempt to destroy as many resources as possible. If FALSE, do not destroy any resource if there exists at least one that is not destroyable.

Response Elements

destroyedAll: If TRUE, all of the resources were completely destroyed including the contents of containers; otherwise FALSE.

Faults

IllegalArgument: If any of the given paths or types are malformed.

IllegalState: If any of the resources are not allowed to be destroyed.

NotFound: If any of the resources or any portion of their paths do not exist.

Security: If the user does not have READ access on all items in paths other than the last one.

Security: If the user does not have WRITE access to the resources.

Security: If force is FALSE and the resource is a container and the user does not have WRITE access to any resource within the container.

Security: If the user does not have the ACCESS_TOOLS right.

destroyUser

Destroy a user in a domain. The user is removed from any group it belonged to. The user's home folder, /users/domainName/userName, is destroyed if it exists. All resources owned by the user that were not in the user's home folder are transferred to be owned by user nobody in domain composite. If the user's domain is an EXTERNAL domain, only the read-only reference to the user is destroyed. The user still exists within that domain, but is not visible to the TDV Server.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The user's domain name.

userName: The name of the user to destroy.

Response Elements

N/A

Faults

NotAllowed: If the user cannot be destroyed. For example, the admin user in the composite domain cannot be destroyed.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the specified user does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

executeNativeSql

Execute the provided sqlText directly within the data source at dataSourcePath. This can only be used with data sources that support the direct submission of SQL. This cannot be used with published data services.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "result" element is set.

If set to FALSE, then this operation will return when the processing associated with the execution has completed, but it will not wait to fill the "result" element with data.

If includeMetadata is TRUE, the response includes the metadata element. The metadata element describes the names and types of the column data that return in the result, in this call or in a later call to getTabularResult, page 161.

If maxRows is set, result contains at most maxRows number of rows. If maxRows is fewer than the total number of rows of data available, additional calls to getTabularResult need to be made to get the rest of the available data. Use hasMoreRows element in result to determine if additional data is available. This is more accurate than comparing the number of rows returned with maxRows because the server might opt to return fewer than maxRows.

If users or groups is set, queries against system tables return resources that are accessible by the users and groups in these lists. If they are not set, system table queries return resources that are accessible by the current user. The current user must have the READ_ALL_RESOURCES privilege to set these parameters.

In the response, the completed element returns TRUE if all of the results associated with the execution have been exhausted.

The completed element reports whether all possible results have been retrieved. The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request has been created, but is not yet running.
- WAITING: The request is waiting in a queue for the server to process the request.
- RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

All system resources associated with this operation are closed when the transaction containing the call completes. This means that resultId is not valid if it is used outside of the transaction that called this execute operation. If you want to use resultId, you must surround the calls to this execute operation and any operations using resultId with an explicit transaction.

Only users with the ACCESS_TOOLS right can call this operation.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

sqlText: The SQL to execute.

isBlocking: If TRUE (default), do not return until execution completes.

includeMetadata (optional): If TRUE, the response contains information about the column names and their types. Defaults to FALSE.

skipRows (optional): The number of rows to skip in the execution output before generating results. If not set, no rows are skipped. If set, the specified number of rows is skipped in the execution output before returning any results. If skipRows is greater than the total possible number of rows, no rows are returned.

maxRows (optional): The maximum number of rows to return. If not set, all rows are returned.

consumeRemainingRows (optional): If set and TRUE, all remaining rows after maxRows are consumed.

users: (optional) For system table queries, returns results that are accessible by the users included in this list. Each user is specified by a name and optional domain.

groups: (optional) For system table queries, returns results that are accessible by the groups included in this list. Each group is specified by a name and optional domain.

dataSourcePath: The path to the data source.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request performing the execution.

metadata (optional): Table metadata listing the column names and types within the result.

- column: See Column Element, page 249.

rowsAffected (optional): If known, the number of rows affected by the execution (otherwise unset). This includes a count of rows that were skipped (see skipRows element) or consumed (see consumedRemainingRows element).

result (optional): The result data.

hasMoreRows: TRUE if the table has more rows than the number affected (rowsAffected, above).

totalRowCount: Total number of rows in the table.

rows: Type-value pair for each row.

resultId: A handle to the result in the server. Valid only if it is used inside of the transaction that called this execute operation.

Use this with getTabularResult, page 161 and closeResult, page 56.

requestId: The server request ID associated with execution.

Faults

IllegalArgument: If any elements are malformed.

NotAllowed: If the data source does not support SQL execution.

NotFound: If the data source at the given path does not exist.

RuntimeError: If an error occurs during execution.

Security: If the user does not have the ACCESS_TOOLS right and READ, WRITE and other appropriate privileges on the data source.

executeProcedure

Execute the resource located at the given path and type. Only PROCEDURE and TABLE resources can be executed. If dataServiceName is provided, the path must use dotted-path notation to specify a resource relative to the published data service. If not, the path must use absolute TDV slash-path notation.

If "isBlocking" is set to TRUE, then the call to the procedure will not return until the processing associated with the execution has completed and the "outputs" element is set.

If set to FALSE, then the operation will return when the processing associated with the execution has completed, but it will not wait to fill the "outputs" element with data.

If includeMetadata is TRUE, the response includes the metadata element which describes the names and types of the output parameter data which is provided in the result either in this call or in a later call to getProceduralResult, page 143.

The inputs element contains a set of parameters to be inputs to the execution. Each parameter has two elements:

- definition: SQL language type for this parameter value. For example, VARCHAR(40) or BIGINT.

- value: The value of this parameter.

Inputs cannot be provided for a TABLE resource. Executing a TABLE resource is akin to showing the contents of that TABLE (SELECT * FROM <path>).

The outputs element, if set, contains exactly one output parameter, which contains a tabular resultId. Additional calls to getTabularResult, page 161 may be needed to retrieve data produced by this query.

The dataServiceName is the name of the public data service containing the resource to be executed. If dataServiceName is included, use dotted-path notation within sqlText for all resource references. If dataServiceName is not included, use TDV slash-path notation.

If users or groups is set, queries against system tables return resources that are accessible by the users and groups in these lists. If they are not set, system table queries return resources that are accessible by the current user. The current user must have the READ_ALL_RESOURCES privilege to set these parameters.

The completed element reports whether all possible results have been retrieved.

The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request has been created, but is not yet running.
- WAITING: The request is waiting in a queue for the server to process the request.
- RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

The rowsAffected element is set if it is known how many rows were affected by this execution.

The outputs element contains a set of ProcValue instance. If the output type is cursor, you should get a resultId that is a handle to the result in server. You can use it with getTabularResult and closeResult, page 56. If the out type is not cursor, you should get a real output value.

The returned resultId is a handle to the result in the server. This can be used with getProceduralResult, page 143 and closeResult. Output parameters which are cursors have a value returned which is a tabular resultId. This can be used with getTabularResult and closeResult. Closing a tabular resultId closes only that cursor, while closing a procedural resultId closes the procedure and all cursors associated with the procedure.

All system resources associated with this operation are closed when the transaction containing the call completes. This means that resultId is not valid if it is used outside of the same transaction that called this execute operation. If you want to use resultId, you must surround the calls to this execute operation and any operations using resultId with an explicit transaction.

All users can call this operation, but only users with the ACCESS_TOOLS right can omit the dataServiceName.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

isBlocking (optional): If TRUE (the default), do not return until the execution completes.

includeMetadata (optional): If TRUE, the response contains information about the output parameter names and their types. Defaults to FALSE.

inputs (optional): List of parameter values to use as input for the execution.

dataServiceName (optional): The name of the public data service.

path: The path to the resource.

type: The type of the resource.

users: (optional) For system table queries, returns results that are accessible by the users included in this list.

groups: (optional) For system table queries, returns results that are accessible by the groups included in this list.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request performing the execution.

metadata (optional): Table metadata listing the column names and types within the result.

parameter: See Parameters Element, page 255.

rowsAffected (optional): If known, the number of rows affected by the execution; otherwise unset.

outputs (optional): List of parameter values returned as output from the execution.

resultId: The result ID.

requestId: The server request ID associated with execution.

Faults

IllegalArgument: If any elements are malformed.

IllegalArgument: If any required parameters in the inputs element are missing.

IllegalArgument: If unexpected parameters are provided in the inputs element.

NotAllowed: If an attempt is made to execute a SQL script procedure with an insufficient license.

RuntimeError: If an error occurs during execution.

Security: If the user omitted the dataServiceName and does not have the ACCESS_TOOLS right.

Security: If the user does not have appropriate privileges on the resource referred to by path and type.

executeSql

Execute the provided sqlText directly within the TDV server. If dataServiceName is included, use dotted-path notation within sqlText for all resource references. If dataServiceName is not included, use TDV slash-path notation.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "result" element is set.

If set to FALSE, then this operation will return when the processing associated with the execution has completed, but it will not wait to fill the "result" element with data.

If includeMetadata is TRUE, the response includes the metadata element which describes the names and types of the column data provided in the result either in this call or in a later call to getTabularResult, page 161.

If maxRows is set, result contain at most maxRows number of rows. If maxRows is smaller than the total number of rows of data available, additional calls to getTabularResult need to be made to get the rest of the available data. Use hasMoreRows element in result to determine if additional data is available. This is more accurate than comparing the number of rows returned with maxRows because the server might opt to return fewer than maxRows.

If consumeRemainingRows is set and TRUE, all remaining rows in excess of maxRows are consumed.

If users or groups is set, queries against system tables return resources that are accessible by the users and groups in these lists. If they are not set, system table queries return resources that are accessible by the current user. The current user must have the READ_ALL_RESOURCES privilege to set these parameters.

In the response, the completed element returns TRUE if all of the results associated with the execution have been exhausted.

The completed element reports whether all possible results have been retrieved.

The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request has been created, but is not yet running.
- WAITING: The request is waiting in a queue for the server to process the request.
- RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

All system resources associated with this operation are closed when the transaction containing the call completes. This means that resultId is not valid if it is used outside of the same transaction that called this execute operation. If you want to use resultId, you must surround the calls to this execute operation and any operations using resultId with an explicit transaction.

All users can call this operation, but only users with the ACCESS_TOOLS right can omit the dataServiceName.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

sqlText: The SQL to execute.

isBlocking (optional): If TRUE (default), do not return until the execution completes.

includeMetadata (optional): If TRUE, the response contains information about the column names and their types. Default is FALSE.

skipRows (optional): The number of rows to skip in the execution output before generating results. If not set, no rows are skipped. If set, the specified number of rows is skipped in the execution output before returning any results. If skipRows is greater than the total possible number of rows, no rows are returned.

maxRows (optional): The maximum number of rows to return. If not set, the default number of rows is set to the value specified in the configuration option "SOAP Row Limit". To change this configuration option, go to Administration -> Configuration -> Server -> Configuration -> Debugging -> Webservices and change the value in the "SOAP Row Limit" option.

consumeRemainingRows (optional): If set and TRUE, all remaining rows after maxRows are consumed.

users: (optional) For system table queries, returns results that are accessible by the users included in this list.

groups: (optional) For system table queries, returns results that are accessible by the groups included in this list.

dataServiceName (optional): The name of the public data service.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request associated with the execution. See list in the introduction to this topic.

metadata (optional): Table metadata listing the column names and types within the result.

column: See Column Element, page 249.

rowsAffected (optional): If known, the number of rows affected by the execution (otherwise unset). This includes a count of rows that were skipped (see skipRows element) or consumed (see consumedRemainingRows element).

result (optional): The result data.

hasMoreRows: TRUE if the table has more rows than the number affected (rowsAffected, above).

totalRowCount: Total number of rows in the table.

rows: Type-value pair for each row.

resultId: A handle to the result in the server. Valid only if it is used inside of the transaction that called this execute operation.

Use this with getTabularResult, page 161 and closeResult, page 56.

requestId: The server request ID associated with execution.

Faults

IllegalArgument: If any elements are malformed.

RuntimeError: If an error occurs during execution.

RuntimeError: If the user does not have appropriate privileges on any resources referred to by the sqlText.

Security: If the user omitted the dataServiceName and does not have the ACCESS_TOOLS right.

executePreparedSql

Execute a prepared statement within the Composite server. The sqlText is treated as the parameterized query, and the parameterList is used to fill in the parameters.

If "dataServiceName" is provided, then all resource references within "sqlText" must use dotted-path notation to specify resources; otherwise CIS slash-path notation should be used.

If "isBlocking" is set to TRUE, then execution of SQL will not return until the processing associated with the execution has completed and the "result" element is set.

If set to FALSE, then this operation will return when the processing associated with the execution has completed, but it will not wait to fill the "result" element with data.

If "includeMetadata" is "true", then the response will include the "metadata" element which describes the names and types of the column data which will be provided in the "result" either in this call or in a later call to getTabularResult, page 161.

If "skipRows" is set, then that many number of rows will be skipped in the execution output before returning any results. If "skipRows" is greater than the total possible number of rows, then no rows will be returned.

If "maxRows" is set, then "result" will contain at most "maxRows" number of rows.

If "maxRows" is fewer than the total number of rows of data available, then additional calls to "getTabularResult" will need to be made to get the rest of the available data.

Use "hasMoreRows" element in "result" to determine if additional data is available. This is more accurate than comparing the number of rows returned with "maxRows" because the server might opt to return fewer than "maxRows".

If "consumeRemainingRows" is TRUE, then all remaining rows in excess of "maxRows" will be consumed.

If "users" or "groups" is set, then queries against system tables will return resources that are accessible by the users and groups in these lists. If they are not set, then system table queries will return resources that are accessible by the current user.

The current user must have the READ_ALL_RESOURCES privilege to set these parameters.

In the response, the "completed" element will return true if all of the results associated with the execution have been exhausted. The "completed" element reports whether all possible results have been retrieved.

The "requestStatus" element reports the status of the server request associated with the execution. The request status can be one of:

- "STARTED": The request has started. The request has been created, but is not yet running.
- "WAITING": The request is waiting in a queue for the server to process the request.
- "RUNNING": The request is currently being executed by the server.
- "COMPLETED": The execution associated with the request has completed. Results can now be acquired.
- "CLOSING": The request is closing.
- "SUCCESS": The request closed with success.
- "FAILURE":The request closed with failure.
- "TERMINATED": The request was terminated.

The "rowsAffected" element will be set if it is known how many rows were affected by this execution. This includes a count of rows that were skipped (see “skipRows” element) or consumed (see "consumedRemainingRows" element).

The returned "resultId" returns a handle to the result in the server. This can be used with the getTabularResult, page 161 and closeResult, page 56 operations.

The "requestId" is the server request ID associated with execution.

All system resources associated with this operation are closed when the transaction containing the call completes. This means that "resultId" is not valid if it is used outside of the same transaction that called this execute operation. If you want to use "resultId", you must surround the calls to this execute operation and any operations using "resultId" with an explicit transaction.

All users can call this operation, but only users with the ACCESS_TOOLS right can omit the "dataServiceName".

Request Elements

sqlText: The Parameterized SQL to be executed.

isBlocking (optional): If "true", do not return until the execution completes. Defaults to "true".

includeMetadata (optional): If "true", the response will contain information about the column names and their types. Defaults to "false".

skipRows (optional): The number of rows to skip in the execution output before generating results. If not set, then no rows will be skipped.

maxRows (optional): The maximum number of rows to return. If not set, then there all rows will be returned.

consumeRemainingRows (optional): If set to TRUE, then all remaining rows after "maxRows" will be consumed.

dataServiceName (optional): The name of the public data service.

users: (optional) For system table queries, returns results that are accessible by the users included in this list.

groups: (optional) For system table queries, returns results that are accessible by the groups included in this list.

parameterList: The list of parameters to be used for the prepared Statement.

Response Elements:

completed: If "true", all processing associated with execution has completed.

metadata (optional): Table metadata describing the column names and types within the result.

rowsAffected (optional): If known, the number of rows affected by the execution;

otherwise unset.

result (optional): The result data.

resultId: The result ID.

requestId: The request ID.

Faults:

IllegalArgument: If any elements are malformed.

RuntimeError: If an error occurs during execution.

RuntimeError: If the user does not have appropriate privileges on any resources referred to by the "sqlText".

Security: If the user omitted the "dataServiceName" and does not have the ACCESS_TOOLS right.

executeSqlScript

Execute the provided scriptText directly within the TDV server.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "outputs" element is set.

If set to FALSE, then the execution of the Sql script will return when the processing associated with the execution has completed, but it will not wait to fill the "outputs" element with data.

If includeMetadata is TRUE, the response includes the metadata element which describes the names and types of the output parameter data provided in the result either in this call or in a later call to getProceduralResult, page 143.

The inputs element contains a set of parameters to be inputs to the execution. A parameter contains two elements:

- definition: The SQL-language data type for this parameter value. For example, VARCHAR(40) or BIGINT.
- value: The value of this parameter.

If users or groups is set, queries against system tables return resources that are accessible by the users and groups in these lists. If they are not set, system table queries return resources that are accessible by the current user. The current user must have the READ_ALL_RESOURCES privilege to set these parameters.

The completed element reports whether all possible results have been retrieved.

The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request has been created, but is not yet running.
- WAITING: The request is waiting in a queue for request processing.
- RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

The outputs element contains a set of ProcValue instance. If the out type is cursor, you should get a resultId, which is a handle to the result in server. You can use it with getTabularResult, page 161 and closeResult, page 56. If the out type is not cursor, you should get real output value.

The returned resultId returns a handle to the result in the server. This can be used with getProceduralResult, page 143 and closeResult, page 56.

The value returned for output parameters that are cursors is a tabular resultId. This can be used with getTabularResult and closeResult. Closing a tabular resultId only closes that cursor, while closing a procedural resultId closes the procedure and all cursors associated with the procedure.

All system resources associated with this operation are closed when the transaction containing the call completes. This means that resultId is not valid if it is used outside of the same transaction that called this execute operation. Therefore, if you want to use resultId, you must surround the calls to this execute operation and any operations using resultId with an explicit transaction.

Only users with the ACCESS_TOOLS right can call this operation.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

isBlocking (optional): If TRUE (the default), do not return until the execution completes.

includeMetadata (optional): If TRUE, the response contains information about the output parameter names and their types. Defaults to FALSE.

inputs (optional): List of definitions and values for parameters to use as execution input.

scriptText: The SQL to be executed.

users (optional): For system table queries, returns results that are accessible by the users included in this list.

groups (optional): For system table queries, returns results that are accessible by the groups included in this list.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request performing the execution.

metadata (optional): Table metadata listing the column names and types within the result.

column: See Column Element, page 249.

rowsAffected (optional): If known, the number of rows affected by the execution; otherwise unset.

outputs (optional): A set of parameter values returned as output from the execution.

result: The result ID.

requestId: The request ID.

Faults

IllegalArgument: If any elements are malformed.

IllegalArgument: If any required parameters in the inputs element are missing.

IllegalArgument: If unexpected parameters are provided in the inputs element

NotAllowed: If an attempt is made to use this operation with an insufficient license. RuntimeError: If an error occurs during execution.

RuntimeError: If the user does not have appropriate privileges on any resources referred to by the scriptText.

Security: If the user does not have the ACCESS_TOOLS right.

getAllResourcesByPath

Get all of the specified resources for a given path.

Resources are normally uniquely identified by both the path and type of the resource. This operation returns all resources at a given path regardless of type. Multiple resources are returned if a path has multiple resources that differ by type—for example, */shared/examples/ds_inventory* (data source) and */shared/examples/ds_inventory* (view).

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to get.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the resources at the given path. See Resources Element, page 257.

Faults

IllegalArgument: If the path or detail are malformed.

NotFound: If no resources at the given path exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

getAncestorResources

Get all of the ancestors of the specified resource up to and including the root resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the resource.

type: The type of the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the ancestor resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path, type, or detail are malformed.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

getArchiveContents

Get the contents of the archive. The contents contains an explicit list of everything that exists within the archive.

Use this operation with either an export or import archive. Use it anytime after createExportArchive, page 69 or createImportArchive, page 73 is called, as long as it is within the same transaction and the archive has not been closed.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the archive whose contents to retrieve.

Response Elements

contents: The contents of the archive.

- resources (optional): List of resources within the archive.
- users (optional): An explicit list of domains, users, and groups within this archive. This is similar to Users Element, page 260, but some elements are illegal, as marked below.

  — all (illegal): This element is never set in this context.

  — domains (optional): List of domains included within the archive.

  all (illegal): This element is never set in this context.

  domains: A space-delimited list of domain names of the domains.

  — users (optional): List of users within the archive.

  domain (0 or more): A domain that contains users within the archive.

  all (illegal): This element is never set in this context.

  name: The name of the domain containing the users.

  users (optional): A space-delimited list of user names of the users within the archive.

  — groups (optional): List of groups within the archive. Includes user membership.

  domain (0 or more): A domain that contains groups within the archive.

  name: The name of the domain containing the groups.

  all (illegal): This element is never set in this context.

  groups (optional): List of groups and their definitions within the archive.

  name: The name of the group.

  all (illegal): This element is never set in this context.

  user (optional): List of users to be listed as members of the group.
- serverAttributes (optional): A space-delimited list of server attributes (name elements) within the archive.
- exportOptions (options): A space-delimited list of options (name elements) used to export additional data into the archive.

Faults

NotFound: If the archiveId does not exist. This can occur if:

- the archive was previously canceled using this procedure
- the getExportData returned the last chunk of data

- getImportReport or performImportArchive previously returned a FAIL or SUCCESS status
- this operation is called on a different transaction

getArchiveExportData

Export some or all of the archive data. This operation performs the actual export of the archive created using createExportArchive, page 69. After this operation is called, updateExportSettings can no longer be called. If maxBytes is unset, this operation returns the entire archive in one call.

If the status is INCOMPLETE, additional calls to getArchiveExportData, page 104 must be made to get the remainder of the archive.

If the status is SUCCESS, cancelArchive, page 46 is invoked, or the bounding transaction is closed, the archive process is closed and the archive ID becomes invalid. It is generally recommended that you always use maxBytes, because the actual size of the export data cannot be determined ahead of time and it can be very large depending on what is being exported.

This operation can only be used with an export archive. It can be used anytime after createExportArchive is called, as long as it is within the same transaction and the archive has not been closed.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the export archive.

maxBytes (optional): The maximum number of bytes to retrieve for the archive in this call. This must be a positive number.

Response Elements

status:

- CANCELED: If the archive was canceled during this call.
- SUCCESS: If the export has completed.
- INCOMPLETE: If there is more archive data to be retrieved.

archiveReport (optional): List of messages generated during this call, if any. If no messages exist, this element is unset. See Messages Element, page 255.

data (optional): The archive data.

Faults

IllegalArgument: If maxBytes is not a positive number.

IllegalState: If this operation is called using an import archive ID.

NotFound: If archive for the archive ID does not exist.

getArchiveExportSettings

Get the export settings used or to be used by an archive export.

This operation can be used with either an export or import archive. It can be used anytime after createExportArchive, page 69 or createImportArchive, page 73 is called, as long as it is within the same transaction and the archive has not been closed.

Location

/services/webservices/system/admin/archive/operations/

Request Element

archiveId: The ID of the archive.

Response Elements

settings: Description of what was or will be exported for this archive. The settings have the following structure:

- name: The name of the export archive.
- description: A verbose description of the archive.
- type:
  - BACKUP: All information in this archive replaces the server information when imported.
  - ROOT: Resources within the archive cannot be relocated when reimported.
  - PACKAGE: Resources within the archive can be relocated when reimported.

- resources (optional): List of exported resources.
  - all (optional): If set, all resources on the server were or will be exported.
  - resource (0 or more): List of individual exported resources.

  path: The path to the resource.

  type: The type of the resource.

  includeChildren (optional): If TRUE or unset, recursively export all child resources. If FALSE, do not include any children.
- users (optional): List of exported users. See Users Element, page 260.
- serverAttributes (optional): List of exported server attributes.
  - all (optional): If set, all server attributes were or will be exported.
  - attributes (optional): A space-delimited list of exported server attribute names. See Attributes Element, page 248.
- exportOptions (optional): A space-delimited list of archive options (name elements).
- importHints (optional): Hints that can be used during import. See Import Hints, page 252.
- createInfo (optional): Information about the creation of the exported archive.
  - archiveDomain: The domain of the user that created this archive.
  - archiveUser: The name of the that created this archive.
  - archiveVersion: The version of the archiver used to create this archive.
  - createDate: When the archive was created.
  - sourceJvm: What Java Virtual Machine was used to create this archive.
  - sourceOperationSystem: What operating system was used to create the archive.

Faults

NotFound: If archive for the archive ID does not exist.

getArchiveImportReport

Get the current status of an in-progress import.

This operation can only be used with an import archive. It can be used any time after createImportArchive, page 73 is called, as long as it is within the same transaction and the archive has not been closed. Especially if performArchiveImport, page 177 was called without blocking, subsequent calls to getArchiveImportReport, page 106 can be made to determine whether the import has completed or not. It only gets the messages that is generated if import completed.

If an import status is CANCELED or FAIL, the import is in an inconsistent state. The bounding transaction should be rolled back before performing other operations.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the archive.

isBlocking If TRUE, this call does not return until the import has completed. If FALSE, this call returns immediately. Only TRUE is supported.

Response Elements

status:

- CANCELED: The archive was canceled during this call.
- SUCCESS: The import has completed.
- INCOMPLETE: The import is still in progress.
- FAIL: The import failed during or prior to this call.

archiveReport (optional): List of messages generated since the previous call to performArchiveImport or getArchiveImportReport, if any. If no messages exist, this element is unset. See Messages Element, page 255 for message structure.

Faults

IllegalArgument: If the isBlocking element is malformed.

IllegalState: If this operation is called using an export archive ID.

NotFound: If archive for the archive ID does not exist.

getArchiveImportSettings

Get the import settings that are to be used for controlling the archive import.

This operation can only be used with an import archive. It can be used anytime after createImportArchive, page 73 is called, as long as it is within the same transaction and the archive has not been closed.

A number of options can be specified in the importOptions element. If an option is specified, the associated information is exported; otherwise, it is not. By default, if the caller has admin privileges, the setting of the importOptions is that of the exportSettings. If the caller does not have admin privileges, the default is the same except that any items marked as ADMIN ONLY are not included.

The valid options are:

- INCLUDE_CACHING: Include caching configurations for resources.
- INCLUDE_CUSTOM_JAVA_JARS: Include custom Java JARs in the export. (ADMIN ONLY)
- INCLUDE_STATISTICS: Include any resources statistics known about things including the table boundaries, and column boundaries.
- INCLUDE_DEPENDENCY: Gather and include all dependent resources for the resources you choose to export.
- INCLUDE_PHYSICAL_SOURCE_INFO: Include sensitive connection information for included physical sources. (OWNER ONLY)
- INCLUDE_REQUIRED_USERS: Include the information about the required users in the export file.
- INCLUDE_SECURITY: Include resource privilege settings. (OWNER ONLY)

If an option is marked OWNER ONLY, that option is only be applied to resources where the caller is the owner. If the caller has admin privileges, the option is applied to all resources regardless of ownership. If the option cannot be applied, messages are generated, but no fault occurs during import.

Location

/services/webservices/system/admin/archive/operations/

Request Element

archiveId: The ID of the archive.

Response Elements

settings: Description of how much of the archive to import and what modifications should be made during import. The settings have the following structure:

- excludeResources (optional): List of resources that should not be imported. By default this is unset.
- entry (optional): Paths and types for the entries.
- relocateResources (optional): List of mappings (using path-type pairs) of resources from their location in the archive to where they should be imported. By default this is unset.
- rebindResources (optional): List of mappings (using path-type pairs) of resources references within the archive to where they should refer to. By default this is unset.
- rebindUsers (optional): List of mappings (using domainName-userName pairs) of users within the archive from who they are to who they should be. By default this is unset.
- remapAttributes (optional): List of resource attribute settings that should be applied on import. By default this is unset. Each resource has a map element that contains a path-type pair designating a resource, and an attribute element. see Attributes Element, page 248.
- importOptions (optional): Space-separated list of archive options (name elements) indicating what additional features should be imported. By default, the same options used for export are used for import.

Faults

IllegalState: If this operation is called using an export archive ID.

NotFound: If archive for the archive ID does not exist.

getAvailableLoginModuleNames

Retrieve a list of valid login modules types by name.

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Elements

loginModule (optional): List of names of the available login modules:

- name
- bundleEnabled

Faults

N/A

getCachedResourceStatisticsConfig

Get the Cost Based Optimizer (CBO) statistics configuration for a data source. Cache must be configured before configuring statistics.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a cacheable resource.

type: The type of the resource. Can only be TABLE.

Response Elements

cachedStatisticsConfig: The statistics configuration of the given resource.

- configured (optional): TRUE if statistics gathering is configured otherwise FALSE.
- useEnabled (optional): TRUE if gathered statistics are to be used by CBO; otherwise FALSE. Can be used to temporarily disable CBO.
- cardinalityMin (optional): Minimum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityMax (optional): Maximum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityExpected (optional): Expected cardinality for this resource. Setting this value overrides gathered statistics.
- gatherEnabled (optional): Defines what statistics should be gathered for this resource. Valid values are TABLE_BOUNDARY or CUSTOM.

- maxTime (optional): Integer 0 to *n* in minutes; maximum amount of time the process should spend gathering data; 0 means no limit.
- columns (optional): Only applicable if gatherEnabled is set to CUSTOM.
  - column: List indicating what specific data to get for each column.
  - name: Simple column name.
  - flags: Valid values are NONE, BOUNDARY or ALL.
  - columnMin (optional): Minimum value for this resource. Setting this value overrides gathered statistics.
  - columnMax (optional): Maximum value for this resource. Setting this value overrides gathered statistics.
  - columnDistinct (optional): Distinct value this resource. Setting this value overrides gathered statistics.
- onCacheRefresh (optional): If TRUE, statistics gathering should be automatically triggered by cache refresh. If FALSE, refresh mode can be specified.
- refresh (optional): How the statistics data should be refreshed. See Refresh Element, page 256.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getChildResources

Get all the immediate child resources of a resource. Only CONTAINER and DATA_SOURCE resources have child resources. All other resource types return an empty list.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the parent resource.

type: The type of the parent resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of child resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotFound: If any portion of the path does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getClusterConfig

Get cluster configuration data.

Location

/services/webservices/system/admin/resource/operations

Request Elements

N/A

Response Elements

clusterConfig: Cluster configuration data, including cluster display name and list of servers in the cluster.

clusterDisplayName

serverList: List of server names, host names, and ports.

Faults

IllegalState: If the server is not part of a cluster.

Security: If the user does not have both ACCESS_TOOLS and READ_ALL_CONFIG rights.

getConnectorGroup

Get all connectors within a specified group.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

name: The name of the connector group.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

connectors: List of connectors. See Connector Element, page 249.

Faults

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

getConnectorGroupNames

Get all a list of the connector group names if any connector is part of a group.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

N/A

Response Elements

names: List of the connector group names.

Faults

Security: If the user does not have READ access on all items in path other than the last one.

getConnectors

Get all connectors.

Location

/services/webservices/system/admin/resource/operations/

Request Element

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

connectors: List of the connectors. See Connector Element, page 249.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

getCreateDBHealthMonitorTableSQL

Get SQL for creating health monitor table.

Location

/services/webservices/system/admin/server/operations/

Request Elements

tablePath: Path of data source, in TDV format, in which to create the table.

tableName: Name of the table.

Response Elements

sql: Native SQL string.

Faults

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

getDataSourceAttributeDefs

Get the attribute definitions for data sources of the given data source type. These attributes definitions are used when creating and updating data sources, such as the host and port to connect to.

Also see the related operation getDataSourceTypeAttributeDefs, page 119 which defines the attributes for the data source type itself.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceType: Valid input value is a resource:name from getDataSourceTypes, page 120.

Response Elements

attributeDefs: List of attribute definitions. See Attribute Definitions Element, page 247.

Faults

NotFound: The given data source type does not exist.

Security: If the user does not have the ACCESS_TOOLS right.

getDataSourceChildResources

Deprecated as of API version 6.0. Instead use getIntrospectableResourceIdsTask, page 130 or getIntrospectedResourceIdsTask, page 134 to get this list of possible resources, , or getIntrospectionAttributes, page 136 to get the desired data. If the resource is introspected, you can use getResource, page 145. getResources, page 150. or getChildResources, page 111.

Gets information on the child resources within a data source, even if those child resources have not been introspected. Accessing data source childInfo is relatively expensive. If multiple calls to this operation and/or updateDataSourceChildInfos and/or updateDataSourceChildInfosWithFilter are going to be made, making them all on one transaction improves performance. Also see reintrospectDataSource, page 185.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source.

dsPath: The path within the data source for which children are to be found. It can be the root path or the path of any container under root.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

attributes (optional): Optional values to be used for the resource. These may be required to specify login information if such information is not persisted with the data source definition. See Attributes Element, page 248.

Response Elements

resources: List of the child resources. See Resources Element, page 257.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If path, dsPath, dsType, or the detail is malformed.

IllegalState: If the data source is disabled.

NotFound: If the data source resource or any portion of the path to the data source does not exist.

NotFound: If any portion of the dsPath within the data source does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have WRITE access on the data source.

Security: If the user does not have the ACCESS_TOOLS right.

getDataSourceReintrospectResult

Deprecated as of API version 6.0. Instead use introspectResourcesResult, page 167.

Checks for completion of a non-blocking reintrospect that was started with reintrospectDataSource, page 185. Returns no report if the reintrospect is not yet complete.

The reintrospect ID is only valid during a single transaction, so this operation can only be used within an explicit transaction that also contains reintrospectDataSource. If this operation returns a SUCCESS or FAIL status, the reintrospectId is invalidated.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

reintrospectId: The reintrospect ID provided by the reintrospectDataSource.

isBlocking: If TRUE, this operation does not return until reintrospect has completed. If FALSE, this operation returns immediately, regardless of completion.

Response Elements

status:

- SUCCESS or FAIL (as appropriate) if the reintrospect completed during or prior to this call.
- INCOMPLETE if the reintrospect is still in progress and isBlocking is FALSE.
- CANCELED if the reintrospect was canceled by a separate call during this operation.

reintrospectReport (optional): If the status is SUCCESS or FAIL, this report lists messages (changeEntry elements) describing either errors or the changes that occurred during reintrospect. Otherwise, this element does not exist. See Messages Element, page 255.

Faults

IllegalArgument: If the isBlocking element is not a valid boolean.

NotFound: If the reinstrospectId does not exist.

Security: If the user does not have the ACCESS_TOOLS right.

getDataSourceStatisticsConfig

Get the cost-based optimizer (CBO) statistics configuration for a data source.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a resource.

type: The type of the resource. May only be a relational physical DATA_SOURCE.

Response Elements

dataSourceStatisticsConfig: The statistics configuration of the given resource.

- configured (optional): TRUE if statistics gathering is configured; otherwise FALSE.
- useEnabled (optional): TRUE if gathered statistics are to be used by CBO; otherwise FALSE. Can be used to temporarily disable CBO.
- tableGatherDefault (optional): Unless overridden at table level, sets the default table configuration. Values are ALL, COLUMN_BOUNDARY, NONE or TABLE_BOUNDARY.
- numThreads (optional): Integer 1 to N; indicates how many threads should be allocated to gather statistics for this data source.
- maxTime (optional): Integer 0 to N in minutes; sets data source level default for maximum amount of time the process should spend gathering data for each table; 0 means no limit.
- refresh (optional): How the statistics data should be refreshed. See Refresh Element, page 256.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getDataSourceTypeAttributeDefs

Get the attribute definitions for the given data source type. These attribute definitions are used to interpret the attributes of the data source type itself, such as where to find a JDBC driver and what CLASSPATH to use. Also see getDataSourceAttributeDefs, page 115 which defines attributes that are used when creating and updating data source resources.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceType: Valid input value is a resource:name from getDataSourceTypes, page 120.

Response Elements

attributeDefs: List of attribute definitions. See Attribute Definitions Element, page 247.

Faults

NotFound: The given data source type does not exist.

Security: If the user does not have the ACCESS_TOOLS right.

getDataSourceTypeCustomCapabilities

Get a list of capabilities.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceTypeName: The name of the data source type.

Response Elements

customCapabilities: List of capabilities.

- attribute: See Attributes Element, page 248.

Faults

NotFound: If the data source type does not exist.

getDataSourceTypes

Get a list of the available data source types. These data source types can be used to create new data sources.

Location

/services/webservices/system/admin/resource/operations/

Request Element

detail: The level of detail about the types to include in the response. Valid values are:

- SIMPLE: Returns the name and type of the types.
- FULL: Additionally returns type-specific data source attributes, including required connection parameters and other information.

Response Elements

dataSourceTypes: List of data source types known by the server. For attributes, see Attributes Element, page 248.

Faults

IllegalArgument: If detail is not valid.

Security: If the user does not have the ACCESS_TOOLS right.

getDependentResources

Get all of the resources that depend on the given resource. A resource depends on this resource if it makes use of this resource. Also see getUsedResources, page 164, which lists the resources a resource uses.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to analyze for usage dependencies.

type: The type of the resource specified by the given path. Valid values are DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the dependent resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

getDomainGroups

Get the group definitions within a domain.

If the scope is LOCAL_ONLY, only the groups the server has local definitions for are returned. If the scope is ALL, all groups are returned. For locally defined domains, like the composite domain, both options return the same list. For domains with external storage such as LDAP, only some of the groups in the external storage may have been created as local groups, so the returned lists may differ.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name. scope: Either ALL or LOCAL_ONLY.

scope: LOCAL_ONLY or ALL.

Response Elements

groups: List of groups within the domain. See Groups Element, page 251.

Faults

IllegalArgument: If an invalid scope is provided.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getDomains

Get the definitions of all the domains that exist on a server.

This operation can be invoked by any user, even an anonymous user.

Location

/services/webservices/system/admin/user/operations/

Request Element

detail: The level of detail about the domains to include in the response. See Domains Element, page 250.

- SIMPLE: Returns name, type, and annotation.
- FULL: Returns name, type, annotation, and attributes.

Request Example

```
<user:getDomains
xmlns:user="http://www.compositesw.com/services/system/admin/user"
xmlns:common="http://www.compositesw.com/services/system/util/comm
on">
  <user:detail>SIMPLE</user:detail>
</user:getDomains>
```

Response Elements

domains: List of domains.

Response Example

This example shows a response to a request with detail=SIMPLE.

```
<user:getDomainsResponse
xmlns:user="http://www.compositesw.com/services/system/admin/user"
>
  <user:domains>
    <user:domain>
      <user:name>dynamic</user:name>
      <user:domainType>DYNAMIC</user:domainType>
      <user:annotation>Dynamic authentication
domain</user:annotation>
    </user:domain>
    <user:domain>
      <user:name>composite</user:name>
      <user:domainType>COMPOSITE</user:domainType>
      <user:annotation>Composite authentication
domain</user:annotation>
    </user:domain>
  </user:domains>
</user:getDomainsResponse>
```

Faults

IllegalArgument: If an invalid detail value is provided.

Security: If the detail is FULL and the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getDomainTypeAttributeDefs

Get the definitions of attributes that can be used for creating or updating domains of a given domain type. The valid list of domain types can be acquired using getDomainTypes, page 124.

Location

/services/webservices/system/admin/user/operations/

Request Element

domainType: The domain type name. Valid input values include LDAP, DYNAMIC, and COMPOSITE.

Response Elements

attributeDefs: List of attribute definitions. See Attribute Definitions Element, page 247.

Faults

IllegalArgument: If an invalid domain type name is provided.

NotFound: If the domain type does not exist.

Security: If the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getDomainTypes

Get the list of supported domain types.

Location

/services/webservices/system/admin/user/operations/

Request Elements

N/A

Response Elements

domainTypes: List of domain types.

- domainType:
- name
- annotation (optional)
- attributeDefs (optional): See Attribute Definitions Element, page 247.

Faults

Security: If the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getDomainUsers

Get all of the user definitions within a domain.

If the scope is LOCAL_ONLY, only the users the server has local definitions for are returned. If the scope is ALL, all users are returned. For locally defined domains like the composite domain, both options return the same list. For domains with external storage, such as LDAP, only some of the users in the external storage may have been created as local users, so the returned lists may be different.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

scope: Either LOCAL_ONLY or ALL.

Response Elements

users: List of users within the domain.

- user: The name, domain name and ID of a user, plus:
  - explicitRights. For a table of values, see User and Group Rights Mask, page 259.
  - interitedRights. For a table of values, see User and Group Rights Mask, page 259.
  - annotation (optional)
  - groupNames (optional): Names and domains of the groups to which the user belongs.

Faults

IllegalArgument: If an invalid scope is provided.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getExtendableDataSourceTypes

Get all data source types that can be extended.

Location

/services/webservices/system/admin/resource/operations/

Request Element

detail: Reserved for future changes.

Response Elements

dataSourceTypes: List of data source types known by the server.

- dataSourceType (optional):
- name
- type
- attributes (optional): See Attributes Element, page 248.

Faults

IllegalArgument: If detail is not valid. Reserved for future changes.

Security: If the user does not have the ACCESS_TOOLS right.

getGeneralSettings

Retrieve the general settings for Pluggable Authentication Modules (PAM).

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Elements

Security object, with attributes representing configuration details:

enablePAM

mapDisallowUser

logAuthFailures

logPerformance

assignModuleGroups

Faults

- N/A

getGroups

Get the definitions of the given domain groups.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

names: List of group names within the domain.

Response Elements

groups: List of groups within the domain. See Groups Element, page 251.

Faults

NotFound: If the domain does not exist.

NotFound: If any of the requested groups do not exist.

Security: If the user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getGroupsByUser

Get the definitions of all groups that contain the given user and are also within the given domain.

Location

/services/webservices/system/admin/user/operations/

Request Elements

userName: The name of the user whose groups to retrieve.

domainName: The domain of the user whose groups to retrieve.

Response Elements

groups: List of groups within the domain. See Groups Element, page 251.

Faults

NotFound: If the user does not exist.

NotFound: If the domain does not exist.

Security: If the operation user does not have the ACCESS_TOOLS and READ_ALL_USERS rights.

getIntrospectableResourceIdsResult

Get the results from getIntrospectableResourceIdsTask, page 130. The number of resources returned is limited by page size and total number of known resources. Subsequent calls to this operation incrementally return the full list of native resource identifiers available within the data source.

If the block element is set and TRUE, this operation blocks until the task is complete. Otherwise, this operation does not block.

The page size controls the maximum number of resource identifiers that are returned from this call.

The page start determines which resource is returned first. This can be used to jump ahead in the resource list. This jump is relative to the current position. Providing a non-zero page start with every call to this operation has the effect of skipping a page-size number of results oeach time. Specifying the page start sets the position that you would like to start receiving resource identifiers. Subsequent calls to this operation start from that point.

If an insufficient number of resource identifiers have been found during a call to this operation, this operation either times out or returns with COMPLETED equal TRUE. In either case, an empty list of identifiers is returned.

This operation returns a taskId that can be used to get results using getIntrospectableResourceIdsResult, page 128, or canceled using cancelServerTask, page 50.

This operation returns the total number of known resource identifiers in the totalResults element. The totalResults element does not exist until the full total is known. If this number is not known or only partial results are available, this element is empty.

This operation returns a completed element that indicates whether or not processing has completed, and these are the last results the caller receives. If TRUE, a subsequent call to getIntrospectableResourceIdsResult generates a NotFound fault.

The lastUpdate return element indicates when the cache associated with this resource identifier list was last updated. If the result set is currently being updated due to the original getIntrospectableResourceIdsTask, page 130, lastUpdate reflects when the server task was initiated.

The resourceIdentifiers element provides the list of native resources paths, types, and subtypes. Paths are relative to the containing data source.

This operation must be run within the same explicit transaction that the original call to getIntrospectableResourceIdsTask was invoked. Otherwise the taskId is reported as NotFound.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

taskId: The server task ID associated with the original call.

block: Whether or not to block until processing is complete. Defaults is FALSE.

page:

- size: The number of resource identifiers to return in the result.
- start: The page number to start retrieving data from. Defaults to 0.

Response Elements

taskId: The server task ID associated with the original call.

totalResults: If known, the total result set size. Otherwise this element does not exist.

completed: TRUE if processing is completed and the result set has been exhausted.

lastUpdate: The date and time the resource cached was last created.

resourceIdentifiers: List of native resource paths and types available within a data source.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the taskId, or page is malformed.

IllegalState: If the data source is disabled.

NotFound: If the taskId does not exist or has completed.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If a different TDV session was used to create the server task.

getIntrospectableResourceIdsTask

Create a server task to fetch all resource identifiers available for a given data source.

The first time this is called, the underlying data source is queried for its completed list of resources available within the data source. This information is cached so that future invocations of this using the same data source returns the contents of the cache rather than querying the data source. To clear this cache, use clearIntrospectableResourceIdCache, page 53. The results include a lastUpdate field that can be used to help determine whether clearing the resource identifier cache is necessary.

The list of available resources within a data source depends upon the user's access privileges within the data source.

If dsContainerId is provided, the search for introspectable resources starts at that location. If it is unset, the search starts at the data source root.

If recurse is TRUE or unset, the search for introspectable resources is performed recursively throughout the data source starting at the given dsContainerId, or at the data source root if dsContainerId is unset. If recurse is FALSE, only the immediate children of the dsContainerId or data source root are found.

The user name associated with creating the cache is placed in the cache as well. If the data source is non-dynamic the user name is drawn from the data source connection parameters. If the data source is dynamic, it is drawn from the TDV user name and domain.

Each data source has one primary resource identifier cache. If the user name associated with this call differs from the primary cache, the primary cache is marked for expiration. When the expiration period is over, the cache is cleared. This period can be modified using a TDV Server attribute. If the user name associated with the call matches a cache that is currently marked for expiration, the expiration is canceled and it becomes the primary cache. If no cache with a matching user name exists, a new cache is created by querying the data source.

This operation returns a taskId which can be used to get results using getIntrospectableResourceIdsResult, page 128 or canceled used cancelServerTask, page 50.

This operation returns the total number of known resource identifiers in the totalResults element. The totalResults element does not exist until the full total is known. If this number is not known or only partial results are available, this element is empty.

If the TDV session that invokes this operation closes before the task is completed, the task is terminated.

This operation must be run within an explicit transaction so that the taskId can be used in successive calls to getIntrospectableResourceIdsResult.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the data source.

attributes (optional): List of data source type specific attributes. The specific list of supported attributes varies by data source type. (See getDataSourceAttributeDefs, page 115.) Sets the specified attributes, but does not alter the value of unspecified attributes. See Attributes Element, page 248.

dsContainerId (optional): A container within the data source for which to find introspectable resources. If unset, the data source root path is used.

- path: Path to the container.
- type: Container type. See TDV Resource Types and Subtypes, page 261.
- subtype: Container subtype. See TDV Resource Types and Subtypes, page 261.

recurse (optional): If TRUE or unset, the search for introspectable resources is performed recursively throughout the data source starting at the given dsContainerId, or at the data source root if dsContainerId is unset. If recurse is FALSE, only the immediate children of the dsContainerId or data source root are found.

Response Elements

taskId: The ID of the server task performing the work.

totalResults (optional): If known, the total result set size. Otherwise this element does not exist.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the path or dsPath is malformed.

IllegalState: If the data source is disabled.

NotFound: If the data source or any portion of the path to the data source does not exist.

NotFound: If dsPath does not exist within the data source.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have READ access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

getIntrospectedResourceIdsResult

Get the results from getIntrospectedResourceIdsTask, page 134. The number of resource identifiers returned is limited by page size and total number of known resources.

Subsequent calls to this operation incrementally return the full list of resource identifiers introspected from the data source.

If the block element is set and TRUE, this operation blocks until the task is complete. Otherwise, this operation does not block.

The page size controls the maximum number of resource identifiers that are returned from this call.

The page start determines which resource is returned first. This can be used to jump ahead in the resource list. This jump is relative to the current position. Providing a non-zero page start with every call to this operation has the effect of skipping a page-size number of resources each time.

Specifying the page start sets the position where you would like to start receiving resource identifiers. Subsequent calls to this operation starts from that point. If an insufficient number of resource identifiers have been found during a call to this operation, this operation either times out or returns with COMPLETED set to TRUE. In either case, an empty list of identifiers is returned.

This operation returns a taskId that can be used to get results using the getIntrospectedResourceIdsResult operation or canceled using cancelServerTask, page 50.

This operation returns the total number of introspected resource identifiers in the totalResults element. If the data source has not yet been introspected, this element is set to zero.

This operation returns a completed element that indicates whether or not processing has completed and these are the last results the caller receives. If TRUE, a subsequent call to this operation generates a NotFound fault.

The lastUpdate return element indicates when the data source was last introspected. If the data source has never been introspected, the lastUpdate element is unset.

The resourceIdentifiers element provides the list of native resources paths, types, and subtypes. Paths are relative to the containing data source.

This operation must be run within the same explicit transaction that the original call to getIntrospectedResourceIdsTask was invoked. Otherwise the taskId is reported as NotFound.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

taskId: The task ID associated with the original call.

block (optional): Whether or not to block until processing is complete. Default is FALSE.

page (optional):

- size: The number of resource identifiers to return in the result.
- start: The page number to start retrieving data from. Defaults to 0.

Response Elements

taskId: The task ID associated with the original call.

totalResults (optional): If known, the total result set size. Otherwise this element does not exist.

completed: TRUE if processing is completed and the result set has been exhausted.

lastUpdate (optional): The date and time the resource cache was last created.

resourceIdentifiers (optional): List of native resource paths and types available within a data source.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the taskId or page is malformed.

IllegalState: If the data source is disabled.

NotFound: If the taskId does not exist or has completed.

NotFound: If a different TDV session was used to create the server task.

Security: If the user does not have the ACCESS_TOOLS right.

getIntrospectedResourceIdsTask

Create a server task to fetch all resource identifiers for resources that have been introspected on a given data source.

This operation returns a taskId that can be used to get results using the getIntrospectedResourceIdsResult, page 132, or canceled using cancelServerTask, page 50.

This operation returns the total number of resources that are currently introspected on the given data source. If the data source has never been introspected, totalResults is zero.

If the TDV session that invokes this operation closes before the task is completed, the task is terminated.

This operation must be run within an explicit transaction, so that the taskId can be used in successive calls to getIntrospectedResourceIdsResult.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the data source.

attributes (optional): List of data source type specific attributes. The specific list of supported attributes vary by data source type. (See getDataSourceAttributeDefs, page 115.) Sets the specified attributes but does not alter the value of unspecified attributes. See Attributes Element, page 248.

Response Elements

taskId: The ID of the server task performing the work.

totalResults (optional): If known, the total result set size; otherwise, this element does not exist.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the path is malformed.

IllegalState: If the data source is disabled.

NotFound: If the data source or any portion of the path to the data source does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have READ access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

getIntrospectionAttributeDefs

Get all possible attribute definitions for all possible resource type and subtypes that can be applied to data source resources during introspection. These attributes definitions are used to help define the attributes used to introspect resources, which are normally set within the plan request element when calling introspectResourcesTask, page 169.

The attribute definitions may contain display hints to help assist client tools in displaying appropriate editors for each of the attributes.

Also see getIntrospectionAttributes, page 136, which retrieves the attributes applied to the most recent introspection of a particular resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: Path to the resource.

Response Elements

subTypeAttributeDefs: List of attribute definitions by type and subtype.

- type: The resource type.

- subType: The resource subtype.
- attributeDefs: List of introspection attributes definitions available for the type and subType combination. See Attribute Definitions Element, page 247.

Faults

Security: If the user does not have the ACCESS_TOOLS right.

getIntrospectionAttributes

Get the resource attributes that were applied to a resource the last time it was introspected from a data source. This call only queries the TDV, not the underlying data source. Therefore, the data source does not need to be enabled or currently have a valid connection. If the provided resource has not yet been introspected, no attributes are returned. No checks are performed to see if the resource currently exists within the data source. A fault is generated if the resource does not exist within the TDV.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source.

dsPath: The relative path within the data source of the resource for which to gather attributes. If the path is root (`"/"`) or the empty string, the introspection properties for the data source itself are returned.

dsType: The type of the resource.

Response Elements

attributes (optional): The attributes applied to the resource if it has been introspected. Otherwise, this is unset. See Attributes Element, page 248.

Faults

IllegalArgument: If path, dsPath, or dsType is malformed.

NotFound: If the data source resource or any portion of the path to the data source does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have READ access on the data source.

Security: If the user does not have the ACCESS_TOOLS right.

getLicenses

Get all licenses registered with the server. Also see addLicenses, page 39 and removeLicenses, page 187.

Location

/services/webservices/system/admin/resource/operations

Request Elements

N/A

Response Elements

licenses (optional): List of licenses registered with the server. Unset if no licenses are registered with the server. See Licenses Element, page 254.

Faults

Security: If the user does not have both ACCESS_TOOLS and READ_ALL_CONFIG rights.

getLockedResources

Get all of the locked resources in the server. Resources are normally uniquely identified by both the path and type of the resource. This operation returns all resources at a given path regardless of type.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

includeOnlyUnlockableResources (optional): If TRUE, the response includes only resources the caller has the right to unlock.

Response Elements

resources: List of the locked resources. See Resources Element, page 257.

Faults

IllegalArgument: If an illegal detail or includeOnlyUnlockableResources element is provided.

Security: If the user does not have the ACCESS_TOOLS right.

getLoginModule

Retrieve a login module based on its ID.

Location

/services/webservices/system/util/security/operations/

Request Element

id: ID of login module to retrieve.

Response Elements

id: ID of the login module retrieved.

name

bundleName

group (optional)

enabled (optional)

bundleEnabled (optional)

properties (optional): List of property name-value pairs.

Faults

Invalid ID: If the ID specified does not denote a valid LoginModule instance.

getLoginModuleDefaultProperties

Retrieve the default property list for a login module type.

Location

/services/webservices/system/util/security/operations/

Request Elements

name: The bundle-name of the login module class in bundle\class format.

Response Elements

properties (optional): List of property elements, each of which is a name-value pair.

Fault

Invalid ID: If the input name does not designate a valid login module type.

getLoginModuleList

Retrieve a list of the IDs of all the current login module instances.

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Elements

id (optional): List of ID values of the login module instances.

Faults

N/A

getMostRecentIntrospectionStatus

Get the results from the most recent introspection that was run and committed on the given data source.

Results are in the form of introspection change entries (which contain the path, type, and subtype of the resource that was introspected), the introspection action that occurred, and a message, if available, regarding introspection of that resource.

This operation returns different levels of introspection status based on the value of the detail request element:

- NONE: Minimal information to indicate the final run state of the introspection task.
- SIMPLE: Overall status plus counts.
- FULL: List of introspection change entries, containing all entries and messages that occurred during the last completed and committed introspection.

If the data source has never successfully completed an introspection, the status response element is unset.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source.

detail: The level of detail to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

status (optional): The introspection status report. See Introspection Report Status Element, page 253.

Faults

DataSourceError: If a data source connection cannot be established, or if a data source request returns an error.

IllegalArgument: If the path or detail element is malformed.

NotFound: If a data source resource cannot be found at the given path.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

getParentDataSourceType

Get the parent data source type of the given data source type name.

Location

/services/webservices/system/admin/resource/operations/

Request Element

selfDataSourceTypeName: The name of current data source type.

Response Elements

resources: List of resources and their characteristics. see Resources Element, page 257.

parentDataSourceType: The parent's data source information. Can be NULL.

- name
- type
- attributes (optional): See Attributes Element, page 248.

Faults

NotFound: If current data source type is not found.

getParentResource

Get the parent resource of the given resource. The root resource ("/") has no parent.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to get parent of.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the parent resources and their characteristics. The list is empty for the root resource (`"/"`). See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

getPrincipalMapping

Retrieve a principal mapping instance.

Location

/services/webservices/system/util/security/operations/

Request Element

id: The ID of the principal mapping.

Response Elements

id: The ID of the principal mapping.

type: The type of the principal mapping.

identifier: The identifier of the principal mapping.

group: The group to which the principal mapping belongs.

Faults

Invalid Input: If the input ID does not designate a valid principal mapping.

getPrincipalMappingList

Retrieve the list of IDs for all principal mappings.

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Element

id (optional): List of ID values for all principal mappings.

Faults

N/A

getProceduralResult

Get the procedural result associated with the resultId returned from a call to executeProcedure, page 89 or executeSqlScript, page 98.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "outputs" element is set.

If set to FALSE, then this operation will return when the processing associated with the execution has completed, but it will not wait to fill the "outputs" element with data.

If includeMetadata is TRUE, the response includes the metadata element. The metadata element describes the names and types of the output parameter data that are provided in the result, either in this call or in a later call to getProceduralResult, page 143.

The completed element reports whether all possible results have been retrieved.

The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request has been created but is not yet running.
- WAITING: The request is waiting in a queue for the server to process the request.

- RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

The rowsAffected element is set if it is known how many rows were affected by this execution.

The outputs element contains a set of ProcValue instance. If the out type is cursor, you should get a resultId which is a handle to the result in server. You can use it with getTabularResult, page 161 and closeResult, page 56. If the out type is not cursor, you should get real output value.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

resultId: The result ID.

isBlocking: If TRUE (default), do not return until the execution completes.

includeMetadata (optional): If TRUE, the response contains information about the output parameter names and their types. Defaults to FALSE.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request performing the execution.

metadata (optional): Table metadata listing the parameter names and types within the result.

- parameter: See Parameters Element, page 255.

rowsAffected (optional): If known, the number of rows affected by the execution; otherwise unset.

outputs (optional): List of type-value pairs of procedural outputs.

Faults

IllegalArgument: If the isBlocking element is malformed.

NotFound: If the resultId does not exist within the current transaction or has already been closed.

RuntimeError: If an error occurs during execution. The resultId is closed if this occurs.

getResource

Get the specified resource.

Multiple resources are returned if a path has multiple resources that differ by type—for example, /shared/examples/ds_inventory (data source) and /shared/examples/ds_inventory (view).

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to get.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of resources and their characteristics. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

getResourceCacheConfig

Get the cache configuration for a resource. TABLE and PROCEDURE resources support caching.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a cacheable resource.

type: The type of the resource. A valid input is TABLE or PROCEDURE.

Response Elements

The getResourceCacheConfig response has five levels of elements. Elements at the fourth and fifth levels are preceded by their level number in square brackets ([4] and [5]).

cacheConfig: The cache configuration of the given resource.

- allOrNothing (optional): The refresh policy of the cache group: TRUE if cache group refresh policy should be all or nothing; FALSE if cache group refresh policy should be best effort. This flag applies only to cache group. For individual cache groups, this flag always returns TRUE and setting this flag has no effect.
- configured (optional): TRUE if caching should be configured for the given resource; otherwise FALSE. If configured is FALSE, all other elements are ignored.
- enabled (optional): TRUE if the cache is enabled; otherwise FALSE.
- incremental (optional): TRUE if the cache is incrementally maintained; otherwise FALSE.

- storage (optional): How the cached is stored.
  - useDefaultCacheStorage (optional): The server returns TRUE for this value if default cache data source is used to store the cache tables; otherwise, FALSE.
  - mode (optional): The type of storage to use for the cache. May be AUTOMATIC, DATA_SOURCE or DATA_SOURCE_OTPS.
  - bucketMode (optional): Present when storage mode is DATA_SOURCE_OTPS; otherwise ignored. May be AUTO_GEN or MANUAL.
  - bucketProperties (optional): Present when bucketMode is AUTO_GEN; otherwise ignored.
  - [4] bucketCatalog (optional): Database catalog in which to create the bucket.
  - [4] bucketSchema (optional): Database schema in which to create the bucket.
  - [4] bucketPrefix (optional): Short string which begins the name of each bucket.
  - [4] numBuckets (optional): Number of buckets to use for caching.
  - dropCreateIdx (optional): If TRUE, TDV automatically drops indexes before loading cache data, and creates them after loading.
  - storageDataSourcePath (optional): If the mode is DATA_SOURCE or DATA_SOURCE_OTPS, this identifies the path to the data source being used to store cache data.
  - storageTargets: (optional) Used for storing cache data. If the mode is DATA_SOURCE or DATA_SOURCE_OTPS, this identifies the tables. If the latter, it is required when bucketMode is MANUAL; otherwise, storageTargets is ignored.
  - [4] entry (optional):
  - [5] targetName: For a TABLE resource and storage mode DATA_SOURCE, this is always the result. For a TABLE resource and storage mode DATA_SOURCE_OTPS, this may be result, result1, result2, and so on. For a PROCEDURE resource, this is the name of a cursor parameter, or an empty string for the scalar output parameters.
  - [5] path: The path to the table used for storing this data.
  - [5] type: Always TABLE.

- refresh (optional): How the cache should be refreshed. See Refresh Element, page 256.
  - mode: How the cache should be refreshed. May be MANUAL or SCHEDULED.
  - schedule (optional): Present if the mode is SCHEDULED. See Schedule Element, page 258.

  Note: The value of the mode element under schedule is always INTERVAL for cache.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalState: If the resource type does not support caching.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getResourcePlan

Execute the provided path directly within the TDV server and return the execution plan. Only PROCEDURE and TABLE resources can be executed.

The rootNodeTitle is the name of this sqlText execution plan. It is possible to get the execution plan for the resource in the path element.

The parameters element contains a set of parameters to be inputs to the execution for TDV script. A parameter contains two subelements:

- definition: SQL language type for this parameter value. For example, VARCHAR(40) or BIGINT.
- value: The value of this parameter.

The queryPlanRoot contains the root plan.

The name contains this plan's node name.

The type shows this plan's node type. PROCEDURE and TABLE are valid types.

The properties element contains a set of plan node properties.

Only users with the ACCESS_TOOLS right can call this operation.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

rootNodeTitle (optional): The name of this path execution plan. Defaults to Execution Plan.

path: Specify a full resource path of a table or procedure for that SQL to be evaluated.

type: TABLE and PROCEDURE types are valid.

parameters (optional): A list of parameter elements (containing definition-value pairs) to use as input for the execution.

Response Elements

queryPlanRoot (iterative over child nodes, to any depth):

- name: Query plan node name.
- type: Type of this query plan node.
- properties (optional): List of name-value pairs of properties for current node.

Faults

RuntimeError: If an error occurs during execution.

Security: If the user does not have the ACCESS_TOOLS right and other appropriate priveleges.

getResourcePrivileges

Get the privilege information for any number of resources.

The returned privileges per user or group are the privileges specifically given to that user or group. In each privilegeEntry, the combinedPrivs element contains the effective privileges for that user or group based on their membership in all other groups. In each privilegeEntry, the inheritedPrivs element only contains the privileges that were inherited due to group membership. Logically OR'ing the privs and inheritedPrivs is the same as the combinedPrivs.

A user with GRANT privilege or with READ_ALL_RESOURCES right receives all privilege information for all users for a that resource. Other users receive only their own privilege information.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of path-type pairs to get privilege information for.

filter (optional): A filter string. The only legal values in this release are an empty string and ALL_EXPLICIT.

includeColumnPrivileges (optional)

Response Elements

privilegeEntries: List with the privilege information for each of the requested resources.

- privilegeEntry (optional): Path-type pairs and list for one or more privileges.
  - combinedPrivs (optional)
  - inheritedPrivs (optional)

Faults

IllegalArgument: If any path is malformed or type is illegal.

NotFound: If any one of the provided resources does not exist.

Security: If the user does not have READ access on all items in each path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

getResources

Get the specified resources. Only the resources that exist and which the user has proper privileges to access are returned. It is possible to get back a shorter list than requested.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of path-type pairs of the resources to get.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the resources and their characteristics. see Resources Element, page 257.

Faults

IllegalArgument: If any of the provided paths are malformed, or types or detail are illegal.

Security: If the user does not have the ACCESS_TOOLS right.

getResourceStatisticsConfig

Get the statistics configuration for a resource.

Also see getDataSourceStatisticsConfig, page 118.

Multiple resources are returned if a path has multiple resources that differ by type—for example, */shared/examples/ds_inventory* (data source) and */shared/examples/ds_inventory* (view).

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource.

type: The type of the resource. May only be a relational physical TABLE.

Response Elements

statisticsConfig: The statistics gathering configuration of the given resource.

- cardinalityMin (optional): Minimum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityMax (optional): Maximum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityExpected (optional): Expected cardinality for this resource. Setting this value overrides gathered statistics.

- gatherEnabled (optional): Defines what statistics should be gathered for this resource. Valid values are DEFAULT, CUSTOM, DISABLED or TABLE_BOUNDARY.
- maxTime (optional): Number of minutes after which each thread performing statistics gathering gives up: 0 means no timeout; -1 means use data source setting.
- columns (optional): Only applicable if gatherEnabled is set to CUSTOM.
- column: List indicating what specific data to get for each column.
  - name: Simple column name.
  - flags: Valid values are NONE, BOUNDARY or ALL.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the resource if of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getResourceStatsSummary

Get the statistical summary for the resource.

Multiple resources are returned if a path has multiple resources that differ by type—for example, /shared/examples/ds_inventory (data source) and /shared/examples/ds_inventory (view).

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource.

type: The resource type. Valid values are DATA_SOURCE or TABLE.

Response Elements

status: The statistical status of the resource.

msg: Detailed information about the status.

last_refresh_end: Time at which the last refresh operation ended.

curr_refresh_start: Time at which the current refresh operation started.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

getResourceUpdates

Determine all of the updates occurred for a given set of resources. Updates include the addition of new sibling resources, changes to resources, movement of resources, and deletion.

By default, the given list of resources is compared with the most recent versions of themselves. If the changeId element is set, the resources are compared with the given changeId instead. If the provided changeId is older than the current version of a given resource, no changes are reported for that resource.

You can specify one or neither of the changeId or version elements. You cannot specify both.

If an id element for a nonexistent resource or one that is not accessible is provided, that ID appears in the deleted list.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

detail: The level of detailin the added, changed, and moved lists. Valid values are NONE, SIMPLE, and FULL.

discoverChildren: TRUE if all descendents of given resources should be returned. FALSE if you only want the given resources to be checked. With FALSE, new sibling resources are discovered, but not descendents.

includeLockState: TRUE if changes to lock state should be considered as a resource update.

changeId (optional): If set, only resources with a change ID greater than the given change ID are returned.

version (optional; not supported): The element is not currently supported.

resourcesToUpdate: List of resources to check for updates.

entry: An entry for a resource to check.

- path: The path of the resource.
- type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE.
- id (optional): The ID of the resource. This is required to help track resource moves. Otherwise moved resources appears as deleted and added.
- changeId (optional): The Change ID of the resource. This is required to identify precise changes to a resource.
- hasChildren (optional): If TRUE, the client believes this resource has children. This is used to help detect deletion of all child resources.
- isExpanded (optional): Whether or not children of this resource are already known by the client. For use by client tree views. If TRUE, the immediate children of an expanded container are discovered regardless of the discoverChildren setting.
- lastClientRefresh (optional): The last time this resource was refreshed by the client. Time is relative to the server. May be used an alternative to using Change ID to detect changes. This is not as precise as using Change ID.
- lockCreateTime (optional): The known time a lock was created for this resource. Time is relative to the server. If includeLockState is TRUE, this value is used to help detect multiple lock state changes where the current state is similar to but not exactly the same as the previously known state in the client.

Response Elements

resourceUpdates (optional):

- addedResources (optional): List of resources that were created as siblings of the provided resources.
- changedResources (optional): List of resources that have been changed.
- movedResources (optional): List of resources that have been changed including their path.
- deletedResources (optional): List of resource IDs of the resources that have been deleted.

Faults

IllegalArgument: If any of the detail, discoverChildren, includeLockState, changeId, version, or resources elements are malformed.

NotAllowed: If both the changeId and version elements are set.

NotSupported: If the version element is set. This is reserved for future use.

getResultSetPlan

Return the execution plan by resultId.

The rootNodeTitle is the name of this sqlText execution plan.

The resultId element can be obtained from the operation response returned from executeSql, page 92, executeNativeSql, page 86, executeProcedure, page 89, or executeSqlScript, page 98. The queryPlanRoot contains the root plan.

Only users with the ACCESS_TOOLS right can call this operation.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

rootNodeTitle (optional): The name of this resultId execution plan. Defaults to Execution Plan.

resultId: The SQL or Script executed resultId.

Response Elements

queryPlanRoot:

- name: Query plan node name.
- type: Type of this query plan node.
- properties (optional): List of name-value pairs of properties for current node.
- children (optional): List of child nodes belonging to current node. This list is iterative to any depth.

properties (optional): Same as properties element, above.

children (optional), Same as children element, above; and so on.

Faults

RuntimeError: If an error occurs during execution.

Security: If the user does not have the ACCESS_TOOLS right and other appropriate privileges.

getServerActions

Get a list of all available server actions and their definitions. These are also listed in Server Actions Reference, page 347.

Server actions can be invoked using performServerAction, page 178.

Location

/services/webservices/system/admin/server/operations

Request Elements

N/A

Response Elements

actions: List of available server actions.

- action: A server action.
- name: The name of the server action.
- attributeDefs: List of attribute definitions of the server action which conveys how it should be invoked.

attributeDef (optional): See Attribute Definitions Element, page 247.

- annotation: A description of the server action.

Faults

Security: If caller does not have ACCESS_TOOLS right.

getServerAttributeDefChildren

Get the child server attribute definitions of the given path. This is an empty list for attribute definitions that are not of the FOLDER type.

Location

/services/webservices/system/admin/server/operations

Request Elements

path: A path to a server attribute.

Response Elements

attributeDefs: List of child attribute definitions. The names of the returned attribute definitions are fully qualified paths in the server attribute namespace.

- attributeDef (optional): See Attribute Definitions Element, page 247.

Faults

IllegalArgument: If any of the provided paths are malformed.

NotFound: If the path represent a nonexistent server attribute.

NotFound: If any of the paths represent a valid non-public server attribute and the user does not have READ_ALL_CONFIG right.

Security: If the user does not have ACCESS_TOOLS right.

getServerAttributeDefs

Get server attribute definitions for the given paths.

Location

/services/webservices/system/admin/server/operations

Request Elements

paths: Optional list of paths for which to get the attribute definitions.

Response Elements

attributeDefs: List of attribute definitions. The names of the returned attribute definitions are fully qualified paths in the server attribute namespace. See Attribute Definitions Element, page 247.

- attributeDef (optional): See Attribute Definitions Element, page 247.

Faults

IllegalArgument: If any of the provided paths are malformed.

NotFound: If any of the paths represent a nonexistent server attribute.

NotFound: If any of the paths represent a valid non-public server attribute and the user does not have READ_ALL_CONFIG right.

Security: If the user does not have ACCESS_TOOLS right.

getServerAttributes

Get server attributes for the given paths.

Location

/services/webservices/system/admin/server/operations

Request Elements

paths: Optional list of paths for which to get the attributes.

Response Elements

attributes: List of attributes. The names of the returned attributes are fully qualified paths in the server attribute namespace. See Attributes Element, page 248.

Faults

IllegalArgument: If any of the provided paths are malformed.

NotFound: If any of the paths represent a nonexistent server attribute.

NotFound: If any of the paths represent a valid non-public server attribute and the user does not have READ_ALL_CONFIG right.

Security: If the user does not have ACCESS_TOOLS right.

getServerInfo

Get a list of server attributes. These include but are not limited to:

- /Server/Information/Protocol Version: The protocol version of the server.
- /Server/Information/Host IP Address: The IP address of the server.
- /Server/Information/Host Name: The host name of the server.
- /Server/Data Services/JDBC-ODBC/Client Port: The server's JDBC port number.
- /Server/Configuration/Version: The version of the server.
- /Server/Configuration/Services Port: The server's Web Services port number.
- /Server/Information/Supported Protocol Versions: List of protocol version the server supports.

Note: This operation can be invoked anonymously.

Location

/services/webservices/system/util/session/operations/

Request Elements

N/A

Response Elements

attributes: List of server attributes. See Attributes Element, page 248.

Faults

N/A

getServerName

Get the server display name.

Location

/services/webservices/system/admin/server/operations

Request Elements

N/A

Response Elements

serverName: The server display name.

Faults

Security: If the user does not have ACCESS_TOOLS and READ_ALL_CONFIG rights.

getSqlPlan

Execute the provided sqlText directly within the TDV server and return the execution plan. The rootNodeTitle is the name of this sqlText execution plan.

You can get the execution plan for the SQL statement or SQL Script in sqlText. Common SELECT, UPDATE, INSERT, DELETE SQL without input parameters is supported. Input parameters for SQL script are also supported.

The parameters element contains a set of parameters to be inputs to the TDV script. A parameter contains two elements:

- definition: SQL language type for this parameter value. For example, VARCHAR(40) or BIGINT
- value: The value of this parameter.

Only users with the ACCESS_TOOLS right can call this operation.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

rootNodeTitle (optional): The name of this sqlText execution plan. Defaults to Execution Plan.

sqlText: The SQL to be executed.

parameters (optional): Name-value pairs for the arameters to use as execution input.

Response Elements

queryPlanRoot: The root plan, which contains:

- name: Query plan node name.
- type: Type of this query plan node.
- properties (optional): List of properties for the current node and its child nodes (to any depth).

Faults

RuntimeError: If an error occurs during execution.

Security: If the user does not have the ACCESS_TOOLS right and other appropriate priveleges.

getTabularResult

Get the tabular result associated with the resultId returned from a call to executeNativeSql, page 86 or executeSql, page 92, or contained within the outputs element of a call to executeProcedure, page 89 or executeSqlScript, page 98.

If "isBlocking" is set to TRUE, then this operation will not return until the processing associated with the execution has completed and the "outputs" element is set.

If set to FALSE, then this operation will return when the processing associated with the execution has completed, but it will not wait to fill the "outputs" element with data.

If includeMetadata is TRUE, the response includes the metadata element. The metadata element describes the names and types of the column data that are provided in the result, either in this call or in a later call to getTabularResult, page 161.

The completed element reports whether all possible results have been retrieved.

The requestStatus element reports the status of the server request associated with the execution. The request status can be one of the following:

- STARTED: The request has started. The request was created, but is not yet running.
- WAITING: The request is waiting in a queue for the server to process the request. RUNNING: The request is currently being executed by the server.
- COMPLETED: The execution associated with the request has completed. Results can now be acquired.
- CLOSING: The request is closing.
- SUCCESS: The request closed with success.
- FAILURE: The request closed with failure.
- TERMINATED: The request was terminated.

If skipRows is set, that number of rows is skipped in the execution output before returning any results. If skipRows is greater than the total possible number of rows, no rows are returned.

If maxRows is set, result contains at most maxRows number of rows. If maxRows is smaller than the total number of rows of data available, additional calls to getTabularResult, page 161 need to be made to get the rest of the available data. Use hasMoreRows element in result to determine if additional data is available. This is more accurate than comparing the number of rows returned with maxRows because the server might opt to return fewer than maxRows.

If consumeRemainingRows is set and TRUE, all remaining rows in excess of maxRows are consumed.

The rowsAffected element is set if it is known how many rows were affected by this execution. This includes a count of rows that were skipped (see skipRows element) or consumed (see consumedRemainingRows element).

Location

/services/webservices/system/admin/execute/operations/

Request Elements

resultId: The result ID.

isBlocking (optional): If TRUE (the default), do not return until the execution completes.

includeMetadata (optional): If TRUE, the response contains information about the column names and their types. Defaults to FALSE.

skipRows (optional): The number of rows to skip in the execution output before generating results. If not set, no rows are skipped.

maxRows (optional): The maximum number of rows to return. If not set, all rows are returned.

consumeRemainingRows (optional): If set and TRUE, all remaining rows after maxRows are consumed.

Response Elements

completed: If TRUE, all processing associated with execution has completed.

requestStatus: Status of the server request performing the execution.

metadata (optional): Table metadata listing the column names and types within the result:

column: See Column Element, page 249.

rowsAffected (optional): If known, the number of rows affected by the execution; otherwise unset.

result (optional): The result data.

hasMoreRows: TRUE if the table has more rows than the number affected (rowsAffected, above).

totalRowCount: Total number of rows in the table.

Faults

IllegalArgument: If the isBlocking, includeMetadata, or maxRows elements are malformed.

NotFound: If the resultId does not exist within the current transaction or has already been closed.

RuntimeError: If an error occurs during execution. The resultId is closed if this occurs.

getTransformFunctions

Get all functions related to transformations.

Location

/services/webservices/system/admin/resource/operations

Request Element

language: Target language. Valid values are JAVA, SQL, SQL_SCRIPT, XQUERY, XSLT, and CUSTOM. (CUSTOM is not an actual language; it is used for custom functions.)

Response Element

functions: List of transform-related functions.

Fault

IllegalArgument: If the language is malformed.

getUsedDataSources

Get all of the dependency data source resources the given resources depends on.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: A list of resource-path/type pairs. Type can be TABLE, PROCEDURE, LINK, DEFINITION_SET, DATA_SOURCE, or TRIGGER

detail: The level of detail of resources in the response. May be NONE, SIMPLE, or FULL.

Response Elements

resources: List of the resources used by the request-resources, and their characteristics. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in the path.

Security: If the user does not have the ACCESS_TOOLS right.

getUsedResources

Get all of the resources the given resource depends on.

Also see getDependentResources, page 120, which gets all the resources that use a resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to analyze for dependencies.

type: The type of the resource specified by the given path. Valid values are: DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the resources used by the given resource and their characteristics. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or an illegal type or detail is provided.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

getUser

Get the user definition of the caller.

Location

/services/webservices/system/admin/user/operations/

Request Elements

N/A

Response Elements

user: Name, domain and ID of user (see Users Element, page 260), as well as:

- explicitRights. For a table of values, see User and Group Rights Mask, page 259.
- interitedRights. For a table of values, see User and Group Rights Mask, page 259.
- annotation (optional)
- groupNames (optional): Names and domains of the groups to which the user belongs.

Faults

NotFound: If the domain does not exist.

NotFound: If the user does not exist.

Security: If the user does not have the ACCESS_TOOLS right.

getUsers

Get the definitions of the given domain users.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

names: List of user names within the domain.

Response Elements

users: List of users within the domain. See User Element, page 259.

Faults

NotFound: If the domain does not exist.

NotFound: If any of the requested users do not exist.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If the user does not have the READ_ALL_USERS and the user names being retrieved includes any user other than the current user.

getUsersByGroup

Get the definitions of all users within the given group, who are also within the given domain.

Location

/services/webservices/system/admin/user/operations/

Request Elements

groupName: The group name.

domainName: The domain name.

Response Elements

users: List of users within the domain. See User Element, page 259.

Faults

NotFound: If the group does not exist.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS right.

introspectResourcesResult

Get the results from introspectResourcesTask, page 169. The number of results returned is limited by page size and total number of known fields.

Results are in the form of introspection change entries, which contain the path, type, and subtype of the resource that was introspected, the introspection action that occurred, and a message, if available, regarding introspection of that resource.

Subsequent calls to this operation incrementally return the full set of results.

If the block element is set and TRUE, this operation blocks until the task is complete. Otherwise, this operation does not block.

The page size controls the maximum number of change entries that are returned from this call.

The page start determines which result is returned first. This can be used to jump ahead in the result list. This jump is relative to the current position. Providing a non-zero page start with every call to this operation has the effect of skipping a page-size number of results each time.

Specifying the page start sets the position where you would like to start receiving results. Subsequent calls to this operation start from that point. If an insufficient number of results have been found during a call to this operation, this operation either times out or returns with COMPLETED set to TRUE. In either case, an empty list of results is returned.

This operation returns a taskId that can be used to get results using , or canceled using cancelServerTask, page 50.

This operation returns the total number of results in the totalResults element. If it is not known what the total number of results is, this element is unset.

This operation returns a completed element that indicates whether or not processing has completed and these are the last results the caller receives. If TRUE, a subsequent call to getIntrospectableResourceFieldsResult generates a NotFound fault.

This operation returns the introspection status in the status element based on the value specified in the detail element:

- NONE: Minimal information is returned to indicate the running state of the task.
- SIMPLE: Overall status and counts are returned.

- FULL: List of introspection change entries is returned.

The status startTime element is set only after introspection has started. The status endTime element is set only when introspected has completed.

The list of introspection change entries contains only entries for newly added messages or resource identifiers since the last call to introspectResourcesResult, page 167.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

taskId: The server task ID associated with the original call.

block (optional): Whether or not to block until processing is complete. Default is FALSE.

page (optional):

- size: The number of resource identifiers to return in the result.
- start: The page number to start retrieving data from. Defaults to 0.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

taskId: The server task ID associated with the original call.

totalResults (optional): If known, the total result set size. Otherwise this element does not exist.

completed: TRUE if processing is completed and the result set has been exhausted.

status: The introspection status report. See Introspection Report Status Element, page 253.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the taskId, page, or detailLevel is malformed.

IllegalState: If the data source is disabled.

NotFound: If the taskId does not exist or has completed.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If a different TDV session was used to create the server task.

introspectResourcesTask

Create a server task to introspect a data source.

Introspection is the process of analyzing native resources in a data source and creating resources within the TDV that represent them. The introspection plan provides details on what exactly should be introspected. This includes resources to be added, updated, or removed; whether to update all previously introspected resources; introspection attributes for individual resources; and other options.

ntrospection attributes for individual resources need only be provided if any of the introspection attributes for a particular resource have changed. Otherwise, the previously used attributes (if available) or defaults can be used.

The introspection plan contains a list of introspectionPlanEntries. For each entry the following needs to be specified:

- resourceId: The path, type, and subtype of the resource to be introspected. Resource paths are relative to the data source. An empty path, for example, identifies the data source itself.
- action:
  - ADD_OR_UPDATE: The resource is added if it does not already exist. Otherwise it is updated.
  - REMOVE: The resource is removed if it exists.

    Data sources cannot be removed. Use destroyResource, page 84 to remove a data source.
  - ADD_OR_UPDATE_RECURSIVELY: Not supported.
- attributes: List of introspection-specific attributes to apply during introspection. If a specific attribute is not specified, the attribute that was used during the previous introspection is used. If the resource has not previously been introspected, the default value for that attribute is used.

If the plan's failFast option is TRUE, the introspection fails when the first error occurs. Otherwise the plan runs to completion as a best effort. The default is FALSE.

If the plan's commOnFailure option is TRUE, the introspection commits whatever it can. If fastFail is also TRUE, only resources successfully introspected up to that point are committed. The default is FALSE.

If the plan's autoRollback option is TRUE, the introspection task rolls back rather than being committed. This supersedes all commit options. This allows you to perform a dry run of resource introspection. If autoRollback is TRUE, you can use introspectResourcesResult, page 167. If autoRollback is FALSE or unset, the introspection is not automatically rolled back.

If the plan's scanForNewResourcesToAutoAdd option is TRUE, the introspection task scans for native resources that have been newly added to the data source. If a newly added resource is found and its parent container has the autoAddChildren introspection attribute set, that child is automatically introspected.

This operation returns a taskId that can be used to get results using introspectResourcesResult, or canceled using cancelServerTask, page 50.

This operation returns the number of resources that are added, removed, or updated in the totalResults element. The totalResults element does not exist until the full total is known. If this number is not known or only partial results are available, this element is empty.

If the TDV session that invokes this operation closes before the task is completed, the task continues to run until completion. However, the taskId can only be used with cancelServerTask. It cannot be used with introspectDataSourceResult.

The introspection task is run within an independent background transaction on the server. It is not necessary to call introspectResourcesTask, page 169 within an explicit transaction in order to call introspectResourceResult or cancelServerTask. This background transaction survives across server restarts until it is completed.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source.

plan: The introspection plan, which details the resource to act upon. See Introspection Plan Element, page 252.

runInBackgroundTransaction

attributes (optional; same as attribute list and subelements as under plan above). Also see Attributes Element, page 248.

Response Elements

taskId: The ID of the server task performing the work.

totalResults (optional): If known, the total result set size. Otherwise this element does not exist.

completed: Boolean. Whether or not task ran to completion.

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the path, plan, or attributes are malformed.

IllegalState: If the data source is disabled.

NotFound: If a data source resource cannot be found at the given path.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

joinCluster

Join a cluster.

Location

/services/webservices/system/admin/resource/operations

Request Elements

remoteHostName: host name of the remote server to send join request.

remotePort (optional): port of the server to send join request.

remoteDomainName: name of the user domain on the remote server.

remoteUserName: name of the user on the remote server.

remotePassword: password of the user on the remote server.

Response Elements

messages (optional): List of messages regarding the action. See Messages Element, page 255.

Faults

IllegalState: If the server is already part of a cluster.

RemoteServerError: If connection fails with the remote server.

Security: If the user does not have ACCESS_TOOLS, MODIFY_ALL_CONFIG, MODIFY_ALL_USERS, and MODIFY_ALL_RESOURCES rights.

Security: If the authentication fails on the remote server.

Security: If the user does not have ACCESS_TOOLS and MODIFY_ALL_CONFIG rights on the remote server.

lockResource

Lock the specified resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to lock.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, or TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the locked resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or the type or detail is illegal.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path, except the last one.

Security: If the user does not have WRITE access to the last item in path and does not have the MODIFY_ALL_RESOURCES right.

Security: If the user does not have the ACCESS_TOOLS right.

lockResources

Lock all specified resources.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of path and type pairs of the resources to lock:

- entry (optional):
- path
- type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, or TRIGGER.

detail: The level of detail of resources in the response. May be NONE, SIMPLE, and FULL.

Response Elements

resources: List of the locked resources. See Resources Element, page 257.

Faults

IllegalArgument: If any of the provided paths are malformed, or types or detail are illegal.

NotFound: If any of the resources or any portions of the paths do not exist.

Security: If the user does not have READ access on all items in path, except the last one.

Security: If the user does not have WRITE access to the last item in path and does not have the MODIFY_ALL_RESOURCES right.

Security: If the user does not have the ACCESS_TOOLS right.

moveResource

Move the specified resource into a folder using a new name.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to be moved.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, or TRIGGER.

targetContainerPath: The path of the target container to move the resource into.

newName: The new name to call the moved resource.

overwrite: If a resource exists in the target container with the same name and type of the target resource and overwrite is TRUE, the resource within the target container is overwritten. If overwrite is FALSE, a DuplicateName fault is generated and resource are not moved.

Response Elements

N/A

Faults

DuplicateName: If a resource in the target container exists with the same name and type as one of the source and overwrite is FALSE.

IllegalArgument: If any of the given paths or types are malformed.

IllegalState: If the source resource cannot be moved. Resources in */services/databases/system*, */services/webservices/system*, or within any physical data source cannot be moved.

NotAllowed: If the source resource is not allowed to exist within the target container. Resources cannot be moved into a physical data source. LINK resources can only be moved into RELATIONAL_DATA_SOURCEs, SCHEMAs, and PORTs under */services*. Non-LINK resources cannot be moved into any location under */services*.

NotFound: If the source resource or any portion of the path to the target container do not exist.

Security: If the user does not have READ access on all items in the source path.

Security: If the user does not have READ access on the items in the targetContainerPath other than the last item.

Security: If the user does not have WRITE access to the last item in targetContainerPath.

Security: If the user does not have WRITE access to a resource that is to be overwritten.

Security: If the user does not have the ACCESS_TOOLS right.

moveResources

Move the specified resources into a folder.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of source path-type pairs to move.

- entry (optional):
  - path
  - type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

targetContainerPath: The path of the target container to move the resources into.

overwrite: If a resource exists in the target container with the same name and type of the target resource and overwrite is TRUE, the resource within the target container is overwritten. If overwrite is FALSE, a DuplicateName fault is generated and resource are not moved.

Response Elements

N/A

Faults

DuplicateName: If a resource in the target container exists with the same name and type as one of the source and the overwrite is FALSE.

IllegalArgument: If any of the given paths are malformed or if any of the types are illegal.

IllegalState: If any of the source resources is not allowed to be moved. Resources in */services/databases/system*, */services/webservices/system*, or within any physical data source cannot be moved.

NotAllowed: If any of the source resources is not allowed to exist within the target container. Resources cannot be moved into a physical data source. LINK resources can only be moved into RELATIONAL_DATA_SOURCEs, SCHEMAs, and PORTs under */services*. Non-LINK resources cannot be moved into any location under */services*.

NotFound: If any of the source resources or any portion of the path to the target container do not exist.

Security: If the user does not have READ access on all items in the source paths.

Security: If the user does not have READ access on the items in the targetContainerPath other than the last item.

Security: If the user does not have WRITE access to the last item in targetContainerPath.

Security: If the user does not have WRITE access to a resource that is to be overwritten.

Security: If the user does not have the ACCESS_TOOLS right.

parseSqlQuery

Parse the sqlText to check if the syntax is correct for TDV server.

Location

/services/webservices/system/admin/execute/operations/

Request Elements

sqlText: The SQL to be parsed.

Response Elements

No output element if parse sqlText SUCCESS, the input SQL is correct for TDV.

Faults

RuntimeError: If an error occurs during parse.

RuntimeError: If the user does not have appropriate privileges on any resources referred to by the scriptText.

Security: If the user does not have the ACCESS_TOOLS right.

performArchiveImport

Start importing an archive.

This operation can only be used with an import archive. It can be used anytime after createImportArchive, page 73 is called, as long as it is within the same transaction and the archive has not been closed.

If an import status is CANCELED or FAIL, the import is in an inconsistent state, and so the bounding transaction should be rolled back before performing other operations.

An import fails only if the caller does not have permission to import most data. The only exceptions to this are the import options marked OWNER ONLY. Attempts to set owner-only options on resources that the caller does not own nor has admin privileges for generate a warning message in the report.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the archive.

isBlocking: If TRUE, this call does not return until the import has completed. If FALSE, this call returns immediately.

Response Elements

status:

- CANCELED if the archive was canceled during this call.
- SUCCESS if the import has completed.
- INCOMPLETE if the import is still in progress.

archiveReport (optional): One or more entry elements describing messages generated during this call, if any. If no messages exist, this element is unset. See Messages Element, page 255.

Faults

IllegalArgument: If the isBlocking element is malformed.

IllegalState: If this operation is called using an export archive ID.

NotFound: If archive for the archive ID does not exist.

performServerAction

Perform a server action.

Use getServerActions, page 156 to get a complete list of available actions and their attribute definitions. These are also listed in Server Actions Reference, page 347.

Location

/services/webservices/system/admin/server/operations

Request Elements

actionName: The name of the action to invoke.

attributes: List of attributes to apply to the action. See Attributes Element, page 248.

Response Elements

status: If SUCCESS, the action succeeded; otherwise FAIL.

messages: List of entry elements describing messages regarding the action. Messages might be returned regardless of success or failure, depending on the action that was invoked. See Messages Element, page 255.

Faults

IllegalArgument: If any of the attributes are malformed.

IllegalArgument: If an unsupported attribute is provided.

NotAllowed: If a required attribute is missing.

NotFound: If no action for actionName exists.

Security: If caller does not have ACCESS_TOOLS right.

Security: If caller does not have any other rights as required by the specific action.

rbsAssignFilterPolicy

Assign a filter policy for row-based security.

Location

/services/webservices/system/util/security/operations/

Request Elements

name: The name of the filter policy, in one of two forms:

- name: If a name element is specified, the filter policy is retrieved based on its short name.
- procedureName: If a procedureName element is specified, the filter policy is retrieved based on the path of the implementing procedure.

target: The path of the view or table.

operation: (optional):

- ASSIGN: Assigns a policy definition. Default value.
- REMOVE: Removes a filter policy definition.

Response Elements

N/A

Faults

NotFound: If the specified filter policy does not exist

NotFound: If the specified view or table does not exist

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

rbsDeleteFilterPolicy

Remove a filter policy definition. Either a name or a procedure name must be specified.

Location

/services/webservices/system/util/security/operations/

Request Element

name: The name of the filter policy, in one of two forms:

- name: If a name element is specified, the filter policy to delete is identified by short name.
- procedureName: If a procedureName element is specified, the filter policy to delete is identified by the path of the implementing procedure.

Response Elements

N/A

Faults

NotFound: If the specified filter policy does not exist

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

rbsGetFilterPolicy

Retrieve the definition of a filter policy.

Location

/services/webservices/system/util/security/operations/

Request Elements

name: The filter policy's name.

Response Elements

policy: A filter policy definition. See Filter Policy Definition, page 250.

Faults

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

NotFound: If no such filter policy exists.

rbsGetFilterPolicyList

Retrieve a list of filter policies.

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Elements

filterPolicyList:

- filterPolicy (optional): Can include one or more filter policies. See Filter Policy Definition, page 250.

Faults

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

rbsIsEnabled

Checks whether row-based security is enabled or not.

Location

/services/webservices/system/util/security/operations/

Request Elements

N/A

Response Elements

enabled: RBS is enabled (TRUE) or not (FALSE).

Faults

N/A

rbsSetEnabled

Enable or disable row-based security.

Location

/services/webservices/system/util/security/operations/

Request Elements

enabled: Sets whether RBS is to be enabled (TRUE) or disabled (FALSE).

Response Elements

N/A

Faults

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

rbsWriteFilterPolicy

Create or update a filter policy.

Location

/services/webservices/system/util/security/operations/

Request Elements

policy: A filter policy definition. See Filter Policy Definition, page 250.

Response Elements

N/A

Faults

Metadata: If existing metadata precludes the creation of the filter policy as specified.

Security: If user is not member of the admin group, or else lacks ACCESS_TOOLS, READ_ALL_CONFIG and MODIFY_ALL_CONFIG.

rebindResources

Change the bindings of used resources within one or more resources. Within each resource specified in entries, the rebindRule is applied. Each rule changes usage of one resource to another as specified by the rule.

How rebinding is done is resource-specific. Typically, the resource's source text, model, or both are updated to replace the references of oldPath and oldType to newPath and newType.

If a resource in the entries does not currently use a resource identified by oldPath and oldType in the rebindRules, that rule is not applied, but no fault is generated.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of source path-type pairs indicating where to apply rebinding:

- entry
- path: The path to the resource.
- type: Resource type: TABLE, PROCEDURE, LINK, DEFINITION_SET, DATA_SOURCE, or TRIGGER.

rebindRules: List of rebindings to be performed on each resource identified in entries.

- rebindRule (optional): A single rebinding:
- oldPath: The path to the resource that used to be used.
- oldType: The type of the resource that used to be used. See above.
- newPath: The path to the resource that is now to be used.
- newType: The type of the resource that is now to be used. See above.

Response Elements

N/A

Faults

IllegalArgument: If any of the given paths or types are malformed.

NotFound: If any of the resources in entries cannot be found.

Security: If the user does not have READ access on all items in the entry paths other than the last item.

Security: If the user does not have WRITE access to a resource that is to be rebound.

refreshResourceCache

Refresh the cache on a resource. Also see updateResourceCacheConfig, page 225 for enabling and disabling caching.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource with caching enabled.

type: The type of the resource can be either TABLE or PROCEDURE.

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalState: If the cache is disabled or the resource type does not support caching.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

ServerError: If any problems with connecting to or retrieving data from the data source when refreshing.

refreshResourceStatistics

Refresh the statistics on a resource.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Also see updateResourceStatisticsConfig, page 231 for enabling statistics gathering.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource with statistics gathering enabled.

type: The type of the resource. May only be a relational physical DATA_SOURCE or TABLE.

isBlocking (optional): Boolean indicating if call should block until operation completes.

Response Elements

N/A

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the statistics are disabled or the resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

reintrospectDataSource

Deprecated as of API version 6.0. Instead use introspectResourcesTask, page 169.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Performs a reintrospection of the given data source.

If isBlocking is TRUE, this operation does not return until the reintrospect is complete. If isBlocking is FALSE, the call returns immediately with a reintrospect ID that can be used with cancelDataSourceReintrospect, page 48 and getDataSourceReintrospectResult, page 117.

If isBlocking is FALSE and this operation is invoked with an explicit transaction, committing the transaction before the reintrospect completes blocks until it completes. If this operation is invoked with an implicit transaction, the reintrospection is run in the background and commits on success or rolls back on failure. In this case, the reintrospect ID is not found by calls to cancelDataSourceReintrospect or getDataSourceReintrospectResult. A reintrospect on an implicit transaction cannot be canceled or generate a report.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source to be reintrospected.

isBlocking: If TRUE, this call does not return until reintrospection has completed. If FALSE, this call returns immediately regardless of completion.

attributes (optional): Optional values to be used for the resource. These may be required to specify login information if such information is not persisted with the data source definition. See Attributes Element, page 248.

Response Elements

status:

- SUCCESS or FAIL (as appropriate) if the reintrospect completed during this operation.
- INCOMPLETE if the reintrospect is still in progress.

reintrospectReport (optional): If the reintrospect is SUCCESS or FAIL, this report lists what changes occurred during the reintrospection. Otherwise, this element does not exist. See Reintrospect Report Element, page 257.

reintrospectId (optional): If the status is INCOMPLETE, the reintrospect ID is provided.

Faults

IllegalArgument: If the path is malformed.

NotFound: If a data source resource cannot be found at the given path.

IllegalState: If the data source is disabled.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

removeFromCluster

Remove a server node from a cluster.

Location

/services/webservices/system/admin/server/operations

Request Elements

remoteHostName (optional): Host name of the remote server to be removed. Can be omitted if it is the invoking server.

remotePort (optional): Port of the remote server to be removed. Can be omitted if it is the invoking server, or it is the default port.

Response Elements

messages (optional): List of entries describing messages regarding the action. See Messages Element, page 255.

Faults

IllegalState: If the server is not part of a cluster.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

removeLicenses

Unregister one or more licenses from the server. Also see addLicenses, page 39 and getLicenses, page 137.

Location

/services/webservices/system/admin/server/operations

Request Elements

licenseId: List of licenses to be unregistered.

Response Elements

N/A

Faults

NotFound: If any of the given licenses do not exist.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

removeLoginModule

Remove a login module instance from the login sequence.

Location

/services/webservices/system/util/security/operations/

Request Elements

id: The ID of the login module to remove.

Response Elements

N/A

Faults

Invalid ID: If the ID does not designate a valid login module instance.

Security: If the user is not composite\admin.

removePrincipalMapping

Remove a principal mapping from the authentication process.

Location

/services/webservices/system/util/security/operations/

Request Elements

id: The ID of the principal mapping to remove.

Response Elements

N/A

Faults

Invalid ID: If the ID value does not designate a principal mapping.

Security: If the user is not composite\admin.

removeUserFromGroups

Remove a user from one or more groups within a domain.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

userName: The user name.

groupNames: List of groups to remove the user from.

Response Elements

N/A

Faults

NotAllowed: If the user cannot be updated as requested. For example, the group membership might not be updatable to omit the all group.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the user does not exist.

NotFound: If any of the provided groups do not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

removeUsersFromGroup

Remove one or more users from a domain's group.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

groupName: The group name.

userNames: List of user names to remove from the group.

Response Elements

N/A

Faults

NotAllowed: If the group cannot be updated as requested. The group membership might not be updatable, as with the composite domain's "all" group.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the group does not exist.

NotFound: If any of the provided users do not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

renameResource

Rename the resource but not the contents. This call does not modify the script text for a SQL Script procedure.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A source path of the resource to be renamed.

type: The type of the source resource to be renamed. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

newName: The new name of the resource.

Response Elements

N/A

Faults

DuplicateName: If a resource already exists of this type with the new name.

IllegalArgument: If the path is malformed or the type is illegal.

IllegalState: If the resource cannot be renamed. Resources within a physical data source and in the /shared, /services/databases, /services/webservices and user home folders cannot be renamed.

NotFound: If resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in newName.

Security: If the user does not have the ACCESS_TOOLS right.

repairCluster

Repair server nodes and set the timekeeper node.

Location

/services/webservices/system/admin/server/operations

Request Elements

option: Option to perform. Values are: `REGROUP`

remoteHostName (optional): Host name of the remote server to send repair request.

remotePort (optional): Port of the server to send repair request

remoteDomainName (optional): Name of the user domain on the remote server.

remoteUserName (optional): Name of the user on the remote server.

remotePassword (optional): Password of the user on the remote server.

Response Elements

messages: List of entry elements describing messages regarding the action. See Messages Element, page 255.

Faults

IllegalArgument: If the option is malformed.

IllegalState: If the server is not part of a cluster.

RemoteServerError: If error communicating with other cluster nodes.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

Security: If the remote server authentication fails.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG on the remote server.

resourceExists

Check to see if a resource exists.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to check.

type: The type of the resource to check. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

version (optional): Reserved for future use.

Response Elements

exists: TRUE if the resource exists; otherwise FALSE.

Faults

IllegalArgument: If the path is malformed, an illegal type is provided, or any value is provided for version.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have the ACCESS_TOOLS right.

syncDomainGroups

Synchronize the given users in all external domain groups for an external domain.

Location

/services/webservices/system/admin/user/operations

Request Elements:

domainName: The domain name.

names: A list of user names within the domain.

Response Elements:

N/A

Faults:

NotFound: If the domain does not exist.

Security: If the group does not have the ACCESS_TOOLS, READ_ALL_USERS, MODIFY_ALL_USERS rights.

testDataSourceConnection

Test to see if a data source's connection is operational.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the data source to be tested.

Response Elements

status: If SUCCESS, the data source is operational and responds to the test; otherwise, the test status reports FAIL.

messages: List of entry elements describing messages that were generated during the test. See Messages Element, page 255.

Faults

IllegalArgument: If the path is malformed.

NotFound: If the data source does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

unlockResource

Unlock the specified resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of the resource to unlock.

type: The type of the resource to unlock. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

comment (optional): A description about why the resource is being unlocked.

Response Elements

resources: List of the unlocked resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed or the type or detail is illegal.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have WRITE access to the last item in path and does not have the MODIFY_ALL_RESOURCES right.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If the user is not the lock owner and does not have the UNLOCK_RESOURCE right.

unlockResources

Unlock all the specified resources.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

entries: List of path-type pairs of the resources to unlock:

- path: Path to a resource.
- type: Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

comment (optional): A description of why the resource is being unlocked.

Response Elements

resources: List of the unlocked resources. See Resources Element, page 257.

Faults

IllegalArgument: The provided paths are malformed, or types or detail are illegal.

NotFound: If any of the resources or any portions of the paths do not exist.

Security: If the user does not have WRITE access to the last item in path and does not have the MODIFY_ALL_RESOURCES right.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If the user is not the lock owner and does not have the UNLOCK_RESOURCE right.

updateArchiveExportSettings

Update the export settings used or to be used by an archive export. This operation can only be used with an export archive. It can be used anytime after createExportArchive, page 69 is called as long as it is within the same transaction and the archive has not been closed.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the archive.

settings: Description of what is exported for this archive. The settings have the following structure:

- name: The name of the export archive.
- description: A verbose description of the archive.
- type:
  - BACKUP: All information in this archive replaces the server information when imported.
  - ROOT: Resources within the archive cannot be relocated when reimported.
  - PACKAGE: Resources within the archive can be relocated when reimported.
- resources (optional): List of exported resources.
  - all (optional): If set, all resources on the server are exported.
  - resources (optional): Path and type of each resource, plus:

includeChildren (optional): If TRUE or unset, recursively export all child resources. If FALSE, do not include any children.

- users (optional): List of exported users. See Users Element, page 260.
- exportOptions (optional): A space-delimited list of archive options.
- importHints (optional): Hints that can be used during import. See Import Hints, page 252.
- createInfo (optional): Not used. Any setting of this element is ignored.

Response Elements

N/A

Faults

IllegalArgument: If the type is malformed.

IllegalArgument: If any of the resource paths or types are malformed.

IllegalArgument: If any of the settings are malformed or contain illegal values.

IllegalArgument: If any of the server attributes are malformed.

IllegalArgument: If any of the export options are malformed.

IllegalArgument: If any of the import hints are malformed.

IllegalState: This operations can only be called within an explicit transaction context. Use beginTransaction, page 44 and closeTransaction, page 57.

IllegalState: If this operation is called using an import archive ID.

NotAllowed: If an explicitly named resource cannot be exported. The inclusion of implicitly identified resources (using includeChildren) that are not allowed to be exported does not cause this fault.

NotFound: If archive for the archive ID does not exist.

NotFound: If any portion of any of the resource paths and types do not exist.

NotFound: If any of the domains, users, or groups do not exist.

NotFound: If any of the server attributes do not exist.

NotFound: If any of the resources specified in the importHints are not included in the export archive.

NotFound: If any of the users specified in the importHints are not included in any of the resources, privileges, or user data in the export archive.

Security: If the user does not have READ access on all items in the explicitly identified resource paths. Paths to resources implicitly included (through includeChildren) that the user does not have READ access on does not generate this fault.

Security: If the caller does not have admin privileges and attempts to use an export option that is ADMIN ONLY.

Security: If the caller attempts to use an OWNER ONLY export option does not have admin privileges and attempts to use an export option that is ADMIN ONLY.

updateArchiveImportSettings

Update the import settings that are to be used for controlling the archive import.

This operation can only be used with an import archive. It can be used any time after createImportArchive, page 73 is called as long as it is within the same transaction and the archive has not been closed.

Not setting a particular optional element within the settings elements means that no aspect of that setting is applied during import. No checks are made to determine whether any specified resources exist or whether any mapping creates a cyclical relationship.

A number of options can be specified in the importOptions element. If an option is specified, the associated information is exported. Otherwise it is not exported. The valid options are:

- INCLUDE_CACHING: Include caching configurations for resources.
- INCLUDE_CUSTOM_JAVA_JARS: Include custom Java JARs in the export. (ADMIN ONLY)
- INCLUDE_STATISTICS: Include any resources statistics known about the table boundaries, and column boundaries.
- INCLUDE_DEPENDENCY: Gather and include all dependent resources for the resources you choose to export.
- INCLUDE_PHYSICAL_SOURCE_INFO: Include sensitive connection information for included physical sources. (OWNER ONLY)
- INCLUDE_REQUIRED_USERS: Include the information about the required users in the export file.

- INCLUDE_SECURITY: Include resource privilege settings. (OWNER ONLY)
  - If the caller requests an option marked ADMIN ONLY and does not have admin privileges, a Security fault is generated.
  - If the caller requests an option marked OWNER ONLY, that option is applied only to resources where the caller is the owner. If the caller has admin privileges, the option is applied to all resources regardless of ownership. If the option cannot be applied, messages are generated, but no fault occurs during import.

Location

/services/webservices/system/admin/archive/operations/

Request Elements

archiveId: The ID of the archive.

settings: Description of how much of the archive to import and what modifications should be made during import. The settings have the following structure:

- excludeResources (optional): List of path-type pairs indicating resources that should not be imported. By default this is unset.
- relocateResources (optional): A mapping of resources from their location in the archive to where they should be imported. By default this is unset. Each mapping is a path-type pair indicating the “from” location and a path-type pair indicating the “to” location.
- rebindResources (optional): A mapping of resources references within the archive to where they should refer to. By default this is unset. Each mapping is a path-type pair indicating the “from” resource binding and a path-type pair indicating the “to” resource binding.
- rebindUsers (optional): A mapping of users within the archive to whom they should be. By default this is unset. Each mapping is a domainName-userName pair indicating the from user binding and a domainName-userName pair indicating the to user binding.
- remapAttributes (optional): List of resource attribute settings that should be applied on import. By default this is unset.
- map (optional):
  - resource: A path-type pair indicating a resource.
  - attribute: See Attributes Element, page 248.

- importOptions (optional): A space-separated list of name elements (archive options) indicating what additional features should be imported. By default, the same options used for export are used for import.

Response Elements

N/A

Faults

IllegalArgument: If any of the resource paths or types are malformed.

IllegalArgument: If a supplied attribute is malformed.

IllegalArgument: If any of the import options are malformed.

IllegalState: If this operation is called using an export archive ID.

NotAllowed: If the archive type is BACKUP or ROOT and the relocateResources element is set.

NotAllowed: If the archive type is BACKUP and the excludeResources element is set.

NotAllowed: If the archive type is BACKUP and the rebindResources element is set.

NotAllowed: If the archive type is BACKUP and the rebindUsers element is set.

NotAllowed: If the archive type is BACKUP and the remapAttributes element is set.

NotFound: If archive for the archive ID does not exist.

NotFound: If an importOption is set for which the correspond exportOption was not set.

Security: If the caller attempts to set an ADMIN ONLY import option and the caller does not have admin privileges.

updateBasicTransformProcedure

Update the definition of a Basic Transform procedure resource. If the source resource does not currently exist or is not compatible with the transformation, the resource becomes impacted. Only a PROCEDURE with a single output parameter of type XML or a TREE (XML file) is supported.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

transformSourcePath: The location of the resource to be transformed.

transformSourceType: The type of resource (PROCEDURE) to be transformed.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): List of resource-specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If any of the given paths are malformed, or the detail or any of the types are illegal.

NotFound: If the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateCachedResourceStatisticsConfig

Set the cost based optimizer (CBO) statistics configuration for a data source. Cache must be configured before configuring statistics.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a cacheable resource.

type: Cached resource statistics are of type TABLE.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

cachedStatisticsConfig: The statistics configuration of the given resource.

- configured (optional): TRUE if statistics gathering is configured; otherwise FALSE.
- useEnabled (optional): TRUE if gathered statistics are to be used by CBO; otherwise FALSE. This can be used to temporarily disable CBO.
- cardinalityMin (optional): Minimum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityMax (optional): Maximum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityExpected (optional): Expected cardinality for this resource. Setting this value overrides gathered statistics.
- gatherEnabled (optional): Defines what statistics should be gathered for this resource. Valid values are TABLE_BOUNDARY or CUSTOM.
- maxTime (optional): Integer 0 to *n* in minutes. Maximum amount of time the process should spend gathering data; 0 means no limit
- columns (optional): Only applicable if gatherEnabled is set to CUSTOM.
- column (optional): List indicating what specific data to get for each column:

name: Simple column name.

flags: Valid values are NONE, BOUNDARY or ALL.

- onCacheRefresh (optional): TRUE if statistics gathering should be automatically triggered by cache refresh. If FALSE, refresh mode can be specified.
- refresh (optional): How the statistics data should be refreshed.
- mode: How the statistics data should be refreshed (MANUAL or SCHEDULED).
- schedule (if mode is SCHEDULED): See Schedule Element, page 258.

Response Elements

cachedStatisticsConfig (optional): The statistics configuration of the given resource. This element is only present in the response if the detail level is not NONE.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the resource is of the wrong type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateClusterName

Update cluster display name.

Location

/services/webservices/system/admin/server/operations

Request Elements

clusterName: New cluster display name.

Response Elements

N/A

Faults

IllegalState: If the server is not part of a cluster.

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

updateColumnAnnotation

Updates the annotation of a column.

Request Elements

path: A fully qualified path to the column.

detail: The level of detail of resources in the response.Valid values are "NONE", "SIMPLE", or "FULL".

annotation: A description of the column. See Column Element, page 249

Response Elements

resources: A list containing the parent resource of updated column.

Faults

IllegalArgument: If the path is malformed or if the type or detail are illegal.

NotFound: If the column does not exist.

Security: If the user does not have READ access on all items in the path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateConnector

Update the definition of a connector.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

connector: Connector update values. List of connector-type-specific attributes sets the specified attributes but does not alter the value of unspecified attributes. See Connector Element, page 249.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

connector: Updated connector values. Response elements are the same as the request elements.

- name: Identifies the connector that has been updated.
- groupName (optional), and so on.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateCustomDataSourceType

Update a custom data source type.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceType: Resource type of intended data source; updated JDBC URL pattern or driver class name. Other changes are not allowed.

- name
- type
- attributes (optional): List of connector-type-specific attributes. Sets the specified attributes but does not alter the value of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated data source resources. See Resources Element, page 257.

Faults

IllegalArgument: If the name or type is malformed, or the detail or attributes are illegal.

NotFound: If the data source type does not exist.

updateDataServicePort

Update the definition of a TDV data service port container.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: Fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

authMethods (optional): Authentication methods. Valid values are NONE, HTTP_BASIC, HTTP_DIGEST, WSS_USERNAME_TOKEN, NTLM, or KERBEROS.

requireAllAuthMethods (optional): If TRUE, all selected authentication methods are required. If FALSE, any selected authentication method is allowed.

transportSecurity (optional): Transport level security. Valid values are NONE or HTTPS.

bindingType (optional): The WSDL binding type. Valid values are: DOCUMENT_LITERAL, RPC_SOAP, RPC_LITERAL, HTTP_POST, HTTP_GET.

bindingProfileType (optional): binding profile type. Valid values are:

- http://schemas.xmlsoap.org/soap/http
- http://schemas.xmlsoap.org/soap/jms
- http://www.tibco.com/namespaces/ws/2004/soap/binding/JMS

bindingProperties (optional): Parameter for JMS binding profile type.

correlationType (optional): Not used.

isConnectorGroup (optional): Parameter for JMS binding profile type. Typically FALSE.

connector (optional): Parameter for JMS binding profile type. It is used by JMS binding only. Use Manager to create a connector, and set the value of the connector name.

attributes (optional): List of attributes. See Attributes Element, page 248.

annotation: A description of the resource. If not provided, the annotation are left unaltered.

implementation: Points to the operation. Deprecated from 5.0. No need to specify a value.

alternateURL: Deprecated from 5.0. No need to specify a value.

Response Elements

resources: List of the updated data service ports. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateDataSource

Update the definition of a data source resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): List of data source type specific attributes. The specific list of supported attributes vary by data source type. (See getDataSourceAttributeDefs, page 115.) Sets the specified attributes but does not alter the value of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, the detail is illegal, or any of the attribute values provided do not match their definitions.

NotAllowed: If an attempt is made to update a custom Java procedure or custom data source with an insufficient license.

NotFound: If the data source or any portion of path do not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateDataSourceChildInfos

Deprecated as of API version 6.0. Instead use introspectResourcesTask, page 169.

Update the introspect state for child resources within a data source. This causes resources in the namespace to be added or removed as appropriate.

The childInfo paths are paths relative to the data source. The introspect state IGNORED causes the resource to be removed (or stay removed); SELF causes the resource to be added (or stay added); and SELF_AND_CHILDREN causes the resource to be added along with all of its children.

The changes made with this operation are applied when the transaction is committed, or when it is discarded on transaction rollback.

Accessing data source childInfo is relatively expensive. If multiple calls to this operation and/or getDataSourceChildResources are going to be made, making them all on one transaction improves performance.

Also see getDataSourceChildResources and reintrospectDataSource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A path to the data source

childInfos: List of dataSourceChildInfos to be updated.

path: The path within the data source.

type: The type of resource within the data source. Legal values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, TRIGGER.

introspectState: The new state to set. Legal values are IGNORED, SELF, and SELF_AND_CHILDREN.

attributes (optional): Optional values to be used for the resource. These might be required to specify login information if such information is not persisted with the data source definition. See Attributes Element, page 248.

Response Elements

N/A

Faults

DataSourceError: If a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the path element or any path in a childInfo is malformed, or if any type or introspect state on childInfo is illegal.

NotFound: If the data source or any portion of path do not exist.

NotFound: If any of the childInfo paths do not exist and the introspect state is not being set to IGNORED.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

Example

Introspect the table orders: run the webservice and set parameters as follows.

```
updateDataSourceChildInfos
  path=/shared/examples/ds_orders
  childInfos
    childInfo
      path=orders
      type=TABLE
      introspectState=SELF
```

updateDataSourceChildInfosWithFilter

Unsupported in API version 6.0. Instead use introspectResourcesTask, page 169 with an introspection plan.

Updates the introspection state for child resources based upon the specified filter criterion.

Invocation of this procedure adds or removes resources that match the filter set on the path and childInfo path depending on the value passed by introspectState.

Filters created with the API are persistent and they are applied to future introspections (programmatic or manual), so that new tables or directories that match the filter criterion are added or ignored based on those filters and their respective introspection states.

Accessing data source childInfo is relatively expensive. If multiple calls to this operation and/or getDataSourceChildResources, page 115 are going to be made, filtering the request and making them all on one transaction improves performance.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: Path to the data source.

childInfos: List of dataSourceChildInfos to update. For each childInfo:

- path: The path within the data source. The childInfo paths are paths relative to the data source.
- type: The type of resource within the data source. Legal type values are: CONTAINER and TABLE.
- introspectState: The new state to set. Legal introspect state values are:
  - IGNORED: Causes the resource to be removed (or stay removed).
  - SELF: Causes the resource to be added (or stay added).
  - SELF_AND_CHILDREN: Causes the resource to be added along with all of its children.

attributes (optional): Optional values to be used for the resource. These might be required to specify login information if such information is not persisted with the data source definition. See Attributes Element, page 248.

filter: Wild-card character used by the specific datasource to represent any trailing characters.

Response Elements

N/A

Faults

DataSourceError: Given if a data source connection cannot be established or if a data source request returns an error.

IllegalArgument: If the path element or any path in a childInfo is malformed, or if any type or introspect state on a childInfo is illegal.

NotFound: If the data source or any portion of path do not exist.

NotFound: If any of the childInfo paths do not exist and the introspect state is not being set to IGNORED.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

Example

Introspect the data source ds_orders and any other table that starts with the letter "o." Run the webservice API and set parameters as follows:

```
updateDataSourceChildInfosWithFilter
  path="/shared/examples/ds_orders"
  childInfos
    childInfo
      path=o %
      type=TABLE
      introspectState=SELF
        filter="%"
```

In a programmatic invocation of this procedure, the changes made are applied when the transaction is committed, or discarded when the transaction is rolled back. Studio invocation of this procedure implies commit on execution.

Changes are additive. Existing table selections remain selected even if they do not match the current filter, unless the introspectState value is IGNORED.

Datasource filters created with the API are persistent, but the introspectState can be reversed with a separate invocation, or the Studio API can be used to delete the filter while retaining currently introspected tables.

Also see getDataSourceChildResources, page 115 and reintrospectDataSource, page 185.

updateDataSourcePort

Update the definition of a TDV data source port container.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail of resources in the response. Valid values are NONE, SIMPLE, and FULL.

authMethods (optional): A space-separated list containing one or more authentication methods that data service clients can use to authenticate to the server: HTTP_BASIC, HTTP_DIGEST, HTTPS_X509_CERT, WSS_USERNAME_TOKEN, WSS_X509_CERT_BINARY_TOKEN.

requireAllAuthMethods (optional): If TRUE, all selected authentication methods are required. If FALSE, any selected authentication method is allowed.

transportSecurity (optional): The transport security level of the port. It must be one of the following: NONE, HTTPS, HTTPS_WITH_X509_CERT_AUTH.

bindingType (optional)

bindingProfileType (optional)

bindingProperties (optional)

correlationType (optional)

isConnectorGroup (optional)

connector (optional)

attributes (optional): List of attributes. See Attributes Element, page 248.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

timeout (optional)

inputPipeline (optional)

- version
- type
- proprietaryModel (optional)

inputMappingType (optional)

inputMappingOptions (optional)

outputPipeline (optional)

- version
- type
- proprietaryModel (optional)

outputMappingType (optional)

outputMappingOptions (optional)

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If an attempt is made to update an input or output pipeline with an insufficient license.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateDataSourceStatisticsConfig

Update the Cost Based Optimizer (CBO) statistics configuration for a data source. See getDataSourceStatisticsConfig, page 118 for more information.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a resource.

type: The type of the resource. May only be a relational physical DATA_SOURCE.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

- dataSourceStatisticsConfig: The statistics configuration of the given resource:
- configured (optional): TRUE if statistics gathering is configured; otherwise FALSE.
- useEnabled (optional): TRUE if gathered statistics are to be used by CBO; otherwise FALSE. Can be used to temporarily disable CBO.

- tableGatherDefault (optional): Unless overridden at table level, sets the default table configuration. Values are ALL, COLUMN_BOUNDARY, NONE or TABLE_BOUNDARY.
- numThreads (optional): Integer 1 to N; indicates how many threads should be allocated to gather statistics for this data source.
- maxTime (optional): Integer 0 to *n* in minutes. Data source level default for maximum amount of time the process should spend gathering data for each table; 0 means no limit.
- refresh (optional): How the statistics data should be refreshed. See Refresh Element, page 256.

Response Elements

dataSourceStatisticsConfig: Same structure as request elements if request detail is not NONE; otherwise empty.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

NotAllowed: If the resource is not of the write type.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateDataSourceTypeCustomCapabilities

Update the custom capabilities of the data source type.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

dataSourceTypeName: The name of the data source type

customCapabilities: List of custom capability attributes. See Attributes Element, page 248.

Response Elements

updated: Whether or not the update was successful.

Faults

IllegalArgument: If one of the capabilities cannot be updated.

NotFound: If the data source type does not exist.

updateDefinitionSet

Update the contents of a definition set.

The request can contain either sourceDocument or definitions, but not both. Both are optional and it is legal to provide neither. If sourceDocument is provided, the definitions are automatically generated.

XML_SCHEMA_DEFINITION_SET resources are specified with only a source document, and SQL_DEFINITION_SET resources are specified with only a list of definitions.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

sourceDocument (optional): An XML document set that defines this definition set for XML_SCHEMA_DEFINITION_SET resources. If this is a request element, definitions (below) cannot be a request element.

- contents
- charset (optional)
- targetNamespace (optional)
- locationURI (optional)
- schemaLocation (optional)

definitions (optional): Complete list of definitions defined by this definition set for SQL_DEFINITION_SET resources. If this is a request element, sourceDocument (above) cannot be a request element.

- name
- type
- namespace (optional)
- dataType: Information for each data type:
- sqlType (definition plus optional characteristics)
- xmlType (name, namespace, plus optional characteristics)
- pseudoType (definition)
- annotation (optional)
- attributes (optional): Lists of attributes. See Attributes Element, page 248.
- value (optional)

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If path is malformed, the detail is illegal, or any of the attributes, or definition elements are illegal.

IllegalState: If sourceDocument is provided for a SQL_DEFINITION_SET, or if definitions is provided for an XML_SCHEMA_DEFINITION_SET.

NotFound: If the resource or any portion of the path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateDomain

Update a domain. Sets the specified attributes, but does not alter the values of unspecified attributes.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

annotation (optional): A description of the domain. If not provided, the annotation is left unaltered.

attributes (optional): List of attributes. The required attributes vary by domain type. Use getDomainTypeAttributeDefs, page 123 to get the list of attributes that can be updated. See Attributes Element, page 248.

Response Elements

N/A

Faults

IllegalArgument: If any of the given attributes are not valid.

NotAllowed: If any of the given attributes are not updatable.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

Security: If the user does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

updateExternalSqlProcedure

Update the definition of an External SQL procedure (packaged query) resource. If the procedure is updated with a path to a data source that does not exist, the resource becomes impacted, and it does not execute until this is corrected.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

externalSqlText: The native SQL source text.

externalDataSourcePath (optional): A path to the data source in which to run the native SQL. If not provided, the data source is left unaltered.

parameters (optional): List of parameter definitions for this procedure. If not provided, the parameters are left unaltered. See Parameters Element, page 255.

- annotation (optional): A description of this parameter.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource-specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: Either one of the given paths is malformed, or the detail is invalid.

NotFound: If the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateGeneralSettings

Update the general settings for authentication. If a request element is not specified, it is left unchanged.

Location

/services/webservices/system/util/security/operations/

Request Elements

enablePAM (optional): Whether or not to enable PAM.

mayDisallowUser (optional): Whether or not user may be disallowed.

logAuthFailures (optional): Whether or not to log authorization failures.

logPerformance (optional): Whether or not to log performance.

assignModuleGroups (optional): Whether or not to assign module groups.

Response Elements

Resulting settings. Same definitions as for request elements.

Faults

Invalid Input: If the SecurityConfiguration element is missing.

Security: If the user is not composite\admin.

updateGroup

Update a domain group.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

groupName: The group name.

userNames (optional): Names (and optional domains) of users belonging to this group. If not provided, the list of users in the group is left unaltered.

explicitRights (optional): A bit mask for a group's rights. For a table of values, see User and Group Rights Mask, page 259.

annotation (optional): A description of the group. If not provided, the annotation is left unaltered.

Response Elements

N/A

Faults

NotAllowed: If the group cannot be updated as requested. The annotation may not be updatable and the group membership may not be updatable like the composite domain's.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the group does not exist.

NotFound: If any of the provided users do not exist.

Security: If the group does not have the ACCESS_TOOLS and MODIFY_ALL_USERS rights.

updateImplementationContainer

Deprecated as of API version 6.0.

Updates the implementation container of a TDV data service.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

InputPipeline (optional)

inputMappingType (optional)

inputMappingOptions (optional)

outputPipeline (optional)

outputMappingType (optional)

outputMappingOptions (optional)

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If an attempt is made to update an input or output pipeline with an insufficient license.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateLink

Update the definition of a link resource.

If the target resource does not exist, the link becomes impacted.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

targetPath (optional): The fully qualified path of the resource to link to. If not provided, the target path is left unaltered.

targetType (optional): The type of the resource to link to. If not provided, the target type is left unaltered. Only TABLE and PROCEDURE types allowed.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: The path or targetPath is malformed, targetType, or detail are illegal.

NotAllowed: If the targetType is not allowed to be linked. Only TABLE and PROCEDURE resources can be used as the target of a link.

NotAllowed: If an attempt is made to update an input or output pipeline with an insufficient license.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateLoginModule

Update the definition of a login module.

Location

/services/webservices/system/util/security/operations/

Request Elements

id: ID of the login module.

group (optional): Group to which the login module belongs.

enabled (optional): Whether or not the login module is enabled.

properties (optional): Name-value pairs of login module properties.

annotation (optional)

Response Elements

id: ID of the login module.

name: Name of the login module.

bundleName: Name of the bundle to which the login module belongs.

group (optional): Name of the group to which the login module belongs.

enabled (optional): Whether or not the login module is enabled.

bundleEnabled (optional): Whether or not the bundle is enabled.

properties (optional): Name-value pairs of login module properties.

annotation (optional)

Faults

Invalid Input: If an element is malformed, or a required element is missing.

Security: If the user is not composite\admin.

updateLoginModuleList

Update the list of login modules to be used, changing their order in the login sequence.

Location

/services/webservices/system/util/security/operations/

Request Elements

id: One or more login module IDs.

Response Elements

N/A

Faults

Invalid Input: If any ID is not a valid login module.

Missing Identifier: If an ID from the sign-on list is not present in the input list.

Security: If the user is not composite\admin.

updatePrincipalMapping

Update the definition of an existing principal mapping.

Location

/services/webservices/system/util/security/operations/

Request Elements

id: ID of the principal mapping.

type: Type of the principal mapping.

identifier (optional): Text identifier of the principal mapping.

group (optional): Group to which the principal mapping belongs.

Response Elements

Current settings of the same elements as the request elements.

Faults

Invalid Input: If the principal mapping is missing or invalid.

Security: If the user is not composite\admin.

updateResourceAnnotation

Update the annotation of a resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

type: The type of the resource specified by the given path. Valid values are CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, and TRIGGER.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

annotation: A description of the resource.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or if the type or detail is illegal.

NotFound: If the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateResourceCacheConfig

Update the cache configuration for a resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

The updateResourceCacheConfig request has five levels of elements. Elements at the fourth and fifth levels are preceded by their level number in square brackets ([4] and [5]).

path: The path of a cacheable resource.

type: The type of a cached resource can be TABLE or PROCEDURE.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

cacheConfig: The cache configuration of the given resource.

- allOrNothing (optional): This flag only applies to cache groups. TRUE if cache group refresh policy should be all or nothing; FALSE if cache group refresh policy should be best effort. For individual cache groups, this flag always returns TRUE and setting this flag has no effect.
- configured (optional): TRUE if caching should be configured for the given resource; otherwise FALSE. If not provided, the configured setting is left unaltered. If configured is FALSE, all other elements are ignored.
- enabled (optional): TRUE if the cache is enabled; otherwise FALSE. If not provided, the enabled setting is left unaltered.
- incremental (optional): TRUE if the cache is incrementally maintained; otherwise FALSE. If not provided, the incrementally maintained setting is left unaltered.

- storage (optional): How the cached is stored. If not provided, the storage settings is left unaltered.
  - — useDefaultCacheStorage (optional): If TRUE, TDV uses the default cache data source to store cache tables. Only numBuckets and dropCreateIdx values are considered when this value is set to TRUE; the rest are ignored.
  - — mode (optional): The type of storage used for the cache. It is required when useDefaultCacheStorage is not set, or is set to FALSE. May be AUTOMATIC, DATA_SOURCE or DATA_SOURCE_OTPS.
  - — bucketMode (optional): Present when storage mode is DATA_SOURCE_OTPS, ignored otherwise. May be AUTO_GEN or MANUAL.
  - — bucketProperties (optional): Present when bucketMode is AUTO_GEN, otherwise ignored.
  - — [4] bucketCatalog (optional): Database catalog in which to create the bucket.
  - — [4] bucketSchema (optional): Database schema in which to create the bucket.
  - — [4] bucketPrefix (optional): Short string which begins the name of each bucket.
  - — [4] numBuckets: Number of buckets to use for caching.
  - — dropCreateIdx (optional): If TRUE, TDV automatically drops indexes before loading cache data, and creates them after loading.
  - — storageDataSourcePath (optional): If the mode is DATA_SOURCE or DATA_SOURCE_OTPS, this identifies the path to the data source being used to store cache data.
  - — storageTargets (optional): List of targets used for storing cache data. If the mode is DATA_SOURCE or DATA_SOURCE_OTPS, this is a list of the tables. If the mode is DATA_SOURCE_OTPS, storageTargets is required when bucketMode is MANUAL; otherwise, storageTargets is ignored.
  - — [4] entry (optional):
  - — [5] targetName: For a TABLE resource and storage mode DATA_SOURCE this is always result, For a TABLE resource and storage mode DATA_SOURCE_OTPS this may be result, result1, result2, etc. For a PROCEDURE resource, this is the name of a cursor parameter, or an empty string for the scalar output parameters.
  - — [5] path: The path to the table used for storing this data.
  - — [5] type: Always TABLE.

- refresh (optional): How the cache should be refreshed. If not provided, the refresh settings is left unaltered. See Refresh Element, page 256.
  - mode: How the cache should be refreshed. May be MANUAL or SCHEDULED.
  - schedule (optional): If the mode is SCHEDULED, this element has the following child elements:
  - [4] mode (optional): Always INTERVAL.
  - [4] startTime (optional): When the first refresh should occur.
  - [4] fromTomeInADay (optional):
  - [4] endTimeInADay (optional):
  - [4] recurringDay (optional):
  - [4] interval (optional): The number of seconds between refreshes.
  - [4] period (optional):
  - [4] count (optional):
  - [4] isCluster (optional):
- expirationPeriod (optional): The amount of time, in milliseconds, after which the cache is cleared after it is refreshed. If less than zero, the period is set to zero (meaning the cache never expires). If not provided, the enable setting is left unchanged.
- firstRefreshCallback (optional): A path to a procedure with zero input elements that should be invoked before the cache refresh.
- secondRefreshCallback (optional): A path to a procedure with zero input elements that should be invoked after a successful or failed cache refresh.
- clearRule (optional): NONE, ON_LOAD, or ON_FAILURE.

  The normal behavior is that old cache data is cleared on expiration, or when a cache refresh successfully completes and the old cache data is replaced by the new cache data. If no clearRule is provided, the enable setting is left unaltered.
  - If NONE, just the normal behavior occurs.
  - If ON_LOAD, the normal behavior occurs, but the old cache data is immediately cleared as the refresh is started.
  - If ON_FAILURE, the normal behavior occurs, and the old cache data is cleared if the refresh fails.

Response Elements

cacheConfig (optional): The cache configuration of the given resource. This element is only present in the response if the detail level is not NONE. Response subelements are the same as request elements, but with updated values.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalState: If the resource type does not support caching.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: The user must have READ access on all items in path.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateResourceEnabled

Update the enabled state of a resource. DATA_SOURCE and TRIGGER resources can be enabled or disabled.

Note: This *web services operation* is different from the *procedure* of the same name, which is in the resource tree under /lib/resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path to the resource.

type: The resource type either DATA_SOURCE or TRIGGER.

detail: The level of detail to include in the response. Valid values are NONE, SIMPLE, and FULL.

enabled: If TRUE, the resource becomes enabled. If FALSE, the resource becomes disabled.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the type or detail is illegal.

IllegalState: If the target resource does not support enabling or disabling. DATA_SOURCE and TRIGGER resources can be enabled or disabled.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateResourcePrivileges

Set the privilege information for a list of resources.

Only a user with GRANT privilege on a resource can modify the privileges for that resource. The owner of a resource always has GRANT privilege, as do users with the MODIFY_ALL_RESOURCES right.

When mode is OVERWRITE_APPEND, or is not supplied, privileges are applied by user or by group, so that updating privileges for one user or group does not alter privileges from any other user or group. The privileges applied for a user or group replace the previous value for that user or group. When mode is SET_EXACTLY, all privileges on the resource are made to look exactly like the provided privileges.

When updateRecursively is FALSE, the privileges are applied only to the specified resources. When it is TRUE, the privileges are recursively applied into any CONTAINER or DATA_SOURCE resource specified. When recursively applying privileges, the privilege change is ignored for any resource for which the user lacks owner privileges.

Privileges that are not applicable for a given resource type are automatically reduced to the set that is legal for each resource:

- TABLE resources support NONE, READ, WRITE, SELECT, INSERT, UPDATE, and DELETE.
- PROCEDURE resources support NONE, READ, WRITE, and EXECUTE.
- All other resource types only support NONE, READ, and WRITE.

The combinedPrivs and inheritedPrivs elements on each privilegeEntry are ignored and can be left unset.

Note: Operations Reference, page 37, contains an example of how to set the request elements for this operation.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

updateRecursively: If TRUE, all children of the given resources are recursively updated with the privileges assigned to their parent.

updateDependenciesRecursively (optional): If TRUE, all dependencies of the given resources are recursively updated with the privileges assigned to their parent.

privilegeEntries: List of resource names, types, and privileges.

mode (optional): Determines whether to merge privileges with existing ones:

- OVERWRITE_APPEND: Merges and does not update privileges for users or groups not mentioned. Default.
- SET_EXACTLY: Makes privileges look exactly like those provided in the call.

Response Elements

N/A

Faults

IllegalArgument: If any path is malformed, or any type or privilege entry is illegal, or mode is not one of the legal values.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If a path refers to a resource that does not exist.

NotFound: If an unknown domain is provided.

NotFound: If an unknown user is provided.

NotFound: If an unknown group is provided.

Security: If for a given entry path the user does not have READ access on any item in a path other than the last item, or does not have GRANT access on the last item.

Security: If the user does not have the ACCESS_TOOLS right.

updateResources

Update resources.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

resources: List of the resources. See Resources Element, page 257.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If any of the given paths, types, or detail levels are malformed.

NotAllowed: If an attempt is made to update a custom Java procedure, SQL script, or a trigger with an insufficient license.

NotFound: If the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateResourceStatisticsConfig

Update the statistics configuration for a resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: The path of a resource.

type: The type of the resource can only be a physical relational TABLE.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

statisticsConfig: The statistics configuration of the given resource.

- cardinalityMin (optional): Minimum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityMax (optional): Maximum cardinality for this resource. Setting this value overrides gathered statistics.
- cardinalityExpected (optional): Expected cardinality for this resource. Setting this value overrides gathered statistics.
- gatherEnabled (optional): Defines what statistics should be gathered for this resource. Valid values are DEFAULT, CUSTOM, DISABLED or TABLE_BOUNDARY. If not provided, the enable setting is left unaltered.
- maxTime (optional): From -1 to *n* in milliseconds. If not provided, the enable setting is left unaltered. 0 means no timeout; -1 means use data source default; 0 or greater overrides data source setting.
- columns (optional): Only applicable if gatherEnabled is set to CUSTOM.
- column: What specific data to get for each column.

  name: Column name.

  flags: Valid values are NONE, BOUNDARY or ALL.

Response Elements

statisticsConfig (optional): The statistics configuration of the given resource. This element is only present in the response if the detail level is not NONE.

- cardinalityMin (optional): Minimum cardinality for this resource.
- cardinalityMax (optional): Maximum cardinality for this resource.
- cardinalityExpected (optional): Expected cardinality for this resource.
- gatherEnabled (optional): What statistics should be gathered for this resource.
- maxTime (optional): Maximum amount of time the process should spend gathering data; 0 means no limit.
- columns (optional): Only available if gatherEnabled is set to CUSTOM.
- name: Simple column name.
- flags: NONE, BOUNDARY or ALL.

Faults

IllegalArgument: If the path is malformed or an illegal type is provided.

IllegalArgument: If statsGatherEnabled is not one of the valid values.

IllegalArgument: If a column does not support supplied flags.

NotAllowed: If the resource type is not of the right type or its parent data source is not configured.

NotFound: If the resource or any portion of the path to the resource does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access on the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateServerAttributes

Update the server with the given attributes.

Location

/services/webservices/system/admin/server/operations

Request Elements

attributes: List of attributes to update. The names of the attributes should be fully qualified paths in the server attribute namespace. See Attributes Element, page 248.

Response Elements

N/A

Faults

IllegalArgument: If any of the provided attributes are malformed.

NotFound: If any of the attributes do not exist in the server.

Security: If the user does not have MODIFY_ALL_CONFIG right.

updateServerName

Update the server display name.

Location

/services/webservices/system/admin/server/operations

Request Elements

serverName: New server display name.

Response Elements

N/A

Faults

Security: If the user does not have both ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

updateSqlScriptProcedure

Update the definition of a SQL Script procedure resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

scriptText: The SQL Script source text.

scriptModel (optional): A model of the SQL Script. If this element is not provided, the model is cleared.

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer in the following element. FALSE if they were derived from the resource.

parameters (optional): The parameter definitions of the procedure. This element is ignored (even if provided) unless isExplicitDesign is present and TRUE.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource-specific attributes. Sets the specified attributes, but does not alter the values of unspecified attributes.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateSqlTable

Update the definition of a SQL Table resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

- detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

sqlText: The SQL source text.

sqlModel (optional): A model for the display of the SQL. If this element is not provided, the model is cleared.

- version: Version of the SQL model.
- type: SQL model type.
- proprietaryModel (optional)

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer; FALSE if they were derived from the resource.

columns (optional): The column definitions of the table. This element is ignored (even if provided) unless isExplicitDesign is present and TRUE. See Column Element, page 249.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): List of resource-specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateStreamTransformProcedure

Update the definition of any Stream Transform procedure resource. If the source resource does not currently exist or is not compatible with the transformation, the resource becomes impacted. Only a PROCEDURE with a single output parameter, of type XML or a TREE (XML file) is supported.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

transformSourcePath: The location of the resource to be transformed.

transformSourceType: PROCEDURE is the type of the resource to be transformed.

streamModel (optional): A model of the stream transformation. If not present, the model is cleared.

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer; FALSE if they were derived from the resource.

parameters (optional): The parameter definitions of the procedure. This element is ignored (even if provided) unless isExplicitDesign is both present and TRUE.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource-specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If parameters includes INOUT parameters.

NotFound: If the resource or any portion of path does not exist.

Security: If user does not have READ access on all items in path.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateTransformProcedure

Update the definition of a transform procedure resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

transformModel (optional): A model of the transformation:

- version: Version of the transformation model.
- type: Transformation model type.
- proprietaryModel (optional)

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): List of resource-specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path or transformModel is malformed, or detail or attributes are illegal.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateTrigger

Update the definition of a trigger resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

enabled (optional): TRUE to enable the trigger, FALSE to disable it.

conditionType (optional): The type of condition the trigger should wait for.

conditionSchedule (optional): See Schedule Element, page 258.

conditionAttributes (optional): The attributes that define the filter for the conditionType. This element must be provided if (and only if) conditionType is provided. See Attributes Element, page 248.

actionType (optional): The type of action the trigger should take when the condition occurs.

actionAttributes (optional): List of the attributes that define the action that should be taken when the trigger occurs. This element must be provided if (and only if) actionType is provided. See Attributes Element, page 248.

maxEventsQueued (optional): The maximum number of triggering events to queue.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path or detail are illegal.

IllegalState: If the specified conditionType or actionType is not supported, or the attributes for that type are not properly configured.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateUser

Update a domain user's password, group membership, rights or annotation.

The password element is ignored for non-COMPOSITE domains. The oldPassword element is required when callers are changing their own password. Users with the MODIFY_ALL_USERS right do not need to specify the oldPassword element when changing other users' passwords.

Updating a user's group membership is a set operation.

Only users with the MODIFY_ALL_USERS right can use the updateUser call.

Location

/services/webservices/system/admin/user/operations/

Request Elements

domainName: The domain name.

userName: The user name.

oldPassword (optional): An old user password. This is silently ignored if the domain is not a COMPOSITE domain.

password (optional): A new user password. This is silently ignored if the domain is not a COMPOSITE domain.

groupNames (optional): Names and optional domains of groups to which the user belongs. If not provided, the group membership is left unaltered.

explicitRights (optional): A bit mask for a user's rights. For a table of values, see User and Group Rights Mask, page 259.

annotation (optional): A description of the user. If not provided, the annotation is left unaltered.

Response Elements

N/A

Faults

NotAllowed: If the user cannot be updated as requested. For example, the annotation may not be updatable (such as the composite user admin), the password may not be updatable in an LDAP domain, or the group membership may not be updatable to omit the "all" group.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the domain does not exist.

NotFound: If the user does not exist.

NotFound: If any of the provided groups do not exist.

Security: If the user does not have the ACCESS_TOOLS right.

Security: If the user does not have the MODIFY_ALL_USERS right and is updating any user other than self or is updating anything other than the password.

Security: If the user is modifying his or her own password and oldPassword is not correct.

updateUserLockState

Change the lockout state for a user.

Location

/services/webservices/system/admin/user/operations/

Request Elements

userName: user@domain

lockUser: whether the user is to be locked or unlocked

Response Elements

locked: Whether the new status of the user is locked or unlocked; note that a user attempting to sign on rapidly could trigger a new locked state during invocation, so an unlock returning FALSE is not necessarily a failure.

Faults

Security: If the user does not have MODIFY_ALL_USERS right.

updateXQueryProcedure

Update the definition of a XQuery procedure resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

xqueryText: The XQuery source text.

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer in the following element. FALSE if they were derived from the resource.

parameters (optional): The parameter definitions of the procedure. This element is ignored (even if provided) unless isExplicitDesign is present and TRUE. See Parameters Element, page 255.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes, but does not alter the values of unspecified attributes.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateXQueryTransformProcedure

Update the definition of an XQuery Transform procedure resource.

If OUTPUT parameters are provided in the parameters element, they are ignored. The OUTPUT parameter is always derived from the model, even if the explicitlyDesigned element is TRUE.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

xqueryText (optional): The XQuery source text.

xqueryModel (optional): A model of the XQuery transformation. If this element is provided, neither the xqueryText nor parameter elements should be provided.

- version: Version of the XQuery transformation procedure.
- type: Type of the XQuery transformation procedure.
- proprietaryModel (optional)

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer. FALSE if they were derived from the resource.

parameters (optional): List of parameter definitions for the procedure. This element is ignored (even if provided) unless isExplicitDesign is both present and TRUE. See Parameters Element, page 255.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If the call contains both the xqueryModel and xqueryText elements, or the xqueryModel and parameters elements.

NotAllowed: If parameters includes INOUT parameters.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateXSLTProcedure

Update the definition of an XSLT procedure resource.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

xsltText: The XSLT source text.

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer in the following element. FALSE if they were derived from the resource.

parameters (optional): List of parameter definitions of the procedure. This element is ignored (even if provided) unless isExplicitDesign is present and TRUE. See Parameters Element, page 255.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes, but does not alter the values of unspecified attributes. See Attributes Element, page 248.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If an attempt is made to use this operation with an insufficient license.

NotFound: If the resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

updateXsltTransformProcedure

Update the definition of an XSLT Transform procedure resource.

If the source resource does not currently exist or is not compatible with the transformation, the resource becomes impacted.

Only a PROCEDURE with a single output parameter of type XML, or a TREE (XML file), is supported.

Location

/services/webservices/system/admin/resource/operations/

Request Elements

path: A fully qualified path to the resource.

detail: The level of detail about the resources to include in the response. Valid values are NONE, SIMPLE, and FULL.

transformSourcePath: The location of the resource to be transformed.

transformSourceType: The type of resource to be transformed.

xsltText (optional): The XSLT source text.

xsltModel (optional): A model of the XSLT transformation. If this element is provided, neither the xsltText nor parameters elements should be provided.

- version: Version of the XSLT transformation procedure.
- type: Type of the XSLT transformation procedure.
- proprietaryModel (optional)

isExplicitDesign (optional): TRUE if the parameters were provided by the resource designer. FALSE if they were derived from the resource.

parameters (optional): List of parameter definitions of the procedure. This element is ignored (even if provided) unless isExplicitDesign is both present and TRUE. See Parameters Element, page 255.

- annotation (optional): Description of the parameter.

annotation (optional): A description of the resource. If not provided, the annotation is left unaltered.

attributes (optional): Resource specific attributes. Sets the specified attributes but does not alter the values of unspecified attributes.

Response Elements

resources: List of the updated resources. See Resources Element, page 257.

Faults

IllegalArgument: If the path is malformed, or the detail or attributes are illegal.

NotAllowed: If the call contains both the xsltModel and xsltText elements, or the xsltMode and parameters elements.

NotAllowed: If parameters includes INOUT parameters.

NotFound: If the target resource or any portion of path does not exist.

Security: If the user does not have READ access on all items in path other than the last one.

Security: If the user does not have WRITE access to the last item in path.

Security: If the user does not have the ACCESS_TOOLS right.

Recurring Element Structures

This section describes hierarchies of elements that occur frequently in operation requests and responses. Rather than listing each of these structures repeatedly in detail, this section lists each structure once so it can be cross-referenced from the operations in which it occurs.

This section has the following topics:

Attribute Definitions Element

The attributeDefs element contains a list of attribute definitions, each of which has the following format:

- attributeDef (optional):
- name: Name of the attribute. In some contexts, includes a full path name.
- type: Attribute type; for example, LIST, BOOLEAN, INTEGER, STRING, or PASSWORD_STRING.
- updateRule: Whether the attribute is READ_ONLY or READ_WRITE.
- annotation (optional): Text notes about the attribute.
- required (optional): A BOOLEAN indicating whether or not the attribute must be present.

- defaultValue (optional): Value if none is explicitly set; for example, TRUE or expr$1 or 100.
- pattern (optional): Template to use for strings, passwords, and so on.
- minValue (optional): Minimum allowed value for the attribute.
- maxValue (optional): Maximum allowed value for the attribute.
- allowedValues (optional): List of allowed values:
- item: A member of a list of allowed values.
- suggestedValues (optional): List of suggested values:
- item: A member of a list of suggested values.
- displayName (optional): Text string to display in connection with this attribute.
- unitName (optional)
- parentName (optional): Required unless this is the root attribute definition.
- visible (optional): Whether or not the attribute is visible in the UI.
- editorHint (optional)
- dependencyExpression (optional)

Attributes Element

The attributes element appears in many request and response elements, and even in multiple places inside other elements. It can also be iterative; for example, an item in the valueList can itself contain a valueList, an entry in a valueMap can have a key element with its own valueMap, and so on, down to many levels.

An attributes element is a list of one or more attributes, each of which has the following format:

- attribute:
  - name: Name of the attribute.
  - type: Attribute type.
  - value (optional): The attribute's value. The attribute can have a list of values instead (next).
  - valueList (optional): List of value items.

  item:

  type: Item type.

value: Item value.

valueList: List of values, if the item has more than one.

valueMap

valueArray: Array of value items.

— valueMap (optional):

entry:

- key (type, value, valueList, valueMap, valueArray)
- value (type, value, valueList, valueMap, valueArray)
  - — valueArray (optional): Array of value items.
  - — item
  - — unset (optional)

Column Element

The column element defines one column in a table.

- column: Column ID.
- name: Name of column.
- dataType: Information for each data type:
  - — sqlType (definition plus optional characteristics)
  - — xmlType (name, namespace, plus optional characteristics)
  - — pseudoType (definition)
- isPrimaryKey (optional): Whether this column is a primary key in the database.
- attributes (optional): See Attributes Element, page 248.
- isNullable (optional): Whether or not the column can contain NULL values.
- annotation (optional): A description of the column.

Connector Element

A connector element has the following format:

- name: Identifies the connector to update.
- groupName (optional): Connector group.

- annotation (optional): A description of the connector. If not provided, the annotation is left unaltered.
- connectorType: The type of connector.
- attributes (optional): List of connector-type-specific attributes. In the connector request element, only specified attributes can change attribute values. Response values and attributes reflect the new connector state. See Connector Element, page 249.

Domains Element

The domains element contains a list of domain elements, each describing one domain. If the detail request element is SIMPLE only the name, domainType, and annotation are returned. If FULL, attributes are also returned.

- domain (optional):
  - name: The name of the domain.
  - domainType The domain type.
  - annotation: A description of the domain.
  - attributes: List of attributes. See Attributes Element, page 248.

Filter Policy Definition

A policy element contains a filter policy definition. The definition has the following format:

- name: The short name of the filter policy.
- folder: The folder path of the procedure to create to implement the policy.
- enabled: Whether the policy is enabled or disabled.
- form: Whether the filter policy is specified in tabular form (FORM) or using explicit SqlScript code (CODE).
- policyGroup (optional):
  - joinType: INNER, OUTER or UNION.
  - policyList (may be optional): List of policy procedure paths.
  - policyProcedurePath (optional): Path to a policy procedure.

- defaultRule (optional): For FORM policies, the default rule to follow if no users or groups match the current user.
  - filter: ALL (all rows), NONE (no rows), PREDICATE (SQL predicate), or PROCEDURE (predicate computed by procedure).
  - data (optional): For PREDICATE, the SQL text to be used. For PROCEDURE, the path of the procedure to invoke.
- memberRuleList: For FORM policies, list of rules to apply when user or group criteria match. For each:
  - memberRule (optional): Name of rule.
  - member: The user or group to which the rule applies.
  - domain: The user/group domain.
  - name: The user/group name.
  - type: USER or GROUP.
  - filter: ALL (all rows), NONE (no rows), PREDICATE (SQL predicate), or PROCEDURE (predicate computed by procedure).
  - data (optional): For PREDICATE, the SQL text to use. For PROCEDURE, the path of the procedure to invoke.
- assignmentList: List of assignments:
  - assignment (optional)
- notes (optional): Any notes or descriptive text.

Groups Element

The groups element contains one or more group elements with the following format:

- name: Name of the group.
- domainName: Name of domain to which the group belongs.
- id: Group ID.
- explicitRights. For a table of values, see User and Group Rights Mask, page 259.
- effectiveRights. For a table of values, see User and Group Rights Mask, page 259.
- inheritedRights. For a table of values, see User and Group Rights Mask, page 259.

- annotation (optional): Description of the group.
- userNames (optional): List of user entry elements:
  - entry: Name of the entry.
  - name: Name of the user.
  - domain (optional): User's domain.

Import Hints

The importHints element is a list of hints to use during import.

- rebindResources: List of path-type pairs of resources that should be rebound when imported.
  - entry: Name of the entry.
  - path: Path to the resource to rebind.
  - type: Type of resource to rebind.
- rebindUsers: List users that should be mapped to other users on import.
  - domain (0 or more): A domain that contains exported users.
  - name: The name of the domain containing the users.
  - all (optional): If set, all users within this domain are exported.
  - userNames (optional): A space-delimited list of user names.
- name: List of names within the domain.
- remapAttributes: List of resource attributes that should be mapped during import:
  - resource: List of resources to remap. For each:
  - path: Path to the attribute to remap.
  - type: Type of attribute to remap.
  - attributeNames: A space-delimited list of remapped-attribute names.

Introspection Plan Element

An plan element for introspection contains the following:

- updateAllIntrospectedResources (optional): Whether or not to update all currently introspected resources.

- failFast (optional): If TRUE, the introspection fails when the first error occurs. If FALSE (the default), the plan runs to completion as a best effort.
- commitOnFailure (optional): If TRUE, the introspection commits whatever it can. If fastFail is also TRUE, only resources successfully introspected up to that point are committed. The default is FALSE.
- autoRollback (optional): If TRUE, the introspection task rolls back rather than being committed. Supersedes all commit options. This enables you to perform a dry run of resource introspection. If TRUE, you can use introspectResourcesResult, page 167. If FALSE or unset, the introspection is not automatically rolled back.
- scanForNewResourcesToAutoAdd (optional): If TRUE, the introspection task scans for native resources that have been newly added to the data source. If a newly added resource is found and its parent container has autoAddChildren set, that child is automatically introspected.
- entries (optional): List of resources to introspect and the action to take for each.
  - entry (optional): Name of the resource.
  - resourceId: Identifier for the resource.
  - path: Path to the resource.
  - type: Type of resource (CONTAINER, DATA_SOURCE, DEFINITION_SET, LINK, PROCEDURE, TABLE, TREE, TRIGGER).
  - subtype
  - action
  - attributes (optional): List of attributes. See Attributes Element, page 248.

Introspection Report Status Element

An introspection report status element has the following format:

- status: Status of the status element.
- introspectorVersion: Version number of the introspector used.
- startTime (optional): Timestamp of the time the introspection began.
- endTime (optional): Timestamp of the time the introspection ended.
- addedCount
- removedCount
- updatedCount

- skippedCount
- totalCompletedCount
- toBeAddedCount
- toBeUpdatedCount
- totalToBeCompletedCount
- warningCount
- errorCount
- report (optional):
  - entry:
  - path
  - type
  - subtype
  - action
  - durationMs: Elapsed time to introspect this data source, in milliseconds.
  - status
  - messages (optional) See Messages Element, page 255.

Licenses Element

The licenses element is a list of license elements and their subelements. See Licenses Element, page 254.

- license:
  - id
  - product
  - version
  - creationDate
  - activationDate (optional)
  - duration
  - expirationDate (optional)
  - type (optional)
  - restriction
  - owner
  - valid
  - encryptionStrength

Messages Element

The messages element contains one or more entry elements, each of which contains the information for one message (a fault message, an archive report, or some other type of message).

- entry:
  - code
  - name
  - message (optional)
  - detail (optional)
  - severity (optional)

Parameters Element

The parameters element is a list of definitions, one per parameter.

- name

- dataType
  - sqlType
  - xmlType
  - pseudoType
- direction
- isNullable
- attributes (optional): See Attributes Element, page 248.
- annotation (optional): A description of this parameter.

Refresh Element

A refresh element contains information on how cache data is to be refreshed or a trigger is to be updated. For some operations, this element contains mode and schedule only.

- mode: How the data, cache, or trigger should be refreshed: MANUAL or SCHEDULED.
- schedule (if the mode is SCHEDULED): See Schedule Element, page 258.
- expirationPeriod (optional): The number of milliseconds after which the cache is cleared when it has been refreshed. If zero, the cache never expires.
- firstRefreshCallback (optional): An optional path pointing to a procedure with zero input elements that should be invoked before the cache refresh.
- secondRefreshCallback (optional): An optional path pointing to a procedure with zero input elements that should be invoked after a successful or failed cache refresh.
- clearRule: NONE, ON_LOAD, or ON_FAILURE.

  The normal behavior is that old cache data is cleared on expiration, or when a cache refresh successfully completes and the old cache data is replaced by the new cache data. If no clearRule is provided, the enable setting is left unaltered.
  - If NONE, just the normal behavior occurs.
  - If ON_LOAD, the normal behavior occurs, but the old cache data is immediately cleared as the refresh is started.
  - If ON_FAILURE, the normal behavior occurs, and the old cache data is cleared if the refresh fails.

Reintrospect Report Element

A reinstrospectReport element can contain multiple changeEntry elements, each of which has the following format:

- changeEntry:
  - code
  - name
  - message (optional)
  - detail (optional)
  - severity (optional)
  - path

Resources Element

A resources element can contain one or more resources, each of which has the following format:

- resource (optional):
- name
- path
- type
- subtype
- id (optional)
- changeId (optional)
- version (optional)
- IntrospectState (optional)
- ownerDomain (optional)
- ownerName (optional)
- impactLevel (optional)
- impactMessage (optional)
- enabled (optional)
- lockState (optional)
- lockOwnerDomain
- lockOwnerName

- lockCreateTime
- lockParentPath (optional)
- hints (optional): List of hints:
- name
- type
- value (optional)
- valueList (optional)
- valueMap (optional; key-value pairs)
- valueArray (optional)
- unset (optional)
- annotation (optional)
- attributes (optional): List of attributes. See Attributes Element, page 248.

Schedule Element

In the Refresh Element, page 256, if the mode element is SCHEDULED, the schedule element exists; otherwise it does not. For updateTrigger, page 238, the schedule element has the name conditionSchedule.

- schedule (optional):
  - mode (optional): INTERVAL or CALENDAR. (For cache, always INTERVAL.)
  - startTime (optional): When the first refresh should occur.
  - fromTimeInADay (optional)
  - endTimeInADay (optional)
  - recurringDay (optional)
  - interval (optional): The number of seconds between refreshes.
  - period (optional): Present if mode is CALENDAR. Values are HOUR, DAY, WEEK, MONTH.
  - count (optional): Present if mode is CALENDAR.
  - isCluster (optional)

User and Group Rights Mask

Several Web Services operations (createUser, page 80; updateUser, page 240; createGroup, page 72; and updateGroup, page 219) use a bit mask to designate explicitRights among the request elements. The table below lists the bit assignments for user and group rights. These values are additive and expressed in decimal. For example, to grant a user ACCESS_TOOLS, READ_ALL_RESOURCES, and MODIFY_ALL_STATUS rights explicitly, specify a bit mask value of 517.

User Right	Bit Mask Value
ACCESS_TOOLS	1
[reserved]	2
READ_ALL_RESOURCES	4
MODIFY_ALL_RESOURCES	8
READ_ALL_USERS	16
MODIFY_ALL_USERS	32
READ_ALL_CONFIG	64
MODIFY_ALL_CONFIG	128
READ_ALL_STATUS	256
MODIFY_ALL_STATUS	512
UNLOCK_RESOURCE	1024

User Element

The user element contains elements describing an individual user:

- name
- domainName
- id
- explicitRights
- effectiveRights
- inheritedRights

- annotation (optional)
- groupNames (optional): List of groups to which the user belongs:
  - entry:
  - name
  - domain (optional)

Users Element

The users element specifies either all users or subsets of users based on domains, groups, user names, or combinations of them.

- all (optional): If set, all domains, users, and groups.
- domains (optional): List of domains in their entirety.
  - all (optional): If set, all domains.
  - domains: A space-delimited list of domain names.
- users (optional): List of users.
  - domain (0 or more): A domain that contains users.
  - name: The name of the domain containing the users.
  - all (optional): If set, all users within this domain.
  - userNames (optional): A space-delimited list of user names.
- groups (optional): List of groups in their entirety. Includes user membership.
  - domain (0 or more): A domain that contains groups.
  - name: The name of the domain containing the groups.
  - all (optional): If set, all groups within this domain.
  - group (optional): List of groups and their definitions.
- name: The name of the group.
- all (optional): If set, all users within this group are listed as part of the group.
- userNames (optional): List of users to be listed as members of the group.

TDV Resource Types and Subtypes

The following table lists the types and subtypes of TDV resources.

Resource Type / Subtype	Description
CONTAINER / CATALOG_CONTAINER	A catalog folder within a data source, under /services/databases.
CONTAINER / CONNECTOR_CONTAINER	A container for connectors.
CONTAINER / DIRECTORY_CONTAINER	A TDV directory.
CONTAINER / FOLDER_CONTAINER	A TDV folder. It can be created only in another folder under /services/webservices.
CONTAINER / OPERATIONS_CONTAINER	A web service container for operations.
CONTAINER / PORT_CONTAINER	A port container. It can be created only within a SERVICE under /services/webservices.
CONTAINER / SCHEMA_CONTAINER	A schema container. It can be created only within a catalog under /services/databases.
CONTAINER / SERVICE_CONTAINER	A service container. It can be created only within a data source that is under /services/webservices.
CONNECTOR / JMS	A JMS connector, created with no connection information.
CONNECTOR / HTTP	An HTTP connector, created with no connection information.
DATA_SOURCE / FILE_DATA_SOURCE	A comma-separate file data source.
DATA_SOURCE / NONE	A custom Java procedure data source.
DATA_SOURCE / RELATIONAL_DATA_SOURCE	A relational database source.
DATA_SOURCE / WSDL_DATA_SOURCE	A TDV web service data source.

Resource Type / Subtype	Description
DATA_SOURCE / XML_FILE_DATA_SOURCE	An XML file data source.
DATA_SOURCE / XML_HTTP_DATA_SOURCE	An HTTP XML data source.
	An abstract WSDL definition set such as the ones imported from Designer.
	An SCA definition set imported from Designer.
DEFINITION_SET / SQL_DEFINITION_SET	A SQL definition set.
DEFINITION_SET / WSDL_DEFINITION_SET	A WSDL definition set.
DEFINITION_SET / XML_SCHEMA_DEFINITION_SET	An XML schema definition set.
LINK / NONE	A resource published in /services.
PROCEDURE / BASIC_TRANSFORM_PROCEDURE	A Basic XSLT Transformation procedure. It is created with no target procedure and no output columns, so it is not runnable.
PROCEDURE / DATABASE_PROCEDURE	A database stored procedure.
PROCEDURE / EXTERNAL_SQL_PROCEDURE	A packaged query. It is created with no SQL text, so it is not runnable.
PROCEDURE / JAVA_PROCEDURE	A procedure created from a Java data source (JAR file).
PROCEDURE / OPERATION_PROCEDURE	A web service or HTTP procedure operation.
PROCEDURE / SQL_SCRIPT_PROCEDURE	A SQL procedure, created with a simple default script body that is runnable.
PROCEDURE / STREAM_TRANSFORM_PROCEDURE	An XSLT streaming transformation procedure. It is created with no target procedure and no output columns, so it is not runnable.

Resource Type / Subtype	Description
PROCEDURE / XQUERY_TRANSFORM_PROCEDURE	An XQUERY transformation procedure. It is created with no target schema and no model, so it is not runnable.
PROCEDURE / XSLT_TRANSFORM_PROCEDURE	An XSLT transformation procedure. It is created with no target procedure and no output columns, so it is not runnable.
TABLE / DATABASE_TABLE	A database table.
TABLE / DELIMITED_FILE_TABLE	A delimited file table.
TABLE / SQL_TABLE	A TDV view. It is created with no SQL text or model, so it is not runnable.
TABLE / SYSTEM_TABLE	A system table view.
TREE / XML_FILE_TREE	The XML tree structure associated with a file-XML data source.
TRIGGER / NONE	A TDV trigger. It is created disabled.

Built-in Procedures

TDV provides a standard procedure library, similar to a utility library for an Oracle database. The built-in procedures extend the TDV SQL Script language through classes. These procedures function exactly like TDV's custom Java procedures.

This topic describes the TDV built-in procedures, and provides the syntax for the debugging and general utility groups of procedures. This chapter also provides an extended example that invokes several JMS procedures, and lists the SQL definition sets that accompany various groups of procedures.

About TDV Built-in Procedures

You can call built-in procedures from any other procedure. You can also publish built-in procedures under Data Services and call them from client applications. Some procedures are available as published resources when TDV is installed.

Note: For a general description of how to design procedures to query and manipulate data stored in data sources, see "Designing Procedures" in the *TDV User Guide*.

The path to a built-in procedure (for example, /lib/debug/) is automatically added to every script. There is no need to provide fully qualified names for built-in procedures when you call them from other procedures.

- You can let the cursor hover over the name of the built-in procedure in the Studio resource tree to see a tool tip with a short description.
- You can open any procedure and select the Info tab to find a short description for the procedure, as well as its inputs, outputs, and exceptions.
- You can view valid procedure input parameters by pointing a browser to the admin or util Web services definition and displaying the contracts at these URLs:

http://[SERVERNAME]:[PORT]/services/system/admin?wsdl

http://[SERVERNAME]:[PORT]/services/system/util?wsdl

For example, active links for the admin.wsdl and the util.wsdl when TDV is locally installed with default values are:

http://localhost:9400/services/system/admin?wsdl

http://localhost:9400/services/system/util?wsdl

Naming Conflicts between User-Defined and Built-in Procedures

If a user-defined procedure has a name that is identical to the name of any TDV built-in procedure, the result is a naming conflict. In such cases, the path to the built-in system procedure always takes precedence.

For example, any procedure named print automatically maps to the path of the built-in print procedure (/lib/debug/print)—even if the full path to a user-defined procedure named print is specified.

Sample JMS Built-in Procedure

As an introduction to how built-in procedures can be used, here is an example that uses several built-in procedures from /lib/jms:

```
PROCEDURE jmsExampleProc()
BEGIN
 -- Create queue_connector and topic_connector beforehand using
Manager.
 -- Declare a row type variable to send a map message.
 -- The message has keys named using the attributes of the
 -- following ROW type variable.
 DECLARE mapmsg ROW( A INT, B VARCHAR );
 -- Declare a variable to send JMS properties.
 DECLARE complexProperties ROW ( uname VARCHAR, utime INT );
 SET mapmsg = ( 1, 'var' );

 -- Set a simple JMS property.
 CALL SetMessageProperty( 'userId', 'admin' );
 -- Send the map message.
 CALL SendMapMessage( 'topic_connector', 'my.topic', mapmsg );
 -- Clear all properties.
 CALL ClearMessageProperties();
```

```
 -- Set values for the ROW type variable, which is used to set JMS
properties.
 -- The names of such properties are created using the attributes
 -- of the ROW variable.
 -- The value of the property is specified by the following
assignment.
 SET complexProperties = ( 'admin', 1001 );
 CALL SetMessageProperties( complexProperties );
 -- Send several messages. Each message uses the properties in
effect.
 CALL SendTextMessage( 'queue_connector', 'my.queue', 'hello world'
);
 CALL SendTextMessage( 'queue_connector', 'my.queue', 'hello
world2' );
 -- Properties are cleared upon returning.
END
```

Procedures Reference

This section describes all TDV/Studio built-in procedures in alphabetical order, along with their resource tree location, inputs, outputs, and exceptions.

Note: Several procedures (two for lineage and five for Deployment Manager) are in a second location as published resources.

AddUsernameToken

Add a WS-Security UsernameToken to a SOAP envelope.

The UsernameToken is added to the SOAP header that is identified by the actor and mustUnderstand arguments. If the SOAP message does not contain a SOAP header with the specified actor and mustUnderstand values, the header is created.

The passwordType argument (DIGEST (default) or TEXT) determines how the password is encoded in the UsernameToken.

For details on WS-Security UsernameTokens, see:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

This procedure is discussed in the "Adding a Username Token Pipeline Step" section of the *TDV User Guide*.

Location

/lib/services/

Inputs

envelope: A SOAP envelope, which may include a WS Security header.

actor: Determines which WS Security header to process. May be NULL.

mustUnderstand: Indicate whether or not the receiver must understand this header. It may be NULL. If NULL, mustUnderstand defaults to TRUE.

username: The username.

password: The user's password.

passwordType: The password type. Must be TEXT or DIGEST.

Outputs

envelope: The SOAP envelope.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

CancelDataSourceReintrospect

Cancel an in-progress, non-blocking reintrospection process that was started using ReintrospectDataSource, page 313.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Input

reintrospectId: The reintrospection ID provided by the initial StartDataSourceReintrospect call.

Output

status:

- CANCELED if reintrospection was successfully canceled.
- SUCCESS or FAIL (as appropriate) if reintrospection already completed prior to this call. These three values can be found on the Constants tab of the /lib/util/System SQL definition set.

Exceptions

NotFoundException: If reinstrospectId does not exist. This can occur if the reintrospection was previously canceled using this procedure, or if the report has already been retrieved using GetDataSourceReintrospectReport, page 289.

CancelResourceStatistics

Asynchronously cancel statistics that are currently being gathering on a resource. RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics gathering.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or an illegal type is provided.

NotAllowedException: If the resource type does not support statistics. RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path.

ClearAllDataSourceCredentials

Clear all credential data previously stored using the setDataSourceCredentials JDBC method or the SetDataSourceCredentials, page 323 procedure. This procedure provides access to the facility described in "Multiple Credentials for JDBC Connection" in the *TDV User Guide*.

An example of this built-in procedure can be found in the "Example of Java JDBC Client Application Code" section of the *TDV Client Interfaces Guide*.

Location

/lib/util/

Inputs

N/A

Outputs

N/A

Exceptions

N/A

ClearAlternatePrincipal

Remove the effects of setting an alternate principal. (See also SetAlternatePrincipal, page 322.)

Location

/lib/util/

Inputs

N/A

Outputs

N/A

Exceptions

N/A

ClearMessageProperties

Clear all JMS headers and properties that were set using SetMessageProperties, page 326.

Locations

/lib/jms/

/lib/util/

Inputs

N/A

Outputs

N/A

Exceptions

N/A

ClearResourceCache

Clear the cache on a resource. Both procedures and SQL_TABLE resources support clearing of the resource cache.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or an illegal type is provided.

IllegalStateException: If the cache is disabled.

NotAllowedException: If the resource type does not support caching. Only SQL_TABLE resources support caching.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in the path.

ClearResourceStatistics

Clear the statistics on a resource. Statistics are recomputed at the next scheduled refresh.

RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or an illegal type is provided.

IllegalStateException: If statistics gathering is disabled.

NotAllowedException: If the resource type does not support statistics. RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path.

CopyResource

Copy the specified resource into a folder using a new name.

The copyMode parameter controls behavior in cases where a resource exists with the same name and type in the container specified by newPath.

The copyMode options are:

- ALTER_NAME_IF_EXISTS: If a resource of the same name and type as the source resource already exists in the target container, avoid conflicts by automatically generating a new name. Names are generated by appending a number to the end of the provided name.
- FAIL_IF_EXISTS: Fail if a resource of the same name and type of the source resource already exists in the target container. If this occurs, the resource is not copied.
- OVERWRITE_MERGE_IF_EXISTS: If a resource of the same name and type as the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, merge the contents of the source container with the corresponding resource in the target. All resources in the source container overwrite those in the target, but existing resources in the target that are not overwritten remain unaltered.
- OVERWRITE_REPLACE_IF_EXISTS: If a resource of the same name and type as the source resource already exists in the target container, overwrite the resource in the target container. If the source resource is a container, replace the container within the target container with the source container. This is equivalent to deleting the container in the target before copying the source.

Location

/lib/resource/

Inputs

path: A source path of the resource to be copied.

type: The type of the source resource to be copied.

newPath: The path of the target container to copy the resource into.

newName: The new name to call the copied resource.

copyMode: Valid values: ALTER_NAME_IF_EXISTS, FAIL_IF_EXISTS, OVERWRITE_MERGE_IF_EXISTS, or OVERWRITE_REPLACE_IF_EXISTS. These values can be found on the Constants tab of the /lib/resource/ResourceDefs SQL definition set.

Outputs

N/A

Exceptions

DuplicateNameException: If a resource in the target container exists with the same type as the source and the same name as newName, and the copy mode is FAIL_IF_EXISTS.

IllegalArgumentException: If any of the given paths or types are malformed, or if copyMode is not a legal value.

IllegalStateException: If the source resource is not allowed to be copied. Resources in /services/databases/system, /services/webservices/system, or within any physical data source may not be copied.

NotAllowedException: If the source resource is not allowed to exist within the target container. Resources cannot be copied into a physical data source. a LINK resource can only be copied into a RELATIONAL_DATA_SOURCE, SCHEMA, or PORT under */services*. Non-LINK resources cannot be copied into any location under */services*.

NotFoundException: If the source resource or any portion of the new path does not exist.

SecurityException: If the user does not have READ access on all items in the source path.

SecurityException: If the user does not have READ access on the items in the newPath other than the last item.

SecurityException: If the user does not have WRITE access to the last item in newPath.

SecurityException: If the user does not have WRITE access to a resource that is to be overwritten in one of the overwrite modes.

CreateElement

Create a child element in an XML document or element. This method creates an element in an XML document or element. The parentXPath expression selects the parent element, relative to root, of the element to create. The namespacePrefixes and namespaceURIs are used to resolve prefixes to namespaces in the parentXPath expression. Each item in namespacePrefixes must have a corresponding item in xpathNamespaces. The empty string is used to specify the default namespace.

Location

/lib/services/

Inputs

root: An XML document or element.

elementName: The fully qualified name of the element. May not be NULL. For example: {http://examples.com/}Example.

position: The position of the element, relative to its siblings. Use 0 to indicate the element should be created before any existing children. Use -1 to indicate that the element should be created after all existing children. The default value is 0.

parentXPath: An XPath expression that is evaluated against the root to identify the parent of the element that is to be created. May not be NULL.

namespacePrefixes: An array of namespace prefixes used in the parentXPath expression. May be NULL.

namespaceURIs: An array of namespace URIs used in the parentXPath expression. May be NULL.

Outputs

envelope: The modified XML document or element.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

IllegalArgumentException: If the parentXPath expression does not resolve to an element.

CreateResourceCacheKey

Create a cache key for a given resource. Used together with UpdateResourceCacheKeyStatus, page 334 and GetResourceCacheStatus, page 297 to support external cache loading.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

Outputs

cachekey: The new cache key for the resource.

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type does not support being cached.

IllegalStateException: If the resource is not configured for caching.

IllegalStateException: If the data source used by the resource for caching is not properly configured.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

DeleteElement

Delete one or more element nodes from an XML document or element.

The XPath expression selects the nodes to delete. The XPath is evaluated against the root node. All resulting nodes are deleted.

The namespacePrefixes and namespaceURIs are used to resolve prefixes to namespaces in the parentXPath expression. Each item in namespacePrefixes must have a corresponding item in xpathNamespaces. The empty string specifies the default namespace.

Location

/lib/services/

Inputs

root: An XML document or element.

xpath: An XPath expression to evaluate against the root node to select the nodes to delete.

namespacePrefixes: An array of namespace prefixes used in the XPath expression. May be NULL.

namespaceURIs: An array of namespace URIs used in the XPath expression. May be NULL.

Outputs

envelope: The XML document or element.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

EncryptElement

Encrypt an element in the specified SOAP envelope using a symmetric key that is encrypted by a certificate or public key.

The elementName argument determines which element in the message to encrypt. Typically this procedure is used to encrypt either the header or body of the SOAP message.

This method adds WS-Security artifacts to a header in the SOAPenvelope.

Artifacts are added to the SOAP header that is identified by the actor and mustUnderstand arguments. If the SOAP message does not contain a SOAP header with the specified actor and mustUnderstandvalues, the header is created.

The encryptionAlgorithm determines the method of encryption. The default value (AES_128) is sufficient for most purposes. Stronger encryption algorithms, such as AES_192 or AES_256, require an unrestricted Java Cryptography Extension (JCE) policy file to be installed in the server's JVM.

Location

/lib/services/

Inputs

envelope: A SOAP envelope. It may not be NULL.

actor: Determines which WS Security header to process. It may be NULL.

mustUnderstand: Indicates whether or not the receiver must understand this header. If NULL, mustUnderstand defaults to TRUE.

elementName: The name of the element in the envelope to encrypt. If NULL, elementName defaults to *{http://schemas.xmlsoap.org/soap/envelope/}Body*.

encryptionAlgorithm: The symmetric encryption algorithm used to encrypt the data. It may be NULL, TRIPLE_DES, AES_128, AES_192, AES_256. If NULL, encryptionAlgorithm defaults to AES_128.

certificateAlias: The alias of a certificate or public key in the key store to use to encrypt the symmetric key that is used to encrypt the element. It may not be NULL.

keyStore: A serialized Java key store. It may be NULL.

keyStoreType: The type of key store. It must be JKS or PKCS12. It may not be NULL.

keyStorePassword: The password of the key store and of all private keys within it. It may be NULL or empty if there is no password.

Output

envelope: The SOAP envelope containing the encrypted element and the generated WS Security artifacts in a SOAP header.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

SecurityException: If the element could not be encrypted.

ExecuteBasicTransform

Execute a basic transformation on the input XML value and retrieve metadata information.

Location

/lib/resource/

Syntax

```
executeBasicTransform(IN inputXmlValue XML, OUT outputs CURSOR (
ID INTEGER,
PARENT_ID INTEGER,
"DEPTH" INTEGER,
NAME VARCHAR(255),
"XPATH" VARCHAR(255),
"PATH" VARCHAR(255),
"POSITION" INTEGER,
"VALUE" VARCHAR(65536)
)
```

Input

inputXmlValue: The input XML value.

Output

columnDependencies: A cursor whose rows have the following columns:

- ID: The node ID.
- PARENT_ID: The parent node ID.
- DEPTH: Depth in the XML tree.
- NAME: The element node name.
- XPATH: The XPath for the node.
- PATH: The path to the node.
- POSITION: The position of the node.
- VALUE: The node value.

Exceptions

N/A

ExplainAttributes

Retrieve the list of attributes in a resource set.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

resourceSetDefinition: The definition of the resource set to analyze.

Outputs

attributeList: A cursor whose rows encode the attributes of the data sources included in the specified resource set:

- resourcePath: Path to the resource.
- resourceType Type of resource.
- attributeName: Name of attribute
- attributeType: Type of attribute.
- attributeValue: Value of the attribute, serialized as a JSON value.
- isEndpoint: True if the attribute value is a physical property of a data source; false if it is a logical property of a data source as a TDV metadata artifact.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

ExplainPrincipals

Retrieve the list of principals included in the specified resource set.

Location

/lib/users/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

resourceSetDefinition: The definition of the resource set to analyze.

Outputs

principalList: A cursor whose rows encode the principals included in the specified resource set:

- domain: The domain name.
- name: The resource type.
- isGroup: True if the member is a group.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

ExplainResources

Retrieve the list of resources included in the specified resource set.

The second through fifth inputs (highlighted in bold) support paged, random resource access. When it is employed by the Deployment Manager UI client, it alleviates the performance issues related to large resource sets.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

resourceSetDefinition: The definition of the resource set to analyze.

searchPath: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored. If set and effective, searchType must also be set and searchLevel must be non-zero. When searchPath is set, the matched resource is not included in the result.

searchType: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored. Accepts the set of resource type values accepted by the Web Services operations ("Admin API calls")—for example, CATALOG_CONTAINER, SCHEMA_CONTAINER. If set and effective, searchLevel must also be set.

searchSubtype: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored.

Accepts the set of resource subtype values accepted by the Web Services operations ("Admin API calls")—for example, CATALOG_CONTAINER, SCHEMA_CONTAINER.

searchLevel: 0 denotes the resources matching the search criteria. A negative value denotes unlimited depth. A positive value denotes the maximum depth, relative to the matched resources, at which descendent resources are to be included in the result.

Outputs

resourceList: A cursor whose rows encode the resources included in the specified resource set:

- resourcePath: The resource path.
- resourceType: The resource type.
- isNonRelocatableOnTarget: True if the resource cannot be relocated on the target; false if it can.
- owner: The resource owner.
- createTime: Time the resource was created.
- modifyTime: Time the resource was last modified.
- modifyUser: User who last modified the resource.
- resourceId: The resource id.
- parentId: The parent resource id.
- depthLevel: The depth level of the resource within the resource tree.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

GenerateEvent

Generate a custom event with the specified name and value. This event can be used to activate a trigger that has been configured to listen for this event name.

Location

/lib/util/

Syntax

```
generateEvent (IN eventName VARCHAR (255), IN value VARCHAR (4096))
```

Inputs

eventName: The name of the event.

value: The value for the event.Output: The cs_server_events.log file records events generated by this procedure.

Exceptions

N/A

Remarks

The input eventName is the name of the event.

The input value is the value for the event.

The cs_server_events.log file records events generated by this procedure.

Example

```
PROCEDURE CallsGenEv()
    BEGIN
        CALL GenerateEvent('runAReport', ' ');
    END
```

GetClaim

Returns the Claim value from the bearer token for the specific Claim provided in the argument. The built-in procedure is also discussed in the *TDV Administration Guide* chapter *OAuth Administration.*

Location

/lib/users

Input

The claim name that is carried in the bearer token sent by the Client Application. The bearer token is encoded token in JSON format with name-value pairs.

Output

Returns the Claims value for the Claim name thats passed as the argument.

GetColumnDependencies

Retrieve the column dependencies of the specified view. The analysis is done based on the view definition, regardless of whether or not the view is cached.

This built-in procedure is discussed in the section "View Column Dependencies and References" in the *TDV User Guide*.

Location

/lib/resource/ (procedure)
/services/databases/system/lineage/ (published resource)

Inputs

resourcePath: The path to the resource to analyze. The supported resource types are TDV SQL views in plain or published form.

columnFilter (optional): A comma-separated sequence of case-insensitive column names, indicating the columns whose dependencies should be analyzed. An empty string or NULL indicates all view columns.

ignoreCaches: TRUE if the analysis should ignore whether dependent resources are cached or not, otherwise FALSE. The default is FALSE.

recursively: TRUE if the analysis should be performed recursively down to base-level dependencies. FALSE if the analysis should be performed at a single dependency level.

Outputs

columnDependencies: A cursor whose rows encode column dependencies, having the following columns:

- columnName: The name of the resource column having the column dependency encoded in the row.
- dependencyDatasourcePath: The path to the data source containing the resource that owns the dependency. Empty if not applicable.
- dependencyDatasourceType: The type of the data source containing the resource that owns the dependency. Empty if not applicable. The set of data source types consists of all the data source adapter names accepted by TDV.
- dependencyResourcePath: The path to the resource owning the dependency. Empty if not applicable.
- dependencyResourceType: The type of the resource owning the dependency. Empty if not applicable. The set of table or procedure resource types accepted by TDV is as follows:

- Database table
- Delimited file
- Excel table
- TDV SQL view
- System table
- SAP RT table
- SAP RFC table
- SAP AQ query table
- SAP Infoset query table
- Siebel table
- Database stored procedure
- Packaged query
- Java procedure
- Web Service operation
- TDV SQL Script procedure
- XQuery procedure
- XSLT procedure
- Transform procedure
- Basic transform procedure
- Stream transform procedure
- XQuery transform procedure
- XSLT transform procedure
- dependencyIdentifier: The column name if the dependency is on a column; otherwise, a literal.
- dependencyKind: One of the following:
- column: A dependency on a column.
- literal: A dependency on a constant value.
- parameter: A dependency on a dynamic value provided at runtime.
- derivationKind: One of the following:
- direct: The value of the dependency is preserved by the dependent column.

- indirect: The value of the dependency is transformed by the dependent column.
- cardinalityInfo: When applicable, one of the following:
- aggregate: An aggregate function is involved in the derivation of the dependent column.
- analytic: An analytic function is involved in the derivation of the dependent column.
- derivations: When the dependent column is not a direct projection of the dependency, this field denotes how the dependent column is derived.
- position: The line number and column number of the dependency formatted as [line number],[column number].

Exceptions

IllegalArgumentException: If the viewPath is malformed, the specified resource cannot be found, or the specified resource is impacted.

GetColumnProfiles

Retrieve statistical and auxiliary data type information about the specified set of published table columns. Cardinality statistics must be enabled and gathered on the data source to enable gathering of profiling information. Please note that the table or view may have underlying procedures and column profiles can be got for such views if cardinality statistics are enabled. This procedure is present in "/services/databases/system/profile/GetColumnProfiles".

When a published resource has an underlying view, caching and cardinality statistics must be enabled and gathered before using the profile APIs. Enabling cardinality statistics at the data source level may not be sufficient in gathering profile information.

Inputs

database: The name of the published database to which the column belongs. Special character '*' can be used as a wildcard to denote any database. This parameter is required.

catalog: The name of the published catalog to which the column belongs. Special character '*' can be used as a wildcard to denote any/no catalog. This parameter is optional (i.e. it may be set to NULL to denote no catalog).

schema: The name of the published schema to which the column belongs. Special character '*' can be used as a wildcard to denote any/no schema. This parameter is optional (i.e. it may be set to NULL to denote no schema).

table: The name of the published table to which the column belongs. Special character '*' can be used as a wildcard to denote any table. This parameter is required.

columnFilter: A comma-separated list of column names. This parameter is optional. If set to NULL all table columns will be profiled.

Outputs

A cursor whose rows encode table column profiles, having the following columns:

database: The name of the published database to which the column belongs.

catalog: The name of the published catalog to which the column belongs. May be NULL.

schema: The name of the published schema to which the column belongs. May be NULL.

table: The name of the published table to which the column belongs.

column: The name of the published column.

minValue: The textual representation of the column's minimum value. NULL if unknown or not applicable.

maxValue: The textual representation of the column's maximum value. NULL if unknown or not applicable.

distinctCount: The number of distinct column values. NULL if unknown or not applicable.

nullCount: The number of NULL column values. NULL if unknown or not applicable.

partitionable: TRUE if partitions can be defined against the specified column; otherwise, FALSE.

message: A message specifying any detected reason preventing the analysis of the specified table column.

Exceptions:

An IllegalArgumentException is thrown when the specified resources cannot be found. An exception is also thrown when the user does not have the authorization to read the specified resource(s).

GetColumnReferences

Retrieve the column references of the specified view, table or procedure. The analysis is done whether or not the specified view or table is cached.

This built-in procedure is discussed in the section "View Column Dependencies and References" in the *TDV User Guide*.

Location

/lib/resource (procedure)
/services/databases/system/lineage/ (published resource)

Inputs

resourcePath: The path to the resource to analyze. This parameter is required.

columnFilter (optional): A comma-separated list of case-insensitive column names, indicating the columns whose references should be analyzed. Empty string or NULL indicates all columns in the view or table.

Outputs

columnReferences: A cursor whose rows encode column references, having the following columns:

- columnName: The name of the resource column having the column reference encoded in the row.
- referentResourcePath: The path to the resource containing the column reference.
- referentResourceType: The type of the resource containing the column reference. The supported resource types are TDV SQL views in plain or published form.
- referenceContext: One of the following:
- In WITH clause: A reference within a WITH clause.
- In SELECT clause as output: A reference that is projected by a SELECT clause.
- In SELECT clause as input: A reference that is used, but not projected, by a SELECT clause.

- In FROM clause: A reference within a FROM clause.
- In WHERE clause: A reference within a WHERE clause.
- In TIMESERIES clause: A reference within a TIMESERIES clause.
- In GROUP BY clause: A reference within a GROUP BY clause.
- In HAVING clause: A reference within a HAVING clause.
- In ORDER BY clause: A reference within a ORDER BY clause.
- referentColumnName: The name of the referent column. Populated only if referenceContext is in SELECT clause as output; otherwise, empty.
- derivationKind: Populated only if referenceContext is in SELECT clause as output; otherwise, empty. One of the following:
- direct: The value of the dependency is preserved by the dependent column
- indirect: The value of the dependency is transformed by the dependent column.
- cardinalityInfo: Populated only if referenceContext is in SELECT clause as output; otherwise, empty. When applicable, one of the following:
- aggregate: An aggregate function is involved in the derivation of the dependent columns.
- analytic: An analytic function is involved in the derivation of the dependent column.
- reference: A textual representation of the column reference.
- position: The line number and column number of the reference formatted as [line number],[column number]

Exceptions

IllegalArgumentException: If the viewPath is malformed, the specified resource cannot be found, or the specified resource is impacted.

GetDataSourceReintrospectReport

Get the reintrospect report for a reintrospection, if available. A reintrospection is started using ReintrospectDataSource, page 313. Retrieving the report using this call invalidates reintrospectID.

Location

/lib/resource/

Inputs

reintrospectId: The reintrospection ID provided by the ReintrospectDataSource procedure.

isBlocking: If TRUE, this call does not return until reintrospection has completed or has been canceled by another thread. If FALSE, this call returns immediately, regardless of completion.

Outputs

status: SUCCESS or FAIL (as appropriate) if the reintrospection completed during or prior to this call. INCOMPLETE if isBlocking is FALSE and the reintrospection is still in progress. CANCELED if the reintrospection was canceled by a separate call during this call. These values are available on the Constants tab of the */lib/util/System* SQL definition set.

report: The reintrospection report.

Exceptions

NotFoundException: If the reinstrospectId does not exist. It does not exist if it was canceled or if a previous GetDataSourceReintrospectReport already retrieved the report.

GetEnvironment

Get the value of an environment variable from the last operation.

Note: These environment variables are local to individual procedure executions; they are not global.

All built-in variable names are available on the Constants tab of the /lib/util/System SQL definition set. The variable names are:

- System.CASE_SENSITIVE_IN_COMPARISONS: TRUE or FALSE. Reflects the case sensitivity being used in string comparisons for SQL and SQL Script operations in this scope.
- System.IGNORE_TRAILING_SPACES_IN_COMPARISONS: TRUE or FALSE. Reflects whether or not trailing spaces are ignored in string comparisons for SQL and SQL script operations in this scope.
- System.NUM_ROWS_AFFECTED: A numeric value.
- System.TRIGGER_EVENT_NAME: The trigger name if the current request is the result of a trigger. NULL otherwise.

- System.TRIGGER_EVENT_TYPE: The trigger type if the current request is the result of a trigger. NULL otherwise.
- System.TRIGGER_EVENT_VALUE: The trigger value if the current request is the result of a trigger. NULL otherwise.
- System.TRIGGER_PATH: Path if the current request is result of a trigger. NULL otherwise.
- System.CACHED_RESOURCE_PATH: The path to the resource whose cache is being refreshed, if the current request is the result of a cache refresh callback. NULL otherwise.
- System.CACHED_RESOURCE_TYPE: The type of the resource whose cache is being refreshed, if the current request is the result of a cache refresh callback. NULL otherwise.
- System.CACHED_RESOURCE_PARAM_KEY: The parameter key of the resource whose cache is being refreshed, if the current request is the result of a cache refresh callback. NULL otherwise.
- System.CACHE_DATASOURCE_PATH: The path of the cache data source, if the current request is the result of a cache refresh callback. NULL otherwise.
- System.CACHED_RESOURCE_CACHE_KEY: The cache key used by cache refresh, if the current request is the result of a cache refresh callback. NULL otherwise.
- System.CACHED_RESOURCE_BUCKET_PATH: The path of the cache table used by cache refresh, if (1) the current request is the result of a cache refresh callback, and (2) the cache refresh mode is one table per snapshot (OTPS). NULL otherwise.
- System.CACHED_RESOURCE_REFRESH_OUTCOME: The outcome of the cache refresh, if the current request is the result of a cache refresh callback. NULL otherwise. TRUE for success, FALSE for failure and NULL for unknown.
- System.CACHED_RESOURCE_ERROR_MESSAGE: The error message generated by the cache refresh, if the current request is the result of a cache refresh callback and the cache refresh failed. NULL otherwise.

For backward compatibility, the following are also accessible without the `System.` prefix:

- CASE_SENSITIVE_IN_COMPARISONS
- IGNORE_TRAILING_SPACES_IN_COMPARISONS
- NUM_ROWS_AFFECTED
- TRIGGER_EVENT_NAME

- TRIGGER_EVENT_TYPE
- TRIGGER_EVENT_VALUE
- TRIGGER_PATH

Location

/lib/util/

Syntax

```
getEnvironment (IN variableName VARCHAR (40),
OUT propValue VARCHAR (2048))
```

Inputs

variableName: The name of a variable. Variable names are not case-sensitive. For example, both sample and SAMPLE are the same variable.

Outputs

propValue: The value stored in the variable, or NULL if no value is stored.

Example

```
PROCEDURE proc4()
    BEGIN
        PATH /shared/sources/scripts;
        DECLARE x VARCHAR(4096);

        CALL insertProc(); -- This procedure is in the PATH
        CALL getEnvironment('NUM_ROWS_AFFECTED', x);
        CALL log(x);
    END
```

GetPartitionClauses

Retrieve the partition SQL clauses (predicates) to be used in order to define partition queries against the specified published table using the specified column. Cardinality statistics must be enabled and gathered on the data source to enable gathering of profiling information. Please note that the table or view may have underlying procedures and partition clauses can be got for such views if cardinality statistics are enabled. This procedure is present in "/services/databases/system/profile/GetPartitionClauses".

When a published resource has an underlying view, caching and cardinality statistics must be enabled and gathered before using the profile APIs. Enabling cardinality statistics at the data source level may not be sufficient in gathering profile information.

Inputs

database: The name of the published database to which the partition column belongs. This parameter is required.

catalog: The name of the published catalog to which the partition column belongs. This parameter is optional (i.e. it may be set to NULL to denote no catalog).

schema: The name of the published schema to which the partition column belongs. This parameter is optional (i.e. it may be set to NULL to denote no schema).

table: The name of the published table to which the partition column belongs. This parameter is required.

column: The name of the partition column. This parameter is required.

partitionCount: A numeric value indicating the desired number of partitions. This parameter is optional.

nullsFirst: A boolean value indicating whether NULL values should be returned by the first partition clause or the last.

Outputs

A cursor whose rows encode partition SQL clauses (predicates), having the following columns:

partitionClause: The SQL-encoded partition clause to be used in order to retrieve a partition.

Exceptions:

IllegalArgumentException: If the specified resources cannot be found or the values of the specified column cannot be partitioned or access to the specified set of column values is not allowed.

GetPrincipalSet

Retrieve the list of principals included in the specified principal set.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

principalMapping: The principal mapping.

operation:

- Add: The CAR file contains only the clone domain, group or user information.
- Remove: The CAR file contains the deleted domain, group or user information.

Outputs

principalSetArchive: The archive containing the principal set.

Exceptions

IllegalArgumentException: If the specified principal set is invalid.

GetTableProfiles

Retrieve statistical and data source lineage information about the specified set of published tables or views. Cardinality statistics must be enabled and gathered on the data source to enable gathering of profiling information. This procedure is present in "/services/databases/system/profile/GetTableProfiles".

When a published resource has an underlying view, caching and cardinality statistics must be enabled and gathered before using the profile APIs. Enabling cardinality statistics at the data source level may not be sufficient in gathering profile information.

Inputs

database: The name of the published database to which the table belongs. Special character '*' can be used as a wildcard to denote any database.%This parameter is required. Specifying null will throw an error.

catalog: The name of the published catalog to which the table belongs. Special character '*' can be used as a wildcard to denote any/no catalog.%This parameter is optional (i.e. it may be set to NULL to denote no catalog).

schema: The name of the published schema to which the table belongs. Special character '*' can be used as a wildcard to denote any/no schema.%This parameter is optional (i.e. it may be set to NULL to denote no schema).

table: The name of the published table. Special character '*' can be used as a wildcard to denote any table.%This parameter is required. Specifying null will throw an error.

Outputs

A cursor whose rows encode table profiles, having the following columns:

database: The name of the published database to which the table belongs.

catalog: The name of the published catalog to which the table belongs. May be NULL.

schema: The name of the published schema to which the table belongs. May be NULL.

table: The name of the published table.

rowCount: The number of rows. NULL if unknown.

partitionColumn: The name of the column to be used to define partition queries against this table. NULL if unknown.

dataSourceType: The name and version of the data source this published table retrieves data from. Applicable only to published tables that serve SELECT * queries against them using pass-through requests that are fully executed within the underlying data source; otherwise, this value is set to NULL.

message: A message specifying any detected reason preventing the analysis of the specified table.

Exceptions:

An IllegalArgumentException is thrown when the specified resources cannot be found. An exception is also thrown when the user does not have the authorization to read the specified resource(s).

GetProperty

Get the value of a system property. The properties are global and shared across scripts.

All property names are available on the Constants tab of the /lib/util/System SQL definition set.

Any one of the following property names can be submitted to get its value:

- CLUSTER_ID: The server's cluster ID.
- CURRENT_USER_DOMAIN: The current user's domain.
- CURRENT_USER_ID: A current user's ID as a numeric value.
- CURRENT_USER_NAME: The current user's name.
- SERVER_HOSTNAME: The server's host name.
- SERVER_ID: The server's ID.
- SERVER_JDBC_PORT: The server's JDBC port.
- SERVER_VERSION: The server's software version string.
- SERVER_VERSION_NUMBER: The server's software version number only.
- SERVER_WEB_PORT: The server's HTTP port.
- SESSION_ID: The session ID.
- TRANSACTION_ID: The transaction ID.

Location

/lib/util/

Syntax

```
getProperty (
IN propertyName VARCHAR (255),
OUT propertyValue VARCHAR (4096))
```

Input

propertyName: The name of the property. (See list above.)

Output

propertyValue: The text to write to the debug console.

Exception

IllegalArgumentException: If an unsupported property name is requested.

Example

```
PROCEDURE proc5()
    BEGIN
        DECLARE x VARCHAR(4096);
        CALL getProperty('CURRENT_USER_ID', x);
        CALL log(x);
        CALL getProperty('CURRENT_USER_NAME', x);
        CALL log(x);
        CALL getProperty('CURRENT_USER_DOMAIN', x);
        CALL log(x);
    END
```

GetResourceCacheStatus

Used together with CreateResourceCacheKey, page 276 and UpdateResourceCacheKeyStatus, page 334 to support external cache loading. Returns cache status information for a given cache key.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

cacheKey: the key returned by CreateResourceCacheKey.

Outputs

status: Internal cache status.

bucket: TDV path to the chosen target table for resources in OTPS (one table per snapshot) mode, NULL for other resources.

message: Additional information.

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type does not support being cached.

IllegalStateException: If the resource is not configured for caching.

IllegalStateException: If the data source used by the resource for caching is not properly configured.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

GetResourceSet

Retrieve the list of resources included in a specified resource set.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

resourceSetDefinition: Definition of the resource to analyze.

resourceSetIndex: The index of the resources included in the specified resource set during the last deployment session.

principalMapping: The principal mapping containing the principals to be selected in the source site and mapped to target site principals.

preview (optional; default is false): True if this procedure should return only the changes (resourceSetIndexDelta) since the last deployment; otherwise false. If resourceSetIndex is NULL, all resources within the specified resource set are listed as CREATED.

Outputs

createdResourceArchive: The archive containing the resources created since the last deployment, if one exists, otherwise all resources, that are included in the specified resource set. NULL if preview input argument is set to true.

updatedResourceArchive: The archive containing the resources updated since the last deployment, if one exists, that are included in the specified resource set. NULL if preview input argument is set to true.

resourceSetIndexDelta: The set of resources in the specified resource set that were CREATED, UPDATED, RELOCATED or DELETED since the last deployment session, based on the resourceSetIndex provided.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

HasClaim

Returns a boolean value to indicate if a specific Claim provided exists or not in the bearer token. The built-in procedure is also discussed in the *TDV Administration Guide* chapter *OAuth Administration.*

Location

/lib/users

Input

The claim name that is carried on the bearer token sent by the Client Application.

Output

Returns TRUE if the claim name exists in the token and FALSE if it does not.

ListAttributes

Retrieve the resources and attributes for a specified resource set on a given site.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

siteName: Location of the resource.

kind: The content type—RESOURCE_SET_REFERENCE or RESOURCE_SET_DEFINITION.

resourceSet: Resource set name or set definition. If this is NULL, the procedure lists both names and definitions.

Outputs

attributeList: A cursor whose rows encode the attributes of the data sources included in the specified resource set:

- resourcePath: Path to the resource.
- resourceType Type of resource.
- attributeName: Name of attribute
- attributeType: Type of attribute.
- attributeValue: Value of the attribute, serialized as a JSON value.
- isEndpoint: True if the attribute value is a physical property of a data source; false if it is a logical property of a data source as a TDV metadata artifact.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

ListPrincipals

Retrieve the list of principals included in the specified resource set.

Location

/lib/users/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

siteName: The site name.

kind: The content type—RESOURCE_SET_REFERENCE or RESOURCE_SET_DEFINITION.

resourceSet: Resource set name or set definition. If this is NULL, the procedure lists both names and definitions.

Outputs

principalList: A cursor whose rows encode the principals included in the specified resource set:

- domain: The domain name.
- name: The resource type.
- isGroup: True if the member is a group.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

ListResources

Retrieve the list of resources included in the specified resource set.

The fourth through seventh inputs (highlighted in bold) support paged, random resource access. When it is employed by the Deployment Manager UI client, it alleviates the performance issues related to large resource sets.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

siteName: The site name.

kind: The content type—RESOURCE_SET_REFERENCE or RESOURCE_SET_DEFINITION.

resourceSet: Resource set name or set definition. If this is NULL, the procedure lists all resources.

searchPath: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored. If set and effective, searchType must also be set and searchLevel must be non-zero. When searchPath is set, the matched resource is not included in the result.

searchType: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored. Accepts the set of resource type values accepted by the Web Services operations ("Admin API calls")—for example, CATALOG_CONTAINER, SCHEMA_CONTAINER. If set and effective, searchLevel must also be set.

searchSubtype: Applicable only when resourceSetDefinition contains a tree root which is set to "/"; otherwise ignored. Accepts the set of resource subtype values accepted by the Web Services operations ("Admin API calls")—for example, CATALOG_CONTAINER, SCHEMA_CONTAINER.

searchLevel: 0 denotes the resources matching the search criteria. A negative value denotes unlimited depth. A positive value denotes the maximum depth, relative to the matched resources, at which descendent resources are to be included in the result.

Outputs

resourceList: A cursor whose rows encode the resources included in the specified resource set:

- resourcePath: The resource path.
- resourceType: The resource type.
- isNonRelocatableOnTarget: True if the resource does not belong to the core resource set and is a (direct or indirect) dependency of a resource in the core resource set.
- owner: The resource owner.
- createTime: Time the resource was created.
- modifyTime: Time the resource was last modified.
- modifyUser: The user who last modified the resource.
- resourceId: The resource id.
- parentId: The parent resource id.
- depthLevel: The depth level of the resource within the resource tree.

Exceptions

IllegalArgumentException: If the specified resource set definition is invalid.

LoadResourceCacheStatus

Load the status for a resource cache. Also check for any data that is not accessible in the cache, and clear it.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type does not support being cached.

IllegalStateException: If the resource is not configured for caching.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

Log

Write the text you provide to the log file, with severity level INFO.

Location

/lib/debug/

Syntax

```
log (IN textToLog VARCHAR (4096))
```

Input

textToLog: The text to write to the log.

Outputs

N/A

Exceptions

N/A

Example

```
PROCEDURE proc1()
    BEGIN
        CALL Log('Hello');
        CALL Log('Hello World');
```

```
    END
```

LogError

Write the text you provide to the log file, with severity level ERROR.

Location

/lib/debug/

Syntax

```
logError (IN textToLog VARCHAR (4096))
```

Input

textToLog: The text to write to the log.

Outputs

N/A

Exceptions

N/A

Example

```
PROCEDURE proc2()
    BEGIN
        CALL logError('There is an error.');
    END
```

LogMessageToFile

Write the contents of a message to a file at a specified path.

Location

/lib/services/

Inputs

element: An XML document or element.

filePath: The path of the file to write log messages to. The path is relative to the log directory of the server.

fileMode: OVERWRITE or APPEND. OVERWRITE indicates that the message should overwrite the current contents of the file. APPEND indicates that the message should be appended to the end of the file.

header: If present, this value is written to the file immediately before the message contents. It may be NULL.

footer: If present, this value is written to the file immediately after the message contents. It may be NULL.

prettyPrint: If true, the message is indented to make it easier to read.

Output

element: The modified XML document or element.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

ServerException: If an error occurs while writing to the file.

MoveResource

Move the specified resource into a folder using a new name.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource to move.

type: The type of the source resource to move.

newPath: The path of the target container to move the resource into.

newName: The new name to call the moved resource.

overwrite: If a resource exists in the target container with the same name and type as the target resource, and overwrite is TRUE, the resource within the target container is overwritten. If overwrite is FALSE, DuplicateNameException is generated and the resource is not moved.

Outputs

N/A

Exceptions

DuplicateNameException: If a resource in the target container exists with the same name and type as the source, and overwrite is FALSE.

IllegalArgumentException: If any of the given paths or types are malformed.

IllegalStateException: If the source resource is not allowed to be moved. Resources in */services/databases/system*, */services/webservices/system*, or within any physical data source may not be moved.

NotAllowedException: If the source resource is not allowed to exist within the target container. Resources cannot be moved into a physical data source. A LINK resource can only be moved into a RELATIONAL_DATA_SOURCE, SCHEMA, or PORT under */services*. Non-LINK resources cannot be moved into any location under */services*.

NotFoundException: If the source resource or any portion of the path to the target container does not exist.

SecurityException: If the user does not have READ access on all items in the source path.

SecurityException: If the user does not have READ access on the items in the newPath other than the last item.

SecurityException: If the user does not have WRITE access to the last item in newPath.

SecurityException: If the user does not have WRITE access to a resource that is to be overwritten.

Pause

Specify a sleep time, in milliseconds, for script execution. A value of less than zero is treated as zero.

Location

/lib/util/

Syntax

```
pause (IN timeInMilliSeconds INTEGER)
```

Input

timeInMilliSeconds: The number of milliseconds to pause.

Outputs

N/A

Exceptions

N/A

Example

```
PROCEDURE proc6()
    BEGIN
        CALL log('pausing for 3 secs');
        CALL pause(3000);
        CALL log('pause completed');
    END
```

PreviewResourceSet

Retrieve the list of resource changes since the last deployment of the specified resource set, by the specified deployment plan.

Location

/lib/resource/ (procedure)
/services/databases/system/deployment/ (published resource)

Inputs

resourceSet: A resource set name.

deploymentPlan: The path to a deployment plan using the specified resource set.

Output

resourceSetIndexDelta: The resources in the specified resource set that were CREATED, UPDATED, RELOCATED or DELETED since the last deployment session of the specified deployment plan.

Exceptions

IllegalArgumentException: If the specified resource set or deployment plan is invalid.

Print

Write debug messages to the console when running from Studio.

These print messages are available for the specific script being run, and are not carried across scripts. The print messages are displayed in Studio, as shown in the procedure that follows the syntax and example. If you turn on tracing, the message also appears in the log file.

Location

/lib/debug/

Syntax

```
print (IN textToPrint VARCHAR (4096))
```

Input

textToPrint: The text to write to the debug console.

Outputs

N/A

Exceptions

N/A

Example

```
PROCEDURE proc3()
    BEGIN
        CALL print('Test printing built-in.');
    END
```

To verify a print message that you have defined

1. Right-click the examples node in the resource tree and select New SQL Script.
2. Type a name for the script and click OK.
3. On the SQL Script panel, type the "CALL print..." line shown in the example above.
4. Click the Execute button.
5. Verify that the text you specified appears on the Console panel.

ProcessSecurityHeader

Process a WS Security SOAP header in a SOAP envelope. If the envelope contains a WS Security header with the specified actor, it is processed. All security elements in the header are evaluated. If any header security elements indicate that the envelope contains signed elements, the signatures of those elements is verified. If any header security elements indicate that the envelope contains encrypted elements, those elements are decrypted.

Location

/lib/services/

Inputs

envelope: A SOAP envelope, which may include a WS Security header.

actor: Which WS Security header to process. It may be NULL.

keyStore: A serialized Java key store containing certificates used to verify signatures and decrypt elements. It may be NULL.

keyStoreType: The type of key store. It must be JKS or PKCS12. May be NULL only if keyStore is NULL.

keyStorePassword: The password of the key store and of all private keys within it. May be NULL.

Output

envelope: The SOAP envelope. Some elements may have been decrypted.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

SecurityException: If the security elements in the message are invalid or unsupported.

SecurityException: If a signature could not be verified.

SecurityException: If an encrypted element could not be decrypted.

RefreshResourceCache

Refresh the cache for a table, procedure, or policy. Refreshes preexisting procedure cache variants, or only the NULL variant (when acceptable) if no others are present. This procedure launches an asynchronous process in the server which runs in its own transaction.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Syntax

```
RefreshResourceCache "path" "type" [pollingInterval]
```

Inputs

path: The path to the resource, or the cache policy name.

type: The type of the resource (TABLE, PROCEDURE, or POLICY).

Outputs

- N/A

Example

```
/lib/resource/RefreshResourceCache
"/shared/examples/ds_orders/cache" "TABLE"
```

Exceptions

Exception: If any problems with connecting to or retrieving data from the data source when refreshing.

IllegalArgumentException: If the path is malformed or an illegal type is provided.

IllegalStateException: If the cache is disabled.

NotAllowedException: If the resource type does not support caching. Only SQL_TABLE resources support caching.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

RefreshResourceCacheSynchronously

Refresh the cache on a resource synchronously.

Location

/lib/resource/

Syntax

```
RefreshResourceCacheSynchronously "path" "type"
```

Inputs

path: The path to the resource, or the cache policy name, enclosed in double-quotes.

type: The type of the resource (TABLE or POLICY), enclosed in double-quotes.

pollingInterval: An optional parameter denoting how frequently (timed in milliseconds) to poll for the cache refresh outcome. The default value is 1 second.

Outputs

N/A

Exceptions

RuntimeException: If the cache refresh fails.

IllegalArgumentException: If the path is malformed or an illegal type is provided.

IllegalStateException: If the cache is disabled.

NotAllowedException: If the resource type does not support caching. Only SQL_TABLE resources support caching.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

RefreshResourceStatistics

Refresh the statistics on a resource for use by the cost-based optimizer.

RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

isBlocking: Indicates whether the call blocks until completion.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or an illegal type is provided.

IllegalStateException: If the statistics gathering is disabled.

NotAllowedException: If the resource type does not support statistics. RELATIONAL_DATA_SOURCE, FILE_DATA_SOURCE, DATABASE_TABLE and DELIMITED_FILE_TABLE are the only resources that support statistics.

NotFoundException: If the resource or any portion of the path to the resource does not exist.

SecurityException: If the user does not have READ access on all items in path.

ReintrospectDataSource

Perform a reintrospection of the given data source. This is a non-blocking call that returns before reintrospection is complete. To block until the reintrospection is complete and also get the report, use GetDataSourceReintrospectReport, page 289. To cancel reintrospection, use CancelDataSourceReintrospect, page 268. If reintrospection is not complete before committing the transaction, the commit blocks until reintrospection is complete.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Input

path: The path to the data source being reintrospected.

Output

reintrospectId: The reintrospection ID. This value be used in calls to CancelDataSourceReintrospect or GetDataSourceReintrospectReport.

Exceptions

IllegalArgumentException: If the path is malformed.

NotFoundException: If a data source resource cannot be found at the given path.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

- SecurityException: If the user does not have WRITE access to the last item in path.

RenameResource

Rename a resource.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: A source path of the resource to rename.

type: The type of the source resource to rename.

newName: The new name of the resource.

Outputs

N/A

Exceptions

DuplicateNameException: If a resource already exists with the new name and that is the same type as the resource being renamed.

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource is not allowed to be renamed. Resources within a physical data source, user home folders, the */shared* folder, */services/databases*, and */services/webservices* cannot be renamed.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

ResourceExists

Check to see if a resource exists.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the resource to check.

type: The type of the resource to check.

version: Reserved for future use. (Only accepts NULL.)

Output

exists: TRUE if the resource exists, otherwise FALSE.

Exceptions

IllegalArgumentException: If the path is malformed, if an illegal type is provided, or if version is not NULL.

SecurityException: If the user does not have READ access on all items in path other than the last one.

Search

This is a packaged query used to perform a free text search in the TDV metadata. The search is performed across TDV resources such as datasources, tables, procedures, folders, catalogs, schemas and flows. Details that are fetched as search results include details about the resource - resource id, name, type, the field type (name/annotation/column/parameter/definition), contents of the matching field types, the parent name, id and path and the field type rank. Web service operations are also searched as part of the procedure.

Location

/lib/resource/

Inputs

Name	Datatype	Null	Description
query	LONGVARCHAR	Yes	When the specified query is null or empty, no results will be returned. Search text can be single or multi-word. When multiple words are specified, results contain records matching any/all the words.
fieldTypeFilter	LONGVARCHAR	Yes	When specified fieldTypeFilter is null or empty, all field types are matched for. Field type filter can be a single type filter or multiple comma-separated allowable values. The allowed values for this are "name", "annotation", "column", "parameter" and "definition". This input allows for search to be restricted to the specified comma-separated list of field types. Field type names are not case-sensitive.
resourceTypeFilter	LONGVARCHAR	Yes	When specified resourceTypeFilter is null or empty, all resource types are matched for. Resource type filter can be a single filter or multiple comma-separated allowable values. The allowed values for this are "datasource", "table", "procedure" , "folder", "catalog", "schema" and "flow". This input allows for search to be restricted to the specified comma-separated list of resource types. Resource type names are not case-sensitive.
annotateResult	BIT	Yes	Default value is 0. Allowed values are 0 and 1. When set to 1, results are annotated with the specified or default markers in "startMarker" and "stopMarker".
startMarker	VARCHAR	Yes	Used for annotating the start of the result. The default value is "".
stopMarker	VARCHAR	Yes	Used for annotating the end of the result. The default value is "".

Name	Datatype	Null	Description
reportDataFlows	BIT	Yes	Default value is 0. Allowed values are 0 and 1. When set to 1, resource_type in results will be “Flow” if the specific folder/view/sql script were created from WebUI as a flow. When null or 0, the matching folder/view/sql script though created as flows from WebUI will show up as their original resource types respectively.
nameRankWeight	INTEGER	Yes	This parameter is used for designating rank weight to "name" field type. When unspecified, the rank weight for the "name" field type is defaulted to 10.
annotationRankWeight	INTEGER	Yes	This parameter is used for designating rank weight to "column" and "parameter" field types. When unspecified, the rank weight for the "column" and "parameter" field types is defaulted to 5.
definitionRankWeight	INTEGER	Yes	This parameter is used for designating rank weight to "definition" field type. When unspecified, the rank weight for the "definition" field type is defaulted to 2.

Outputs

The **result** output is a cursor. The rows encode resources matching the search query under the specified conditions, having the following columns

resourceId : The id of the resource.

resourceName : The name of the resource.

resourceType : The resource type.

matchingFieldTypes : The types of the resource fields, delimited by the character “\”, matching the search term. Example : name\column\definition

matchingFieldContents : The contents of the matching fields, within single quotes and delimited by the character “\”, listed in the same order as the value of matchingFieldTypes.

parentDatasource : The name of the parent data source of the matching resource. Applicable only to resources that are contained within a data source.

parentDatasourceId : The id of the parent data source of the matching resource. Applicable only to resources that are contained within a data source.

parentPath : The parent path of the matching resource.

rank : The search rank of the result. This is the sum of all field types ranks for the matching resource.

SendEMail

Send an email message with the specified headers and content. It supports only NULL for the from address.

Location

/lib/util/

Syntax

```
SendEmail (IN from VARCHAR (4096),
IN replyTo VARCHAR (4096),
IN to VARCHAR (4096),
IN cc VARCHAR (4096),
IN bcc VARCHAR (4096),
IN subject VARCHAR (4096),
IN contentType VARCHAR (4096),
IN content VARCHAR (4096))
```

Inputs

from: The address the message is from. NULL causes use of the server's configured from-address. Only NULL is supported.

replyTo: The address to place in the replyTo field of the message.

to: A comma-separated list of email addresses.

cc: A comma-separated list of email addresses.

bcc: A comma-separated list of email addresses.

subject: The message subject.

contentType: TEXT_PLAIN or TEXT_HTML.

content: The message body.

Outputs

N/A

Exceptions

IllegalArgumentException: If from is not NULL.

IllegalArgumentException: If any of the address lines are malformed.

IllegalArgumentException: If there is not at least one to-address.

IllegalArgumentException: If there is more than one address in replyTo.

IllegalArgumentException: If contentType is not TEXT_PLAIN or TEXT_HTML.

IllegalStateException: If the server's from-address is not configured.

Example

```
PROCEDURE proc_SendEMail()
    BEGIN
        PATH /shared/sources/proceduresForDoc;
        CALL proc_GetProperty();
        CALL SendEMail(NULL, NULL, 'joe@smith.com',
                       NULL, NULL, 'hi', 'TEXT_PLAIN', NULL);
    END
```

SendMapMessage

Send a JMS map message based on a ROW type variable.

Locations

/lib/jms/

/lib/util/

Inputs

connectorName: Name of the JMS connector.

destinationName: JNDI name of the JMS queue or topic.

row: Message body keyed using the fields in the ROW type variable.

Outputs

N/A

Exceptions

N/A

SendResultsInEMail

Send an email message with the specified headers and content, and with the results of the given view or procedure as attachments.

The option keywords are used to control behavior. If options is NULL, the default is SUMMARY, CSV_ATTACH. The options are:

- CSV_ATTACH: Causes cursor result sets to be attached as CSV files.
- SEND_ERROR: Causes any errors from the executed procedure or table to be sent instead of being reported as an exception.
- SKIP_IF_NO_RESULTS: Sends no email if there are zero results.
- SUMMARY: Appends a summary of the results to the message content.

Location

/lib/resource/

Inputs

from: The address the message is from. NULL causes use of the server's configured from-address. Only NULL is supported.

replyTo: The address to place in the replyTo field of the message.

to: A comma-separated list of email addresses.

cc: A comma-separated list of email addresses.

bcc: A comma-separated list of email addresses.

subject: The message subject.

contentType: This can be TEXT_PLAIN or TEXT_HTML.

content: The message body.

path: The path to a view or procedure to execute.

type: Either TABLE or PROCEDURE, as appropriate.

parameters: If executing a PROCEDURE with any parameters, this is a comma-separated list of parameter values. Otherwise this should be NULL.

options: A comma-separated list of option keywords.

Outputs

N/A

Exceptions

IllegalArgumentException: If from is not NULL.

IllegalArgumentException: If any of the address lines are manformed.

IllegalArgumentException: If to, cc, and bcc are all NULL.

IllegalArgumentException: If there is more than one address in replyTo.

IllegalArgumentException: If contentType is not TEXT_PLAIN or TEXT_HTML.

IllegalArgumentException: If type is not TABLE or PROCEDURE.

IllegalArgumentException: If path is malformed.

IllegalArgumentException: If parameters does not contain the right number or type of parameter values.

IllegalArgumentException: If options contains any unknown options.

IllegalStateException: If the server's from-address is not configured.

NotFoundException: If the specified view or procedure does not exist.

SecurityException: If the user does not have READ permission on all items in path other than the last one.

SecurityException: If the user does not have EXECUTE or SELECT permission on the last item in the list as appropriate.

SendTextMessage

Send a JMS text message.

Locations

/lib/jms/

/lib/util/

Inputs

connectorName: Name of the JMS connector.

destinationName: JNDI name of the JMS queue or topic.

text: Message body text.

Outputs

N/A

Exceptions

N/A

SetAlternatePrincipal

Establish an alternate identity within the current session, preserving the original identity for afterwards. This allows upstream servers to pool connections to TDV.

Location

/lib/util/

Inputs

user: The new user's name.

domain: The user's domain.

password: The user's password.

Outputs

N/A

Exceptions

SecurityException: If the identity specified is not known.

SecurityException: If the password is not valid for the user.

SetAlternateSecurityProperty

Set an alternate security property value to the identity within the current session. This is used to allow user passing security property to data source.

Location

/lib/users/

Inputs:

name: the security property name

value: the security property value

Outputs:

N/A

SetDataSourceCredentials

Set a username and password to use with pass-through authentication and a specific data source.

This procedure provides access to the facility described in the "Multiple Credentials for JDBC Connection" section of the *TDV User Guide*.

This procedure can be used in place of the JDBC facility when the incoming connection is received from a source other than JDBC (for example, Web Service or ODBC), or when the JDBC client cannot be modified.

The JDBC facility allows a NULL value to be specified for dbPath. This built-in procedure does not support a NULL dbPath, but an empty string for dbPath can be used to access the same behavior.

If the client does any sort of connection pooling or connection reuse, a subsequent user may have access to data accessible from these credentials. Always discard the connection or call either the clearAllDataSourceCredentials JDBC method or ClearAllDataSourceCredentials, page 270 before returning the connection to a connection pool.

Location

/lib/util/

Inputs

dbPath: The path identifying the data source to which these credentials pertain. If an empty string is provided, it is used as the default pass-through authentication credential.

username: The username to use when attempting to connect to a pass-through authentication data source.

password: The password to use when attempting to connect to a pass-through authentication data source.

Outputs

N/A

Exceptions

IllegalArgumentException: If any variable is set to an illegal value.

SetEnvironment

Set an environment variable to a value.

Location

/lib/util/

Syntax

```
setEnvironment (IN variableName VARCHAR (40),
IN propValue VARCHAR (2048))
```

Inputs

variableName: The name of a variable. Variable names are not case-sensitive. For example, both sample and SAMPLE are the same variable.

value: The new value for the variable.

Input Variable Names and Values

Note: Environment variables starting with `System.` are system-defined and may have restricted sets of legal values.

All built-in variable names are available on the Constants tab of the /lib/util/System SQL definition set. The variable names and their legal values are:

- System.CASE_SENSITIVE_IN_COMPARISONS: TRUE or FALSE. Reflects the case sensitivity being used in string comparisons for SQL and SQL script operations in this scope.
- System.IGNORE_TRAILING_SPACES_IN_COMPARISONS: TRUE or FALSE. Reflects whether or not trailing spaces are ignored in string comparisons for SQL and SQL script operations in this scope.
- System.NUM_ROWS_AFFECTED: A numeric value.
- System.TRIGGER_EVENT_NAME: The trigger name if the current request is the result of a trigger. NULL otherwise.
- System.TRIGGER_EVENT_TYPE: The trigger type if the current request is the result of a trigger. NULL otherwise.

- System.TRIGGER_EVENT_VALUE: The trigger value if the current request is the result of a trigger. NULL otherwise.
- System.TRIGGER_PATH: A path if the current request is the result of a trigger. NULL otherwise.

For backward compatibility, the following are also accessible without the `System.` prefix:

- CASE_SENSITIVE_IN_COMPARISONS
- IGNORE_TRAILING_SPACES_IN_COMPARISONS
- NUM_ROWS_AFFECTED
- TRIGGER_EVENT_NAME
- TRIGGER_EVENT_TYPE
- TRIGGER_EVENT_VALUE
- TRIGGER_PATH

Outputs

N/A

Exceptions

IllegalArgumentException: If an unsupported variable name is requested.

IllegalArgumentException: If an illegal value is set to a variable.

Example

```
PROCEDURE proc7()
    BEGIN
        DECLARE x VARCHAR(4096);
        CALL getEnvironment('NUM_ROWS_AFFECTED', x);
        CALL log(x);
        SET x = '100';
        CALL setEnvironment('NUM_ROWS_AFFECTED', x);
        CALL getEnvironment('NUM_ROWS_AFFECTED', x);
        CALL log(x);
    END
```

SetEnvironmentFromNodeValue

Evaluate an XPath expression against the envelope, and store the result in the specified environment variable. The result of the XPath expression is interpreted as a single string.

The namespacePrefixes and namespaceURIs are used to resolve prefixes to namespaces in the XPath expression. Each item in namespacePrefixes must have a corresponding item in namespaceURIs. The empty string specifies the default namespace.

Location

/lib/services/

Inputs

envelope: A SOAP envelope, which may include a WS Security header.

xpath: An XPath expression of the value.

namespacePrefixes: An array of namespace prefixes used in the XPath expression. May be NULL.

namespaceURIs: An array of namespace URIs used in the XPath expression. May be NULL.

variableName: The name of an environment variable. Variable names are not case- sensitive. For example, both sample and SAMPLE are the same variable.

Output

envelope: The SOAP envelope. Some elements may have been decrypted.

Exception

IllegalArgumentException: If any of the arguments are invalid.

SetMessageProperties

Set JMS headers or properties for the subsequent JMS messages to be sent using SendTextMessage, page 321.

Ordinarily this procedure is preferable to SetMessageProperty, page 327, which sets many property values as strings.

Note: You can use this procedure to set header values like JMSExpiration, but the values are overwritten by the producer using its own defaults. TDV cannot control this messaging library behavior.

Locations

/lib/jms/

/lib/util/

Input

properties: A ROW type where the type name corresponds to the property name. If the name matches a standard JMS header, a JMS header is created.

Outputs

N/A

Exceptions

N/A

SetMessageProperty

Set a JMS header or property for the subsequent JMS messages to be sent using SendTextMessage, page 321.

Note: To make sure properties you set have correct data types, it is better to use SetMessageProperties, page 326.

Locations

/lib/jms/

/lib/util/

Inputs

name: Property name. If the name corresponds to a standard JMS header, a JMS header is created.

value: Property value as string. If the value is NULL, the named property is cleared.

Outputs

N/A

Exceptions

N/A

SetNodeValueFromEnvironment

Set an element or attribute value from an environment variable.

The XPath expression is used to select a node from the envelope. The node value is then set to the value of the specified environment variable. The namespacePrefixes and namespaceURIs are used to resolve prefixes to namespaces in the XPath expression.

Each item in namespacePrefixes must have a corresponding item in namespaceURIs. The empty string is used to specify the default namespace.

Location

/lib/services/

Inputs

envelope: A SOAP envelope, which may include a WS Security header.

xpath: An XPath expression that evaluates to an element.

namespacePrefixes: An array of namespace prefixes used in the XPath expression. May be NULL.

namespaceURIs: An array of namespace URIs used in the XPath expression. May be NULL.

variableName: The name of an environment variable. Variable names are not case-sensitive. For example, both sample and SAMPLE are the same variable.

attributeName: If NULL, the element value is set. If not NULL, the element attribute value is set.

Output

envelope: The SOAP envelope. Some elements may have been decrypted.

Exception

IllegalArgumentException: If any of the arguments are invalid.

SignElement

Sign an element in the specified SOAP envelope using a private key.

Location

/lib/services/

Inputs

envelope: A SOAP envelope. It may not be NULL.

actor: Determines which WS Security header to process. It may be NULL.

mustUnderstand: Indicate whether or not the receiver must understand this header. It may be NULL. If NULL, mustUnderstand defaults to TRUE.

elementName: The name of the element in the envelope to sign. It may be NULL. If NULL, elementName defaults to *{http://schemas.xmlsoap.org/soap/envelope/}Body*.

certificateAlias: The alias of a private key in the key store to use to sign the element.

keyStore: A serialized Java key store containing the private key used to sign the element. It may be NULL.

keyStoreType: The type of key store. It must be JKS or PKCS12. It may not be NULL.

keyStorePassword: The password of the key store and of all private keys within it. It may be NULL or empty if there is no password.

Outputs

envelope: The SOAP envelope containing the signed element and generated WS Security header elements.

Exceptions

IllegalArgumentException: If any of the arguments are invalid.

SecurityException: If the element could not be signed.

SqlPerf

Run a SQL performance test. Each worker thread runs in a tight loop for the specified amount of time. With each iteration, a new transaction is created, the specified query is executed, and results are retrieved into TDV for evaluation and then discarded.

Location

/lib/resource/

Inputs

sql: Any valid SELECT statement in TDV SQL.

threads: Number of worker threads to use.

duration: Time in seconds.

Outputs

numExecutions: Total number of times the query was executed.

queriesPerSecond: Throughput.

rowsPerQuery: Number or rows the query returns.

queryTimeMillis: Average time to execute the query, in milliseconds.

queryTimeMillisMin: Minimum time in milliseconds to execute the query and retrieve results.

queryTimeMillisMax: Maximum time in milliseconds to execute the query and retrieve results.

SyncDomain

Synchronize the local external domain with the specified external domain server. Only domains other than COMPOSITE/DYNAMIC domains can be synchronized.

Location

/lib/util/

Inputs

domainName: Local domain to synchronize with the external domain server.

Outputs

N/A

Exceptions

NotFoundException: If the local domain does not exist.

SecurityException: If the procedure fails to synchronize local domain with the specified external domain server.

SecurityException: If the user does not have ACCESS_TOOLS and either MODIFY_ALL_USERS or MODIFY_ALL_STATUS rights.

TestAllDataSourceConnections

Test all data sources to see if they are operational.

Location

/lib/resource/

Inputs

N/A

Outputs

status: If SUCCESS, all enabled data sources are operational. If FAIL, at least one test failed. These values are available on the Constants tab of the /lib/util/System SQL definition set.

messages: A list of the messages generated during the test. Messages may be present in both SUCCESS and FAIL conditions. Messages are separated by newline characters.

Exceptions

SecurityException: If the user is not an administrator.

TestDataSourceConnection

Test to see if a data source's connection is operational.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Input

path: The path to the data source to be tested.

Outputs

status: If SUCCESS, the data source is operational. If FAIL, the test failed. These values are available on the Constants tab of the /lib/util/System SQL definition set.

messages: A list of the messages generated during the test. Messages may be present both in SUCCESS and FAIL conditions. Messages are separated by newline characters.

Exceptions

IllegalArgumentException: If path is malformed.

NotFoundException: If the data source does not exist.

SecurityException: If the user does not have READ access on all items in path.

TestUserIdentity

Allow a SQL script to determine if the current identity matches the one specified.

Location

/lib/util/

Inputs

type: USER or GROUP.

name: The user or group's name.

domain: The user or group's domain.

Output

result: TRUE if the current user or group corresponds to the inputs, otherwise FALSE.

Exceptions

N/A

UpdateResourceCacheEnabled

Update the enabled state of a resource cache.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource.

enabled: If TRUE, the resource cache is enabled. If FALSE, the resource cache is disabled.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type does not support being cached.

IllegalStateException: If the resource is not configured for caching.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

UpdateResourceCacheKeyStatus

Update the cache key for a specified resource of type TABLE, or of type PROCEDURE with zero parameters. Also see CreateResourceCacheKey, page 276 and GetResourceCacheStatus, page 297. The cache key identifies a snapshot of values for a query. The cache key and corresponding values are stored in the cache storage table.

The user may wish to create a cache key to assist with the manual insertion of data into the cache storage table. This procedure provides a switch to manually set the status to TRUE after inserting the data, to let the cache system know when the new snapshot is ready for use.

Use the message input to publish a note to the cache refresh status. All inputs are required, although an empty string may be entered if no message is desired.

This built-in procedure is discussed in the "TDV Caching" section of the *TDV User Guide*.

Location

/lib/resource/

Inputs

path: The path to the resource.

type: The type of the resource (TABLE or PROCEDURE).

cacheKey: The cache key from CreateResourceCacheKey.

status: Whether cache refresh succeeded (TRUE) or failed (FALSE).

startTime: Timestamp when the cache refresh starts.

message: If refresh failed, explains what was wrong.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type does not support being cached.

IllegalStateException: If the resource is not configured for caching.

IllegalStateException: If the data source used by the resource for caching is not properly configured.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

UpdateResourceEnabled

Update the enabled state of a DATA_SOURCE resource.

Note: This *procedure* is different from the *web services operation* of the same name, which is in the resource tree under /services/webservices/system/admin/resource/operations/.

Location

/lib/resource/

Inputs

path: The path to the DATA_SOURCE resource.

type: The type of the resource (DATA_SOURCE).

enabled: If TRUE, enables the resource; if FALSE, disables the resource.

Outputs

N/A

Exceptions

IllegalArgumentException: If the path is malformed or the type is illegal.

IllegalStateException: If the resource type is not DATA_SOURCE.

NotFoundException: If the resource does not exist.

SecurityException: If the user does not have READ access on all items in the path other than the last one.

SecurityException: If the user does not have WRITE access to the last item in path.

SQL Definition Sets

Groups of TDV built-in procedures have SQL definition sets associated with them. These definition sets define data types, exceptions, and constants for use in the procedures.

Note: For a discussion of definition sets, see the "Definition Sets" chapter of the *TDV User Guide*.

The SQL definition sets for TDV built-in procedures are these:

extendedSql SQL Definition Set

The extendedSql SQL definition set defines data types for extended SQL (ESQL).

Location

/lib/types/extendedSql

Types Tab

The Types tab contains definitions for Extended SQL procedure data types.

Data Type Name	Base Type
CURSOR	CURSOR
ROW	ROW
XML	(anonymous)

Jms SQL Definition Set

The Jms SQL definition set defines one data type for use with JMS procedures.

Location

/lib/jms/Jms

Types Tab

The Types tab contains the definition for the JMS procedure data type.

Data Type Name	Base Type
MapValue	ROW

ResourceDefs SQL Definition Set

The ResourceDefs SQL definition set defines data types and constants for use with resource procedures.

Location

/lib/resource/ResourceDefs

Types Tab

The Types tab contains definitions for resource procedure data types.

Data Type Name	Base Type
CopyMode	VARCHAR(255)
ParameterValues	VARCHAR(4096)
ReintrospectID	VARCHAR(255)
ReintrospectReport	VARCHAR(255)
ResourceName	VARCHAR(32767)
ResourcePath	VARCHAR(4096)
ResourceType	VARCHAR(40)
ResourceVersion	VARCHAR(255)

Constants Tab

The Constants tab contains definitions of resource procedure constants.

Constant Name	Type
ALTER NAME IF EXISTS	VARCHAR(255)
CSV ATTACH	VARCHAR(255)
FAIL IF EXISTS	VARCHAR(255)
OVERWRITE MERGE IF EXISTS	VARCHAR(40)
OVERWRITE REPLACE IF EXISTS	VARCHAR(40)
RESOURCE SUBTYPE BASIC TRANSFORM PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE CATALOG	VARCHAR(40)
RESOURCE SUBTYPE COMPOSITE WEB SERVICE	VARCHAR(40)
RESOURCE SUBTYPE DATABASE PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE DATABASE TABLE	VARCHAR(40)
RESOURCE SUBTYPE DELIMITED FILE	VARCHAR(40)
RESOURCE SUBTYPE DIRECTORY	VARCHAR(40)
RESOURCE SUBTYPE EXCEL NON ODBC POI DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE EXTERNAL SQL PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE FILE DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE FOLDER	VARCHAR(40)
RESOURCE SUBTYPE JAVA PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE NATIVE FUNCTION	VARCHAR(40)
RESOURCE SUBTYPE NONE	VARCHAR(40)
RESOURCE SUBTYPE OPERATION	VARCHAR(40)

Constant Name	Type
RESOURCE SUBTYPE PORT	VARCHAR(40)
RESOURCE SUBTYPE RELATIONAL DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE REST DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE SCHEMA	VARCHAR(40)
RESOURCE SUBTYPE SERVICE	VARCHAR(40)
RESOURCE SUBTYPE SQL SCRIPT PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE SQL TABLE	VARCHAR(40)
RESOURCE SUBTYPE STREAM TRANSFORM PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE TRANSFORM PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE WSDL DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE XML FILE	VARCHAR(40)
RESOURCE SUBTYPE XML FILE DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE XML HTTP DATA SOURCE	VARCHAR(40)
RESOURCE SUBTYPE XQUERY TRANSFORM PROCEDURE	VARCHAR(40)
RESOURCE SUBTYPE XSLT TRANSFORM PROCEDURE	VARCHAR(40)
RESOURCE TYPE CONTAINER	VARCHAR(40)
RESOURCE TYPE DATA SOURCE	VARCHAR(40)
RESOURCE TYPE DEFINITION SET	VARCHAR(40)
RESOURCE TYPE LINK	VARCHAR(40)
RESOURCE TYPE PROCEDURE	VARCHAR(40)
RESOURCE TYPE TABLE	VARCHAR(40)

Constant Name	Type
RESOURCE TYPE TREE	VARCHAR(40)
RESOURCE TYPE TRIGGER	VARCHAR(40)
SEND ERROR	VARCHAR(4096)
SKIP IF NO RESULTS	VARCHAR(4096)
SUMMARY	VARCHAR(4096)

sql SQL Definition Set

The sql SQL definition set defines data types and constants for use with SQL procedures.

Location

/lib/types/sql

Types Tab

The Types tab contains definitions for SQL procedure data types.

Data Type Name	Base Type
BIGINT	BIGINT
BINARY	BINARY(1024)
BIT	BIT
BLOB	BLOB
BOOLEAN	BOOLEAN
CHAR	CHAR(255)
CLOB	CLOB
DATE	DATE
DECIMAL	DECIMAL(32,2)
DOUBLE	DOUBLE

Data Type Name	Base Type
FLOAT	FLOAT
INTEGER	INTEGER
INTERVAL DAY	INTERVAL DAY
INTERVAL DAY TO HOUR	INTERVAL DAY TO HOUR
INTERVAL DAY TO MINUTE	INTERVAL DAY TO MINUTE
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND
INTERVAL HOUR	INTERVAL HOUR
INTERVAL HOUR TO MINUTE	INTERVAL HOUR TO MINUTE
INTERVAL HOUR TO SECOND	INTERVAL HOUR TO SECOND
INTERVAL MINUTE	INTERVAL MINUTE
INTERVAL MINUTE TO SECOND	INTERVAL MINUTE TO SECOND
INTERVAL MONTH	INTERVAL MONTH
INTERVAL SECOND	INTERVAL SECOND
INTERVAL YEAR	INTERVAL YEAR
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH
LONGVARBINARY	VARBINARY(1024)
LONGVARCHAR	VARCHAR(2147483647)
NUMERIC	NUMERIC(32,0)
REAL	FLOAT
SMALLINT	SMALLINT
TIME	TIME
TIMESTAMP	TIMESTAMP
TINYINT	TINYINT

Data Type Name	Base Type
VARBINARY	VARBINARY(1024)
VARCHAR	VARCHAR(255)

Constants Tab

The Constants tab contains definitions for SQL procedure constants.

Constant Name	Type	Value
BOOLEAN FALSE	BOOLEAN	false
BOOLEAN TRUE	BOOLEAN	true

System SQL Definition Set

The System SQL definition set defines data types, exceptions, and constants for use with system utility procedures.

Location

/lib/util/System

Types Tab

The Types tab contains definitions for system utility data types.

Data Type Name	Base Type
Content	VARCHAR(65535)
EMailAddress	VARCHAR(1024)
MapValue	ROW
MessageValue	VARCHAR(64000)
OperationStatus	VARCHAR(255)
PropertyName	VARCHAR(255)
PropertyValue	VARCHAR(4096)

Data Type Name	Base Type
Text	VARCHAR(2147483647)

Exceptions Tab

The Exceptions tab lists the System utility exceptions. For an explanation of these exceptions, refer to the "TDV SQL Script" chapter of the *TDV Reference Manual.*

Exception Name
CannotExecuteSelectException
CannotOpenCursorException
CannotOpenNonSelectException
CursorAlreadyOpenException
CursorNotOpenException
CursorTypeMismatchException
DuplicateNameException
EvaluationException
IllegalArgumentException
IllegalStateException
NotAllowedException
NotFoundException
NotSupportedException
NullVariableException
ParseException
PipeNotOpenException
ProcedureClosedException
ProtocolException

Exception Name
SOAPFaultException
SecurityException
SystemException
TransactionClosedException
TransactionFailureException
UnexpectedRowCountException
UnopenedCursorException

Constants Tab

The Constants tab contains System utility constant definitions. The values of the constants marked with an asterisk (*) begin with a “System.” prefix.

Constant Name	Type
CACHED RESOURCE BUCKET PATH*	VARCHAR(255)
CACHED RESOURCE CACHE KEY*	VARCHAR(255)
CACHED RESOURCE ERROR MESSAGE*	VARCHAR(255)
CACHED RESOURCE INCREMENTAL MAINTENANCE LEVEL*	VARCHAR(255)
CACHED RESOURCE PARAM KEY*	VARCHAR(255)
CACHED RESOURCE PATH*	VARCHAR(255)
CACHED RESOURCE REFRESH OUTCOME*	VARCHAR(255)
CACHED RESOURCE TYPE*	VARCHAR(255)
CACHE DATASOURCE PATH*	VARCHAR(255)
CACHE IS INCREMENTAL*	VARCHAR(255)
CANCELED	VARCHAR(255)

Constant Name	Type
CASE SENSITIVE IN COMPARISONS*	VARCHAR(255)
CLUSTER ID	VARCHAR(255)
CURRENT USER DOMAIN	VARCHAR(255)
CURRENT USER ID	VARCHAR(255)
CURRENT USER NAME	VARCHAR(255)
FAIL	VARCHAR(255)
IGNORE TRAILING SPACES IN COMPARISONS*	VARCHAR(255)
INCOMPLETE	VARCHAR(255)
NUM ROWS AFFECTED*	VARCHAR(255)
SERVER HOSTNAME	VARCHAR(255)
SERVER ID	VARCHAR(255)
SERVER JDBC PORT	VARCHAR(255)
SERVER VERSION	VARCHAR(255)
SERVER WEB PORT	VARCHAR(255)
SESSION ID	VARCHAR(255)
SUCCESS	VARCHAR(255)
TEXT HTML	VARCHAR(255)
TEXT PLAIN	VARCHAR(255)
TRANSACTION ID	VARCHAR(255)
TRIGGER EVENT NAME*	VARCHAR(255)
TRIGGER EVENT TYPE*	VARCHAR(255)
TRIGGER EVENT VALUE*	VARCHAR(255)
TRIGGER PATH*	VARCHAR(255)

UserDefs SQL Definition Set

The UserDefs SQL definition set defines data types for use with user-related procedures.

Location

/lib/users/UserDefs

Types Tab

The Types tab contains definitions for user procedure data types.

Data Type Name	Base Type
DomainName	VARCHAR(255)
UserGroups	CURSOR
type	CHAR(1)
id	INTEGER
domain	VARCHAR(255)
name	VARCHAR(255)
UserGroupsType	CHAR(1)
UserName	VARCHAR(255)

Server Actions

Server actions are basic TDV actions that can be made available to Web users.

About Server Actions

Server actions provide a way for Web services operations to perform basic actions in the TDV Server by defining those actions in an XML payload sent to the server by executing performServerAction, page 178.

Server Actions Reference

This section describes all TDV/Studio built-in server actions, in alphabetical order.

CheckLicense

Test whether the TDV has a valid license for the specified component. If the license is valid, this server action simply returns. Otherwise a fault is generated.

Argument

component (STRING): The license component name to check.

Faults

NotAllowed: If a valid license does not exist for the specified component.

Security: If the caller does not have the ACCESS_TOOLS rights.

ClearDataSourceConnectionPools

Clear the connection pools of the specified data sources.

Argument

dataSourcePaths (STRING_ARRAY): Resource paths to the data sources for which the connection pools are to be cleared.

Sample Request

```
<ns1:performServerAction
xmlns:ns1="http://www.compositesw.com/services/system/admin/server
" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <ns1:actionName>ClearDataSourceConnectionPools</ns1:actionName>
 <ns1:attributes>
  <ns2:attribute
xmlns:ns2="http://www.compositesw.com/services/system/util/common">
   <ns2:name>dataSourcePaths</ns2:name>
   <ns2:type>STRING_ARRAY</ns2:type>
   <ns2:valueArray>
    <ns2:item>/users/composite/admin/mysql</ns2:item>
   </ns2:valueArray>
  </ns2:attribute>
 </ns1:attributes>
</ns1:performServerAction>
```

Sample Response

```
<server:performServerActionResponse
xmlns:server="http://www.compositesw.com/services/system/admin/ser
ver">
 <status>SUCCESS</status>
 <server:messages>
  <common:entry
xmlns:common="http://www.compositesw.com/services/system/util/common">
   <common:code>9901130</common:code>
   <common:name>ClearDataSourceConnectionPools</common:name>
   <common:message>Data source connection pool(s)
cleared</common:message>
  </common:entry>
 </server:messages>
</server:performServerActionResponse>
```

Fault

Security: If the caller does not have the ACCESS_TOOLS right and either the MODIFY_ALL_STATUS right or WRITE access to the specified data sources.

ClearRepositoryCache

Clear and reset the repository cache.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

ClearQueryPlanCache

Clear and reset the query plan cache.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

ClearServerProfile

Clear and reset the server profile.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

Echo

Echo the given message, with an optional prefix. The message is returned within the messages element, optionally preceded with a prefix.

Arguments

message (STRING): The message to echo.

prefix (STRING; optional): An optional prefix for the message.

FreeUnusedMemory

Free all unused server memory.

GetServerProfile

Get the current server profile statistics.

Fault

Security: If the caller does not have ACCESS_TOOLS and READ_ALL_STATUS rights.

PurgeCompletedRequests

Purge all completed requests from the server.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

PurgeCompletedSessions

Purge all completed sessions from the server.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

PurgeCompletedTransactions

Purge all completed transactions from the server.

Fault

Security: The caller must have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

RegenerateFiles

Regenerate all files that have CFGT templates.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_CONFIG rights.

ResetSystemNamespace

Reset the system namespace; that is, destroy and recreate the /services/databases/system, /services/webservices/system, and /lib portion of the resource namespace.

The resetting is done in a separate transaction, and the results are not visible until after the current transaction is closed.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_RESOURCES rights.

ShutdownServer

Shut down the server.

If the server is already shutting down, the server is shut down immediately. If this shutdown action was not directly requested by the monitor, the monitor reports it as an unplanned shutdown.

ShutdownServer is an asynchronous call that attempts to complete its processing and return prior to commencement of the shutdown. Because it is possible for the connection to be interrupted before the action can be completed, you can specify delayMs to postpone when shutdown begins.

The server attempts to shut down as cleanly as possible. However, if doing so takes longer than timeoutMs milliseconds after the end of delayMs, the server is immediately shut down. Therefore, if both arguments are specified, the maximum time for the server to be shut down is delayMs plus timeoutMs.

Arguments

delayMs (LONG; optional): The number of milliseconds to wait before shutting down. A delayMs of less than or equal to 0 means no delay. Default is 0 ms.

timeoutMs (LONG; optional): The number of milliseconds to wait before forcing a hard shutdown. If timeoutMs is negative, the server is allowed to take as long as it needs to shut down. If timeoutMs is 0, the server is shut down without waiting. Default is 10000 ms.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

TerminateRequests

Terminate the specified server requests.

Argument

requestIds (LONG_ARRAY): The IDs of the requests to terminate.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

TerminateSessions

Terminate the specified server sessions.

Argument

sessionIds (LONG_ARRAY): The IDs of the sessions to terminate.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

TerminateTransactions

Terminate the specified server transactions.

Argument

transactionIds (LONG_ARRAY): The IDs of the transactions to terminate.

Fault

Security: If the caller does not have the MODIFY_ALL_STATUS right.

TestAllDataSources

Update the status of all data sources in the server. Test results can be queried using the SYS_DATA_SOURCES system table. This action returns after all tests have been completed.

Fault

Security: If the caller does not have ACCESS_TOOLS and MODIFY_ALL_STATUS rights.

DSL API

DSL (Domain Specific Language) API enables users to perform headless development on the TDV server.

The DSL API can be executed through REST/JDBC/ODBC/ADO.Net or any other client using which you can query the TDV Server.

This chapter explains the following resources that can be created/altered/deleted using DSL:

Note: Refer to the chapter TDV Server REST APIs for more details on how to perform the tasks explained in this chapter using REST APIs.

Data Sources

Using DSL & REST APIs, datasources (such as relational, file-delimited, excel) can be created, deleted and updated. Relational datasource and its properties can be read from system tables under /services/databases/system/model. The various functions performed on the datasources can be specified using the DSL-based specific keywords or by using the Native properties syntax. Native properties syntax accepts property name value pairs in which attribute name and value can be specified. Attribute definitions of a datasource may be obtained from the system table:
/services/databases/system/model/SYS_DATASOURCE_ATTRIBUTE_DEFS.

Refer to System Tables for a detailed description of the system tables that stores the information about different data sources.

The following data sources are discussed in this section:

Relational Data Sources, page 357

File Delimited Data Sources, page 359

Excel Data Sources, page 361

DSL Syntax

Following is the syntax for Create, Update Delete of a datasource. A data source could be created using a database adapter and full introspection of the datasource is performed. The datasource could be altered to specify connection properties, authentication properties and annotation later on.

```
DROP DATASOURCE /path/name

_____________________________________________________________

CREATE|ALTER DATASOURCE /path/name

(RELOCATE AS /path/name)?                                    //only for alter

BASED ON ADAPTER 'Oracle 11g (Thin Driver)| File-Delimited' //adapter name, maybe
relational, file-delimited, excel

(SET NATIVE PROPERTIES {json_object})?

(CONNECT USING

    (HOST 'xxx.xxx.xxx.xx')?                                  //only for relational
    (PORT 1521)?                                              //only for relational
    (DATABASE_NAME 'ordersDatabase')?                         //only for relational
    (SELECT_MODE 'Direct')?                                  //for microsoft sql server
datasource url


    (LOCAL_ROOT_PATH 'localPath' | URL 'url')?             //for file-delimited and MS
Excel (non ODBC)
    (FILE_FILTERS '*.xls,*.xlsx')?                        //for file-delimited(default
= *.csv,*.txt) and excel (default = *.xls,*.xlsx)

    (DSN 'excelodbc')?                                      //for MS Excel (ODBC))?

-- relational
(AUTHENTICATE

    (IN [BASIC | KERBEROS] MODE)?                          //for relational default =
BASIC

    USING

    (DOMAIN 'domainName')?                                  //for file-delimited

    (LOGIN 'username')?                                     //for relational and
file-delimited
    (PASSWORD 'password')?
```

```
    (PASS_THROUGH_LOGIN [TRUE|FALSE])?                    //for relational basic mode
of authentication
    (TICKET_CACHE 'ticketCache' | NULL)?                   //for relational kerberos
mode of authentication
    (USE_PASS_THROUGH_CERTIFICATE_FOR_ENCRYPTION [TRUE|FALSE])? )?      //for
relational kerberos mode of authentication default = false

)?

(FORMAT USING

        (CHARACTER_SET 'iso-8859-1')?                      //for file-delimited and
excel, defaults based on adapter for allowable values : Refer Section 1, 2, 3

       (DELIMITER ',')?                                   //for file-delimited default
= ','allowable values : Refer Section 1, 2

       (TEXT_QUALIFIER '"')?                              //for file-delimited default
= '"' allowable values : Refer Section 1, 2

       (STARTING_ROW 1)?                                  //for file-delimited default
= 1

       (HAS_HEADER_ROW [TRUE|FALSE])?                     //for file-delimited(default
= FALSE) and excel(default = TRUE)

       (IGNORE_TRAILING_DELIMITER)?                       //for file-delimited default
= TRUE

        (DATA_RANGE 'A1')?                                  //for MS Excel (non-ODBC)
default = A1

        (BLANK_COLUMN_TYPE 'Varchar')?                     //for excel default =
Varchar, allowable values: Varchar, Double, Boolean, Datetime

        (FIRST_ROW_CATEGORY [TRUE | FALSE])?               //for excel default = true

        (IGNORE_INVALID_DATA_FETCH [TRUE | FALSE])?        //for excel default = true

        (DATA_WYSIWYG [TRUE|FALSE])?                       //for excel default=true

       (BLANK_AS_NULL [TRUE|FALSE])?                      //for excel default = true)?

(SET ANNOTATION 'this is a datasource created using DSL api' | NULL)?
```

Relational Data Sources

This section describes the usage of DSL APIs for creating, altering or deleting a relational data source.

Creating a Relational Data Source with Native Connection Properties

Native properties may be specified using "SET NATIVE PROPERTIES" in the DSL syntax with name value pairs in JSON format. The JSON format can be specified as key value pairs and the keys represent the definition_name taken from model.SYS_DATASOURCE_ATTRIBUTE_DEFS table. The definition_name may be specified in a case-insensitive manner.

Considerations

Listed below are some points to consider while working with Relational Data sources using native properties syntax:

1. Collision between native properties with standard DSL expanded form will throw error. For example if "urlIP" is specified as a property in SET NATIVE PROPERTIES syntax and also specified in "CONNECT USING HOST 'xxx'" syntax, error will be displayed indicating this.
2. Warning when properties are used when a DSL expanded syntax is available for a property/definition. For example, if "urlIP" is specified as a property in SET NATIVE PROPERTIES syntax and not specified using a standard DSL syntax as "CONNECT USING HOST", a warning will be logged in the cs_server.log indicating the syntax that can be used for this particular property. This is done to encourage users to use DSL syntax whenever possible. However, if there is no corresponding DSL syntax for a property, then no warning is logged. Refer to the section Examples, page 358 for an understanding of how to create or alter a data source.

Examples

```
curl -d "[\"CREATE DATASOURCE /shared/examples/relationalDS BASED ON
ADAPTER 'PostgreSQL 9.1' SET NATIVE PROPERTIES
{\\\"urlIP\\\":\\\"localhost\\\", \\\"urlPort\\\":9408,
\\\"urlDatabaseName\\\":\\\"orders\\\", \\\"login\\\":\\\"tutorial\\\",
\\\"password\\\":\\\"password\\\"} SET ANNOTATION 'this is a Postgres
datasource created using DSL api native properties'\"]" -u "admin:admin" -X
POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Creating a datasource with DSL keywords

Considerations

Listed below are some points to consider while working with Relational Data sources using DSL keywords:

1. When connection properties are not specified during creation, introspection is not done on the datasource as its a required information for introspection. Datasource is created without introspection. When the datasource is altered later to add connection properties, full introspection of the datasource is done.
2. When no authentication properties are specified during creation, an error will be displayed during introspection, but the datasource will be created successfully.

3. For QUOTED_STRING ticket_cache, an explicit NULL keyword or 'NULL' (as a quoted string) can be specified. If null is invalid for some reason during introspection, an error will be displayed.

Note: If an unexpected input is given or if a resource is missing, an error is displayed.

Examples

Create a Postgres datasource ds1

```
curl -d "[\"CREATE DATASOURCE /shared/examples/ds1 BASED ON ADAPTER
'PostgreSQL 9.1'
        CONNECT USING host 'localhost' port 5432 DATABASE_NAME 'orders'
        AUTHENTICATE IN BASIC MODE USING LOGIN 'tutorial' PASSWORD
'password' PASS_THROUGH_LOGIN_TRUE
        SET ANNOTATION 'this is a datasource created using DSL api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create an Oracle datasource oracleds

```
curl -d "[\"CREATE DATASOURCE /shared/examples/oracleds BASED ON ADAPTER
'Oracle 11g (Thin Driver)'
        CONNECT USING host 'xxx.xx.x.xx' port 1521 DATABASE_NAME 'xxxxx'
        AUTHENTICATE IN BASIC MODE USING LOGIN 'user' PASSWORD 'password'
        SET ANNOTATION 'this is an oracle datasource created using DSL
api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create a Postgres datasource ds1 with space in its name

```
curl -d "[\"CREATE DATASOURCE /shared/examples/\\\"ds 1\\\" BASED ON
ADAPTER 'PostgreSQL 9.1'
        CONNECT USING host 'localhost' port 5432 DATABASE_NAME 'orders'
        AUTHENTICATE IN BASIC MODE USING LOGIN 'tutorial' PASSWORD
'password' PASS_THROUGH_LOGIN_TRUE
        SET ANNOTATION 'this is a datasource created using DSL api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

File Delimited Data Sources

This section describes the usage of DSL APIs for creating, altering or deleting a file-delimited data source.

Considerations

Listed below are some points to consider while working with File-Delimited Data Sources:

1. When the parameters "root" and "url" are both provided, an error message is displayed.
2. During an alter operation, the properties that were previously set during a create will not be altered unless a new value or NULL is specified.

Examples

Following are examples to create a file-delimited datasource:

Create a file-delimited datasource

```
curl -d "[\"CREATE DATASOURCE /shared/examples/excelds
            BASED ON ADAPTER 'File-Delimited'
            CONNECT USING LOCAL_ROOT_PATH '/Users/shared/csv' FILE_FILTERS
'*.csv'
FORMAT USING CHARACTER_SET 'utf-8'
          SET ANNOTATION 'this is a file delimited datasource created using
DSL api'
        \"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
-H "Content-Type:application/json"
```

Create with native properties

```
//Error: root path specified in both LOCAL_ROOT_PATH as well as native
properties syntax
curl -d "[\"CREATE DATASOURCE /shared/examples/fileds BASED ON ADAPTER
'File-Delimited' SET NATIVE PROPERTIES {\\\"root\\\":\\\"/Users/Shared\\\"}
CONNECT USING LOCAL_ROOT_PATH '/Users/Shared' FILE_FILTERS '*.csv,*.txt'
FORMAT USING CHARACTER_SET 'utf-8' SET ANNOTATION 'this is a file delimited
datasource created using DSL api'\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"

'root' already specified in DSL as 'local root path'

//Warning logged in server_log but datasource created. root unspecified
using DSL syntax but specified in native properties
curl -d "[\"CREATE DATASOURCE /shared/examples/fileds BASED ON ADAPTER
'File-Delimited' SET NATIVE PROPERTIES {\\\"root\\\":\\\"/Users/Shared\\\"}
CONNECT USING FILE_FILTERS '*.csv,*.txt' FORMAT USING CHARACTER_SET 'utf-8'
SET ANNOTATION 'this is a file delimited datasource created using DSL
```

```
api'\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"

WARN 2019-05-07 10:08:46.637 -0700 CreateOrAlterResourceFilter - 'root'
could be specified in DSL instead of native property using syntax like
'CONNECT USING local_root_path'
```

Excel Data Sources

This section describes the usage of DSL APIs for creating, altering or deleting a Excel data source.

Considerations

Listed below are some points to consider while working with Excel Data Sources:

- During an alter operation, the properties that were previously set during a create will not be altered unless a new value or NULL is specified.

Examples

MS Excel (non-ODBC) Data Source Creation

```
//create excel datasource using DSL syntax
curl -d "[\"CREATE DATASOURCE /shared/examples/excelds
        BASED ON ADAPTER 'Microsoft Excel (non-ODBC)'
        CONNECT USING LOCAL_ROOT_PATH '/Users/Shared/excel'
        FORMAT USING CHARACTER_SET 'utf-8'
        SET ANNOTATION 'this is a excel datasource created using DSL
api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"


//specify create ds using only native properties
curl -d "[\"CREATE DATASOURCE /shared/examples/excelds
        BASED ON ADAPTER 'Microsoft Excel (non-ODBC)'
        SET NATIVE PROPERTIES {\\\"root\\\":\\\"/Users/Shared/excel\\\",
                               \\\"filters\\\":\\\"*.xls\\\",
                               \\\"charset\\\":\\\"utf-8\\\"}
        SET ANNOTATION 'this is a excel datasource created using DSL
api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

MS Excel (ODBC) only on windows

```
//create excel datasource DSL
curl -d "[\"CREATE DATASOURCE /shared/examples/excelds
        BASED ON ADAPTER 'Microsoft Excel'
        CONNECT USING DSN 'excelodbc'
        FORMAT USING CHARACTER_SET 'utf-8'
        SET ANNOTATION 'this is a excel ODBC datasource created using DSL
api'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"

//specify create ds using native properties only --expect warning messages
in logs
curl -d "[\"CREATE DATASOURCE /shared/examples/nativeexcelds4
    BASED ON ADAPTER 'Microsoft Excel'
    SET NATIVE PROPERTIES {\\\"dsn\\\":\\\"excelodbc\\\",
\\\"charset\\\":\\\"utf-8\\\"}
    SET ANNOTATION 'this is a excel datasource created using DSL api native
properties'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

System Tables

There are 3 system tables under /services/databases/system/model directory, representing information about datasources:

ALL_DATASOURCES

ALL_RESOURCE_PROPERTIES

SYS_DATASOURCE_ATTRIBUTE_DEFS

ALL_DATASOURCES

The model.ALL_DATASOURCES table is similar to /services/databases/system/ALL_DATASOURCES, however, it includes both published as well as non-published datasources. It also includes an extra column "IS_PUBLISHED" that indicates if the datasource is published or not. Information about these columns can be viewed from the "Info" tab of the respective system table from Studio. The columns of this table are :

Column Name	TDV JDBC DataType	Nullable	Description
DATASOURCE_ID:	INTEGER		Identifier of the data source

Column Name	TDV JDBC DataType	Nullable	Description
DATASOURCE_NAME:	VARCHAR		Datasource Name
DATASOURCE_TYPE:	VARCHAR		Type of Datasource
ADAPTER_NAME	VARCHAR		Name of the Adapter
IS_PUBLISHED	VARCHAR		Identifies if datasource is published or not
GUID:	VARCHAR		128 bit unique identifier.
ANNOTATION	VARCHAR	Yes	Annotation for the data source
OWNER_ID	INTEGER		Identifier of the user who created/owns the data source.
OWNER	VARCHAR		User name of the person that owns/created the data source
PARENT_PATH	VARCHAR		Path to the parent container
DATASOURCE_CREATOR_ID	INTEGER		Identifier of the user who created this data source.
DATASOURCE_CREATION_TIMESTAMP	BIGINT		Timestamp when the data source was created.
DATASOURCE_MODIFIER_ID	INTEGER		Identifier of the user who last modified this data source.
DATASOURCE_MODIFICATION_TIMESTAMP	BIGINT		Timestamp of the last modification of this data source.
ADAPTER_TYPE_CATEGORY	VARCHAR		Adapter type category. Could be one of RELATIONAL, EXCEL_FILE, WEBSERVICE, REST, DELIMITED_FILE, XML_FILE, CJP.

ALL_RESOURCE_PROPERTIES

The columns of this table are:

Column Name	TDV JDBC Data Type	Nullable	Description
METADATA_ID	INTEGER		Primary key identifier of the table
PROPERTY_NAME	VARCHAR		Name of the property
DATA_TYPE	VARCHAR		Data type of the property
PROPERTY_VALUE	VARCHAR		Value of the property

SYS_DATASOURCE_ATTRIBUTE_DEFS

This table contains the datasource attribute definitions for all adapters. The information from this table could be used in the SET NATIVE PROPERTIES syntax of DSL API while creating datasources:

Column Name	TDV JDBC Data Type	Nullable	Description
ADAPTER_NAME:	VARCHAR		Name of the database adapter
ADAPTER_TYPE	VARCHAR		Type of the adapter
ADAPTER_TYPE_CATEGORY	VARCHAR		Adapter type category.
DEFINITION_NAME:	VARCHAR		Name of the attribute definition
DISPLAY_NAME:	VARCHAR		Display name used for this attribute definition
DEFINITION_TYPE:	VARCHAR		Type of the attribute definition
REQUIRED:	BIT		Indicates if the attribute definition is mandatory
DEFAULT_VALUE:	VARCHAR		Default value of the attribute definition

Column Name	TDV JDBC Data Type	Nullable	Description
ALLOWED_VALUES:	VARCHAR		Allowed values for this attribute definition
EDITOR_HINT:	VARCHAR		Editor hint of this attribute definition
IS_ADVANCED	BIT		Indicates if the attribute is an advanced attribute
DISPLAY_PARENT_NAME	VARCHAR		Parent of the DISPLAY_NAME column
DEPENDENCY_EXPRESSION	VARCHAR		Indicates the attribute definition's "Depends on Item" and "Depends on Value”.
UPDATE_RULE	VARCHAR		Update rule. Could be one of READ_ONLY, READ_WRITE, WRITE_ON_CREATE,WRITE_ON_EDIT, WRITE_ON_IMPORT
ANNOTATION	VARCHAR		Annotation for the attribute definition
DEFINITION_PARENT_NAME	VARCHAR		Attribute definition's parent name usually prefixed with adapter name so as to resolve similar definition names in different adapters.

Logging

To turn on debug logging, set the following in conf/server/log4j.properties

```
log4j.logger.com.compositesw.server.qe.physical.ddl=DEBUG
```

When a datasource is created, altered or dropped, the log messages will indicate the command being executed.

Data Views

Using DSL & REST APIs, Data Views can be created, altered or deleted. Data View and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the DSL syntax mentioned below,, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to CUD views:

```
DROP DATA VIEW (IF EXISTS)? /path/name
```

```
CREATE DATA VIEW (IF NOT EXISTS)? /path/name
DEFINE AS  sql
(SET PROPERTIES {"name":{value}})?
(SET ANNOTATION 'this is a view created using DSL api' | NULL)?
```

```
ALTER DATA VIEW /path/name
(RELOCATE AS /path/newName)?
(DEFINE AS  sql)?
(SET PROPERTIES {"name":{value}})?
(SET ANNOTATION 'this view is altered using DSL api' | NULL)?
```

```
SELECT * FROM model.ALL_TABLES where TABLE_NAME = 'name';
```

```
 SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        PROPERTY_NAME = 'propName' AND

        METADATA_ID =
            (SELECT TABLE_ID FROM model.ALL_TABLES

            WHERE TABLE_NAME = 'name' AND //name of view

            PARENT_PATH = '/path'); //parent path of the view
```

Note: Properties and annotation are optional when creating a data view.

Considerations

Listed below are some points to consider while working with Data Views:

1. SQL is an attribute of type String. When a data view is created with a definition sql with a String value for it, the attribute "sql" is added with the definition sql on the view resource that is created. The definition sql can also be altered.
2. When renaming a view or relocating a view to a new location, RELOCATE AS is used in the syntax. Both rename and relocate can be performed in the same command.
3. When a data view is created or altered, the model.ALL_RESOURCE_PROPERTIES table is updated with the properties of the view : "sql". This is the attribute that is set on a data view when it is created using the DSL api. To read the value of a property from model.ALL_RESOURCE_PROPERTIES table, the "rest/execute/v1/actions/query/invoke" can be used.

Examples

Create view with only sql

```
curl -d "[\"CREATE DATA VIEW /shared/examples/myview DEFINE AS SELECT
OrderID FROM /shared/examples/ViewOrder\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create view with sql and annotation

```
curl -d "[\"CREATE DATA VIEW /shared/examples/myview DEFINE AS SELECT
OrderID FROM /shared/examples/ViewOrder SET ANNOTATION 'this view is
created using DSL'\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create data view IF NOT EXISTS with sql and annotation

```
curl -d "[\"CREATE DATA VIEW IF NOT EXISTS /shared/examples/myview DEFINE
AS SELECT OrderID FROM /shared/examples/ViewOrder SET ANNOTATION 'this view
is created using DSL'\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

System Tables

The following system tables under /services/databases/system/model directory, store the information about data sources:

ALL_TABLES

This table displays both the published and non-published tables and views. "VIEW" is displayed for TABLE_TYPE when the table is a view. The table model.ALL_TABLES table has the following columns :

Column name	TDV JDBC Data Type	Nullable	Description
TABLE_ID:	INTEGER		Primary key identifier of the table.
TABLE_NAME:	VARCHAR		Name of the table.
TABLE_TYPE:	VARCHAR		Possible values of this column are 'VIEW' and 'TABLE".
SCHEMA_ID:	INTEGER	Yes	See SCHEMA_ID in Table: model.ALL_SCHEMAS
SCHEMA_NAME:	VARCHAR	Yes	See SCHEMA_NAME in Table: model.ALL_SCHEMAS
CATALOG_ID:	INTEGER	Yes	See CATALOG_ID in Table: model.ALL_CATALOGS
CATALOG_NAME:	VARCHAR	Yes	See CATALOG_NAME in Table: model.ALL_CATALOGS
DATASOURCE_ID:	INTEGER		See DATASOURCE_ID in Table: model.ALL_DATASOURCES
DATASOURCE_NAME:	VARCHAR		See DATASOURCE_NAME in Table: model.ALL_DATASOURCES
GUID:	VARCHAR		128 bit unique identifier
ANNOTATION:	VARCHAR	Yes	Annotation for the table.

Column name	TDV JDBC Data Type	Nullable	Description
OWNER_ID:	INTEGER		Identifier of the person who created/owns the table. Same as USER_ID in Table: ALL_USERS
OWNER:	VARCHAR		Name of the person who created/owns the table. Same as USERNAME in Table: ALL_USERS
PARENT_PATH:	VARCHAR		Path to the parent container
TABLE_CREATOR_ID:	INTEGER		Identifier of the user who created this table. Same as USER_ID in Table: ALL_USERS
TABLE_CREATION_TIMESTAMP:	BIGINT		Timestamp when the table was created.
TABLE_MODIFIER_ID:	INTEGER		Identifier of the user who last modified this table. Same as USER_ID in Table: ALL_USERS
TABLE_MODIFICATION_TIMESTAMP:	BIGINT		Timestamp of the last modification of this table.
IMPACT_MESSAGE	VARCHAR		Impact message indicating errors
IS_AUTO_GENERATED	VARCHAR		Indicates if the table has an attribute called “webui_model”. If the attribute is present, then the value of this column is 'YES' else it is 'NO’.

ALL_COLUMNS

When a data view is created, the columns of the view are added to the model.ALL_COLUMNS table.

Column name	TDV JDBC Data Type	Nullable	Description
COLUMN_ID:	INTEGER		Primary key identifier of the column
COLUMN_NAME:	VARCHAR		Name of the column
DATA_TYPE:	VARCHAR		String representation of the data type
ORDINAL_POSITION:	INTEGER		Position of this column in relation to other columns in the same table
JDBC_DATA_TYPE:	SMALLINT		JDBC/ODBC data types. For JDBC data types, see http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Types.html
COLUMN_LENGTH:	INTEGER		If it is a CHAR or VARCHAR, the length is the maximum length allowed. If it is a DECIMAL or NUMERIC, then the value is the total number of digits. If it is none of the above types, then the value is NULL.
COLUMN_PRECISION:	INTEGER		If it is a DECIMAL or NUMERIC data type, then it is the number of digits. If it is not a DECIMAL or NUMERIC data type, then the value is NULL.
COLUMN_SCALE:	INTEGER		
COLUMN_RADIX:	INTEGER		10 for all numeric data types. Null for all non-numeric types.

Column name	TDV JDBC Data Type	Nullable	Description
NULLABLE:	SMALLINT		Indicates whether the column is nullable 0 if NULL is not allowed 1 if NULL is allowed 2 if it is unknown
IS_NULLABLE:	VARCHAR		Indicates whether the column is nullable YES if it is nullable NO if it is not nullable Blank string is returned if value is not known
TABLE_ID:	INTEGER		See TABLE_ID in Table: model.ALL_TABLES
TABLE_NAME:	VARCHAR		See TABLE_NAME in Table: model.ALL_TABLES
SCHEMA_ID:	INTEGER	Yes	See SCHEMA_ID in Table: model.ALL_SCHEMAS
SCHEMA_NAME:	VARCHAR	Yes	See SCHEMA_NAME in Table: model.ALL_SCHEMAS
CATALOG_ID:	INTEGER	Yes	See CATALOG_ID in Table: model.ALL_CATALOGS
CATALOG_NAME:	VARCHAR		See CATALOG_NAME in Table: model.ALL_CATALOGS
DATASOURCE_ID:	INTEGER		See DATASOURCE_ID in Table: model.ALL_DATASOURCES
DATASOURCE_NAME:	VARCHAR		See DATASOURCE_NAME in Table: model.ALL_DATASOURCES

Column name	TDV JDBC Data Type	Nullable	Description
ANNOTATION:	VARCHAR	Yes	Annotation for the column
OWNER_ID:	INTEGER		Identifier for the user who created/owns the column. Same as USER_ID in Table: ALL_USERS
OWNER:	VARCHAR		User name of the person that owns/created the data source. Same as USERNAME in Table: ALL_USERS
PARENT_PATH:	VARCHAR		Path to the parent container.

Logging

Create Data View

During creation of a data view, a debug message is logged like below :

```
  DEBUG 2019-03-05 16:13:21.889 -0800 DDLNode - Data View
'/shared/examples/myviewWithProps' created with sql 'SELECT OrderID
  FROM      /shared/examples/ViewOrder'
  DEBUG 2019-03-05 16:13:21.889 -0800 DDLNode - Set annotation on resource:
/shared/examples/myviewWithProps
  Annotation: this view is created using DSL
```

"IF NOT EXISTS" can be specified in the CREATE DATA VIEW command optionally. If the data view being created does not exist, its then created. If resource already exists, no error is thrown when IF NOT EXISTS keywords are specified.

Delete Data View

During deletion of a data view, if debug logging is enabled, the following is logged

```
DEBUG 2019-03-05 16:11:52.452 -0800 DDLNode - Deleted :
/shared/examples/myviewWithProps
```

"IF EXISTS" can be specified optionally with the DROP DATA VIEW command. If the data view being dropped exists, it is dropped. If the data view being dropped does not exist, and "IF EXISTS" keywords are used, no error is thrown.

Alter Data View

When a data view is altered, a definition sql may or may not be provided. When a definition sql is provided which is not the same as the existing definition sql, the new sql is set on the view resource.

Annotation is optional, with the alter syntax.

When debug logging is enabled, the following is logged in cs_server.log

```
DEBUG 2019-03-05 16:18:16.521 -0800 DDLNode - Set annotation on resource:
/shared/examples/myviewWithProps
Annotation: this view is altered using DSL
DEBUG 2019-03-05 16:18:16.522 -0800 DDLNode - "Data View
'/shared/examples/myviewWithProps' altered with sql 'SELECT OrderID,
CompanyName
FROM      /shared/examples/ViewOrder'
```

To turn on debug logging for these commands, please set the following in conf/server/log4j.properties

```
log4j.logger.com.compositesw.server.qe.physical.ddl=DEBUG
```

SQL Script Procedures

Using DSL & REST APIs, SQL Script procedures can be created, altered and deleted. SQL Script Procedure and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the following sample DSL syntax, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to Create, Update and Delete sql script procedures. The word "script" is used mainly in the DSL syntax because the subtype of a procedure is "script".

```
DROP SCRIPT (IF EXISTS)? /path/name
```

```
CREATE SCRIPT (IF NOT EXISTS)? /path/name
DEFINE AS  script
(SET PROPERTIES {"name":{value}})?
(SET ANNOTATION 'this is a script created using DSL api' | NULL)?
```

```
ALTER SCRIPT /path/name (RELOCATE AS /path/newName)?
(RELOCATE AS /newPath/newName)?
(DEFINE AS  script)?
(SET PROPERTIES {"name":{value}})?
(SET ANNOTATION 'this is a script altered using DSL api' | NULL)?
```

```
SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        PROPERTY_NAME = 'propName' AND

        METADATA_ID =
            (SELECT PROCEDURE_ID FROM model.ALL_PROCEDURES

            WHERE PROCEDURE_NAME = 'name' AND       //name of script

            PARENT_PATH = '/path');                 //parent path of the
script
```

Considerations

Listed below are some points to consider while working with SQL Script Procedures:

- Properties and annotation are optional when creating a sql script procedure.
- “script” is an attribute of type String. The script mentioned while creating or altering a sql script procedure is saved into the attribute "script" in the script metadata.
- Annotation, if any specified on the script is saved into the script metadata as well.
- The created script can be viewed from the Studio as well. Multiple statements can be included in the script (for example, DROP TABLE can be used before a CREATE TABLE.)
- When a script is altered, the definition script may or may not be provided. When a definition script is provided, which is not the same as the existing

definition script, the new script is set on the script metadata. Properties and annotation are optional as well, with the alter syntax.

- Script may be renamed or relocated, using the "RELOCATE AS" syntax.
- When a script is created or altered, the model.ALL_RESOURCE_PROPERTIES table is updated with the properties of the script. These are the attributes that are set on a script that is created using the DSL api.
- Assuming a table exists, an INSERT script can be used in the create sql script syntax. If the table in which records are inserted does not exist, then the script is created , however it will remain impacted till the table is created.
- To read the value of "script" property from model.ALL_RESOURCE_PROPERTIES table, "rest/execute/v1/actions/query/ invoke" can be used.

Examples

Create script with a CTAS script

```
curl -d
"[\"CREATE SCRIPT /shared/examples/ctasScript DEFINE AS PROCEDURE
ctasScript()
            BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/OrdersTable as
                    select OrderId, ProductID, Discount, OrderDate,
CompanyName, CustomerContactFirstName,
                           CustomerContactLastName, CustomerContactPhone
                    FROM /shared/examples/ViewOrder;
            END\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create script with CTAS script and annotation

```
curl -d
"[\"CREATE SCRIPT /shared/examples/ctasScript DEFINE AS PROCEDURE
ctasScript()
            BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/OrdersTable as
              select OrderId, ProductID, Discount, OrderDate, CompanyName,
CustomerContactFirstName,
                        CustomerContactLastName, CustomerContactPhone
                FROM /shared/examples/ViewOrder;
            END
            SET ANNOTATION 'this script is created using DSL'\"]"
```

```
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Create Script with an INSERT script

Assuming a table exists, an INSERT script can be used in the create sql script syntax. If the table in which records are inserted does not exist, then the script is created , however it will remain impacted till the table is created.

```
curl -d
"[\"CREATE SCRIPT /shared/examples/sqlScriptProc DEFINE AS PROCEDURE
sqlScriptProc()
            BEGIN INSERT INTO /shared/examples/ds_inventory/tutorial/T
                ( SELECT OrderId, ProductID, Discount, OrderDate,
CompanyName, CustomerContactFirstName,
                        CustomerContactLastName, CustomerContactPhone
                FROM /shared/examples/ViewOrder);
            END\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Delete script

```
curl -d "[\"DROP SCRIPT /users/composite/admin/dslscript\"]" -u
"admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Alter script

```
curl -d "[\"ALTER SCRIPT /shared/examples/ctasScript DEFINE AS PROCEDURE
ctasScript()
            BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/OrdersTable as
                    select OrderId, ProductID, OrderDate,
CustomerContactFirstName, CustomerContactLastName
                    FROM /shared/examples/ViewOrder;
            END
            SET ANNOTATION 'this SCRIPT is altered using DSL'\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Relocate/Rename Script

```
//script relocate -- rename only
curl -d "[\"ALTER SCRIPT /shared/examples/ctasScript RELOCATE AS
/shared/examples/dslscript\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"

//script relocate -- path only
```

```
curl -d "[\"ALTER SCRIPT /shared/examples/dslscript RELOCATE AS
/users/composite/admin/dslscript\"]" -u "admin:admin" -X POST
    "http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"

//script relocate -- path and name
curl -d "[\"ALTER SCRIPT /shared/examples/ctasScript RELOCATE AS
/users/composite/admin/dslscript\"]"
    -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Get script of a procedure

```
curl -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json"
    -d "{\"query\":\"SELECT * FROM model.ALL_RESOURCE_PROPERTIES
            WHERE property_name = 'script' AND
                  metadata_id = (
                    SELECT PROCEDURE_ID FROM model.ALL_PROCEDURES WHERE
PROCEDURE_NAME = 'ctasScript' AND parent_path = '/shared/examples')\",
        \"standardSQL\":true}"

[[10132,"Script","java.lang.String","PROCEDURE ctasScript()\nBEGIN\n
CREATE TABLE \/shared\/examples\/ds_inventory\/tutorial\/OrdersTable
AS\nSELECT OrderId, ProductID, OrderDate, CustomerContactFirstName,
CustomerContactLastName\nFROM      \/shared\/examples\/ViewOrder\n;\nEND"]]
```

Script containing multiple DSL statements

```
CREATE SCRIPT /shared/examples/pubdbScr3 DEFINE AS PROCEDURE pubdbScr3()
    BEGIN
        CREATE VIRTUAL DATABASE 'PUB5';
        CREATE VIRTUAL TABLE /services/databases/PUB5/link2 SET TARGET
/shared/examples/ds_orders/tutorial/view1;
    END
```

System Tables

The following system tables under /services/databases/system/model directory, store the information about datasources:

ALL_PROCEDURES

The system table model.ALL_PROCEDURES table is updated with procedure information and has the following columns.

This table is similar to system.ALL_PROCEDURES table, however, this table contains information about both published and non-published procedures.

Column name	TDV JDBC Data Type	Nullable	Description
PROCEDURE_ID:	INTEGER		Identifier of the procedure; Primary key.
PROCEDURE_NAME:	VARCHAR		Name of the procedure
PROCEDURE_TYPE:	SMALLINT		Type of the procedure 1 indicates procedure returns no result 2 indicates procedure returns result
SCHEMA_ID:	INTEGER	Yes	See SCHEMA_ID in Table: model.ALL_SCHEMAS
SCHEMA_NAME:	VARCHAR	Yes	See SCHEMA_NAME in Table: model.ALL_SCHEMAS
CATALOG_ID:	INTEGER	Yes	See CATALOG_ID in Table: model.ALL_CATALOGS
CATALOG_NAME:	VARCHAR	Yes	See CATALOG_ID in Table: model.ALL_CATALOGS
DATASOURCE_ID:	INTEGER		See DATASOURCE_ID in Table: model.ALL_DATASOURCES
DATASOURCE_NAME:	VARCHAR		See DATASOURCE_NAME in Table: model.ALL_DATASOURCES
GUID:	VARCHAR		128 bit unique identifier
ANNOTATION:	VARCHAR		Annotation for the procedure
OWNER_ID:	INTEGER		Identifier of the person who created/owns the procedure. Same as USER_ID in Table: ALL_USERS

Column name	TDV JDBC Data Type	Nullable	Description
OWNER:	VARCHAR		User name of the person who created/owns the procedureSame as USERNAME in Table: ALL_USERS
PARENT_PATH:	VARCHAR		Path to the parent container.
IS_AUTO_GENERATED	VARCHAR		Indicates if the parent folder has an attribute called 'webui_model'. If attribute is present, the value is 'YES' else 'NO'
PROCEDURE_CREATOR_ID:	INTEGER		Identifier of the user who created this procedure. Same as USER_ID in Table: ALL_USERS
PROCEDURE_CREATION_TIMESTAMP:	BIGINT		Timestamp when the procedure was created.
PROCEDURE_MODIFIER_ID:	INTEGER		Identifier of the user who last modified this procedure. Same as USER_ID in Table: ALL_USERS
PROCEDURE_MODIFICATION_TIMESTAMP:	BIGINT		Timestamp of the last modification of this procedure.
IMPACT_MESSAGE	VARCHAR		Impact message indicating errors.

Logging

To turn on debug logging, set the following in conf/server/log4j.properties

```
log4j.logger.com.compositesw.server.qe.physical.ddl=DEBUG
```

Create Script

When a script "/shared/examples/ctasScript" is created with attribute "script", a DEBUG message is logged like below in the cs_server.log file:

```
DEBUG 2019-04-09 11:54:51.992 -0700 DDLNode - About to execute: CREATE
SCRIPT /shared/examples/ctasScript
 DEFINE AS PROCEDURE ctasScript()
BEGIN
    CREATE TABLE /shared/examples/ds_inventory/tutorial/OrdersTable AS
SELECT OrderId, ProductID, Discount, OrderDate, CompanyName,
CustomerContactFirstName, CustomerContactLastName, CustomerContactPhone
FROM      /shared/examples/ViewOrder;
```

"IF NOT EXISTS" can be specified in the CREATE SCRIPT command optionally. When script being created does not exist, its created. If resource already exists, no error is thrown when IF NOT EXISTS keywords are specified. Note that the procedure name mentioned in the script and the name of the script must be the same, else the script creation will fail and will throw an error.

Alter Script

The script is marked impacted when errors are found during compilation. During an alter, the following DEBUG message is logged in the cs_server.log file, if debugging is enabled

```
DEBUG 2019-04-09 12:17:57.978 -0700 DDLNode - About to execute: ALTER
SCRIPT /shared/examples/ctasScript
 DEFINE AS PROCEDURE ctasScript()
BEGIN
    CREATE TABLE /shared/examples/ds_inventory/tutorial/OrdersTable AS
SELECT OrderId, ProductID, OrderDate, CustomerContactFirstName,
CustomerContactLastName
FROM      /shared/examples/ViewOrder
;
END
    SET ANNOTATION 'this SCRIPT is altered using DSL'
```

Delete Script

During a delete operation, the following DEBUG message is logged.

```
DEBUG 2019-04-09 13:14:38.300 -0700 DDLNode - About to execute: DROP SCRIPT
/shared/examples/ctasScript
```

"IF EXISTS" can be specified optionally with the DROP SCRIPT command. When script being dropped exists, it is dropped. When script being dropped does not exist, and "IF EXISTS" keywords are used, no error is thrown.

Folders

Using DSL & REST APIs, Folders can be created, altered or deleted. Folder and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the following DSL syntax, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to Create, Update and Delete folders.

```
DROP FOLDER (IF EXISTS)? /path/name
```

```
CREATE FOLDER (IF NOT EXISTS)? '/parentPath/folderName'

(SET PROPERTIES {json_object})?

(SET ANNOTATION 'this is a FOLDER created using DSL api' | NULL)?
```

```
ALTER FOLDER '/parentPath/folderName' (RELOCATE TO '/newParent/newName')?

(SET PROPERTIES {json_object})?

(SET ANNOTATION 'this is a FOLDER altered using DSL api' | NULL)?
```

```
SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        PROPERTY_NAME = 'propName' AND

        METADATA_ID =
            (SELECT FOLDER_ID FROM model.ALL_FOLDERS

            WHERE FOLDER_NAME = 'name' AND //name of folder

            PARENT_PATH = '/path'); //parent path of the folder
```

Considerations

Listed below are some points to consider while working with Folders:

- "SET PROPERTIES" syntax is optional for both Create and Alter syntax.

- Annotation, if any specified on the folder is saved into the folder metadata as well.
- Use the "CREATE FOLDER IF NOT EXISTS" syntax to create a new folder to avoid any error messages to be displayed when the folder already exists.
- By default Read, Write and Grant privileges are provided to the owner and all privileges to the admin users or users in admin group.
- Alter folder syntax supports both relocate and rename operations of folders. When a folder is relocated to another parent path, and if the parent path does not exist, it is created automatically before folder is created. When a folder is relocated to another path that already exists, an error is thrown. Note that there is no IF NOT EXISTS syntax on "alter folder" command.
- "RELOCATE TO" syntax is used for relocating the folders as well as renaming folders.
- The created folder can be viewed from the Studio as well.
- When a folder is created or altered, the model.ALL_RESOURCE_PROPERTIES table is updated with the properties of the folder like "annotation".

Examples

Create Folder

```
curl -d
"[\"CREATE FOLDER /shared/examples/folder2
    SET ANNOTATION 'this folder is created using DSL'
\"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Delete folder

```
curl -d "[\"DROP FOLDER /shared/examples/folder2\"]" -u "admin:admin" -X
POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Alter Folder

```
curl -d "[\"ALTER FOLDER /shared/examples/folder2
            SET ANNOTATION 'this folder is altered using DSL'
        \"]"
-u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

Read Property creationDate from a Folder

```
SELECT * FROM  /services/databases/system/model/ALL_RESOURCE_PROPERTIES
where
        PROPERTY_NAME = 'creationDate' AND

        METADATA_ID =
            (SELECT FOLDER_ID FROM
/services/databases/system/model/ALL_FOLDERS

            WHERE FOLDER_NAME = 'dslFolder' AND

            PARENT_PATH = '/users/composite/admin')
        ;
```

System Tables

ALL_FOLDERS

The model.ALL_FOLDERS table has the following columns:

Column name	TDV JDBC Data Type	Nullable	Description
FOLDER_ID:	INTEGER		Identifier of the folder.
FOLDER_NAME	VARCHAR		Name of the folder
GUID	CHAR		128 bit unique identifier.
ANNOTATION	VARCHAR		Annotation for the folder
OWNER_ID	INTEGER		Identifier of the person who created/owns the folder. Same as USER_ID in Table: ALL_USERS
OWNER	VARCHAR		User name of the person who created/owns the folder. Same as USERNAME in Table: ALL_USERS
PARENT_PATH	VARCHAR		Path to the parent container.
FOLDER_CREATION_TIME STAMP	BIGINT		Timestamp when the folder was created.

Column name	TDV JDBC Data Type	Nullable	Description
FOLDER_MODIFICATION_TIMESTAMP	BIGINT		Timestamp of the last modification of this folder.
FOLDER_CREATOR_ID	INTEGER		Identifier of the user who created this folder. Same as USER_ID in Table: ALL_USERS
FOLDER_MODIFIER_ID	INTEGER		Identifier of the user who last modified this folder. Same as USER_ID in Table: ALL_USERS

This table is similar to system.ALL_PUBLISHED_FOLDERS table that represents only the folders created under /services/webservices folder.

The model.ALL_FOLDERS table does not include the folders starting with system paths like /policy "/system", "/deployment", "/scratch", "/security", "/packages", "/vcs_sources" or "/services/databases". This table does not include the folders owned by 'system' user.

The existing ALL_PUBLISHED_FOLDERS will continue to show only any folders that may be present under /services/webservices folder.

Logging

During create or alter, a DEBUG message is logged like below :

```
DEBUG 2019-06-11 14:33:11.174 -0700 DDLNode - About to execute: CREATE
FOLDER /shared/examples/folder2

SET ANNOTATION 'this folder is created using DSL'
```

Virtual Databases

Using DSL & REST APIs, Virtual Databases can be created, altered or deleted. Virtual databases refer to the published databases under /services/databases. Virtual database and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the DSL syntax mentioned below,, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to create, upodate and delete virtual databases:

```
DROP VIRTUAL DATABASE (IF EXISTS)? 'name'

_____________________________________________________________

CREATE VIRTUAL DATABASE (IF NOT EXISTS)? 'name'

(SET ANNOTATION 'this is a virtual db created using DSL api' | NULL)?

_____________________________________________________________

ALTER VIRTUAL DATABASE 'name' (RENAME AS 'newName')?

(SET ANNOTATION 'this is a virtual db created using DSL api' | NULL)?

_____________________________________________________________

SELECT * FROM model.ALL_DATASOURCES WHERE PARENT_PATH =
'/services/databases' //to view all published virtual relational
datasources

_____________________________________________________________

//to get the resource properties

SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        METADATA_ID =
            (SELECT DATASOURCE_ID FROM model.ALL_DATASOURCES

            WHERE DATASOURCE_NAME = 'name' AND
//name of virtual db

            PARENT_PATH = '/services/databases');
//parent path of the folder

_____________________________________________________________

//to get database attribute definitions for Composite Database.

SELECT * FROM model.SYS_DATASOURCE_ATTRIBUTE_DEFS WHERE adapter_name =
'COMPOSITE_DATABASE'
```

Considerations

Listed below are some points to consider while working with Virtual Databases:

- During creation of virtual database, the adapter name used is "COMPOSITE_DATABASE".
- The DATASOURCE_TYPE of a virtual database is "VirtualRelational".
- Use the "CREATE VIRTUAL DATABASE IF NOT EXISTS" syntax to avoid any duplicate error messages.
- For annotation QUOTED_STRING, an explicit NULL keyword or a quoted string 'NULL' can be specified. This is to allow annotation to be unset or set to null explicitly.
- The virtual databases are created in the "/services/databases" area.
- Altering a virtual database can alter the annotation or rename the virtual database. Use the "RENAME AS" syntax to rename a published database.
- IF EXISTS" syntax in DROP to avoid any error messages, in case the resource is not found.

Examples

Create a Virtual Database

```
curl -d "[\"CREATE VIRTUAL DATABASE 'PUB1' SET ANNOTATION 'this is a virtual
db'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Rename a Virtual database

```
curl -d "[\"ALTER VIRTUAL DATABASE 'PUB1' RENAME AS 'renamedpub'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Edit annotation

```
curl -d "[\"ALTER VIRTUAL DATABASE 'PUB1' SET ANNOTATION 'altered virtual
db'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Delete if exists /services/databases/renamedpub

```
curl -d "[\"DROP VIRTUAL DATABASE IF EXISTS 'renamedpub'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Get Virtual DB properties

```
SELECT * FROM /services/databases/system/model/ALL_RESOURCE_PROPERTIES
WHERE  metadata_id=
    (select datasource_id from
/services/databases/system/model/ALL_DATASOURCES
        where datasource_name = 'publishedDB' and parent_path =
'/services/databases')
```

System Tables

Virtual databases are listed in

- /services/databases/system/model/ALL_DATASOURCES
- /services/databases/system/ALL_DATASOURCES.

The model schema version has a few extra columns such as IS_PUBLISHED, ADAPTER_NAME and ADAPTER_TYPE_CATEGORY).

The properties are saved in

- /services/databases/system/model/ALL_RESOURCE_PROPERTIES

Virtual Tables and Procedures

Using DSL & REST APIs, Virtual Tables and Procedures can be created, altered or deleted. Virtual database and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the DSL syntax mentioned below,, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to CUD virtual tables or procedures:

```
DROP VIRTUAL TABLE|PROCEDURE (IF EXISTS)? /path/name
```

```
_____________________________________________________________________

CREATE VIRTUAL TABLE|PROCEDURE  (IF NOT EXISTS)? /path/name

SET TARGET /path/name

(SET ANNOTATION 'this is a link created using DSL api' | NULL)?

_____________________________________________________________________

ALTER VIRTUAL TABLE|PROCEDURE  /path/name (RELOCATE TO /path/newName)?

(SET TARGET /path/name )?

(SET ANNOTATION 'this is a link created using DSL api' | NULL)?

_____________________________________________________________________

SELECT * FROM model.ALL_TABLES WHERE PARENT_PATH =
'/services/databases/publishedDB'        //to view all published virtual
tables

SELECT * FROM model.ALL_PROCEDURES WHERE PARENT_PATH =
'/services/databases/publishedDB'    //to view all published virtual
procedures

_____________________________________________________________________

//to get the resource properties

SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        METADATA_ID =
            (SELECT TABLE_ID FROM model.ALL_TABLES

          WHERE TABLE_NAME = 'name' AND
//name of virtual table
        PARENT_PATH = '/services/databases/publishedDB');
//parent path of the published resource
```

Considerations

Listed below are some points to consider while working with Virtual Tables and Procedures:

- Virtual Table / Procedure DSL can be executed using:

  For Create/Alter/Delete: http://localhost:9400/rest/execute/v1/actions/ dsl/invoke

  For read: http://localhost:9400/rest/execute/v1/actions/query/invoke

- During creation of virtual table or procedure, a target is to be specified which refers to the base table or procedure in non-published area. "target" is mandatory in a create command. Specifying target is optional in an alter command and when specified during alter, its used to change the target.
- Validation is performed to check if the specified target is of type TABLE or PROCEDURE specified in the DSL command.
- Use the "CREATE VIRTUAL TABLE | PROCEDURE IF NOT EXISTS" syntax to avoid any error messages if the virtual table or procedure already exists.
- For annotation QUOTED_STRING, an explicit NULL keyword or simply 'NULL' can be specified. This is to allow annotation to be unset or set to null explicitly.
- Altering a virtual table or procedure can also alter the annotation or rename/relocate the virtual table or procedure.
- Drop virtual table or procedure includes an "IF EXISTS" syntax. When this is specified, error will not be displayed for non-existent resources.

Examples

Create a Virtual Table

```
curl -d "[\"CREATE VIRTUAL TABLE /services/databases/PUB3/link1 SET TARGET
/shared/examples/ds_orders/tutorial/customers SET ANNOTATION 'this is a
published table'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Create a Virtual Procedure

```
curl -d "[\"CREATE VIRTUAL PROCEDURE /services/databases/PUB3/link2 SET
TARGET /shared/examples/LookupProduct SET ANNOTATION 'this is a published
procedure'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Rename & Relocate a Virtual table

```
//only rename
curl -d "[\"ALTER VIRTUAL TABLE /services/databases/PUB3/link1 RELOCATE TO
/services/databases/PUB3/renamedlink1 SET ANNOTATION 'altered virtual
table'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

```
//Both rename and relocate
curl -d "[\"ALTER VIRTUAL TABLE /services/databases/PUB3/link1 RELOCATE TO
/services/databases/publishedDB/renamedlink1 SET ANNOTATION 'altered
virtual table'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Unset annotation

```
curl -d "[\"ALTER VIRTUAL TABLE /services/databases/PUB3/link1 SET
ANNOTATION NULL\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Alter Virtual Table Target

```
curl -d "[\"ALTER VIRTUAL TABLE /services/databases/PUB3/link1 SET TARGET
/shared/examples/ds_orders/tutorial/employees SET ANNOTATION 'altered
virtual procedure with table target'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Delete Virtual Table

```
curl -d "[\"DROP VIRTUAL TABLE /services/databases/PUB3/link1\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Delete Virtual Procedure IF EXISTS

```
curl -d "[\"DROP VIRTUAL PROCEDURE IF EXISTS
/services/databases/PUB3/link2\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Get Virtual Table Properties

```
curl -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json"
    -d "{\"query\":\"SELECT * FROM model.ALL_RESOURCE_PROPERTIES props
                WHERE metadata_id = (SELECT table_id FROM model.ALL_TABLES
where table_name = 'link1' and
parent_path='/services/databases/publishedDB')\",
        \"standardSQL\":true}"
```

System Tables

Virtual tables are listed in:

- /services/databases/system/model/ALL_TABLES
- /services/databases/system/ALL_TABLES.

Virtual procedures are listed in:

- /services/databases/system/model/ALL_PROCEDURES
- /services/databases/system/ALL_PROCEDURES.

“The model schema version has extra columns such as IS_AUTO_GENERATED and IMPACT_MESSAGE

The properties are saved in /services/databases/system/model/ALL_RESOURCE_PROPERTIES table.

Logging

To turn on debug logging for these commands, set the following in conf/server/log4j.properties

log4j.logger.com.compositesw.server.qe.physical.ddl=DEBUG

A DEBUG message is logged in the cs_server.log when debugging is enabled.

Virtual Schemas

Using DSL & REST APIs, Virtual Schemas can be created, altered and deleted. Virtual schemas refer to the schemas in the published area under /services/databases. Virtual schema and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using the DSL syntax mentioned below,, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to create, update and delete virtual schemas.

```
DROP VIRTUAL SCHEMA (IF EXISTS)? /path/name
```

```
CREATE VIRTUAL SCHEMA (IF NOT EXISTS)? /path/name
```

```
(SET ANNOTATION 'this is a virtual schema created using DSL api' | NULL)?
```

```
ALTER VIRTUAL SCHEMA /path/name (RELOCATE TO /path/newName)?

(SET ANNOTATION 'this is a virtual schema created using DSL api' | NULL)?
```

```
SELECT * FROM model.ALL_SCHEMAS WHERE PARENT_PATH =
'/services/databases/publishedDB'       //to view published virtual schemas
```

```
//to get the resource properties

SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        METADATA_ID =
            (SELECT SCHEMA_ID FROM model.ALL_SCHEMAS

          WHERE SCHEMA_NAME = 'name' AND
//name of virtual schema
        PARENT_PATH = '/services/databases/publishedDB');
//parent path of the published schema
```

Considerations

Listed below are some points to consider while working with Virtual Schemas:

- Virtual Schema DSL can be executed using

  for create/alter/delete - http://localhost:9400/rest/execute/v1/actions/dsl/invoke

  for read - http://localhost:9400/rest/execute/v1/actions/query/invoke

- For annotation QUOTED_STRING, an explicit NULL keyword or a quoted string 'NULL' can be specified. This is to allow annotation to be unset or set to null explicitly.
- Altering a virtual schema can alter the annotation or rename/relocate the virtual schema.

Examples

Create a Virtual Schema

```
curl -d "[\"CREATE VIRTUAL SCHEMA /services/databases/PUB3/sch1 SET
ANNOTATION 'this is a published schema'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Create a Virtual Schema If not exists

```
curl -d "[\"CREATE VIRTUAL SCHEMA IF NOT EXISTS
/services/databases/PUB3/sch1 SET ANNOTATION 'this is a published
schema'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Rename & Relocate a Virtual schema

```
//only rename
curl -d "[\"ALTER VIRTUAL SCHEMA /services/databases/PUB3/sch1 RELOCATE TO
/services/databases/PUB3/sch2 SET ANNOTATION 'renamed virtual schema'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"

//Both rename and relocate
curl -d "[\"ALTER VIRTUAL SCHEMA /services/databases/PUB3/sch2 RELOCATE TO
/services/databases/publishedDB/sch3 SET ANNOTATION 'altered virtual
schema'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

System Tables

Virtual schemas are listed in

- /services/databases/system/model/ALL_SCHEMAS
- /services/databases/system/ALL_SCHEMAS.

The properties are saved in

- /services/databases/system/model/ALL_RESOURCE_PROPERTIES table.

Virtual Catalogs

Using DSL & REST APIs, Virtual Catalogs can be created, altered, deleted. Virtual catalogs refer to the catalogs in the published area under /services/databases. Virtual catalog and its properties can be read from system tables under /services/databases/system/model.

DSL Syntax

Using a sample DSL syntax mentioned below,, the "rest/execute/v1/actions/dsl/invoke" REST API can be executed to create, update and delete virtual catalogs:

```
DROP VIRTUAL CATALOG (IF EXISTS)? /path/name

_____________________________________________________________________

CREATE VIRTUAL CATALOG (IF NOT EXISTS)? /path/name

(SET ANNOTATION 'this is a virtual catalog created using DSL api' | NULL)?

_____________________________________________________________________

ALTER VIRTUAL CATALOG /path/name (RELOCATE TO /path/newName)?

(SET ANNOTATION 'this is a virtual catalog created using DSL api' | NULL)?

_____________________________________________________________________

SELECT * FROM model.ALL_CATALOGS WHERE PARENT_PATH =
'/services/databases/publishedDB'      //to view published virtual catalog

_____________________________________________________________________

//to get the resource properties

SELECT * FROM model.ALL_RESOURCE_PROPERTIES WHERE

        METADATA_ID =
            (SELECT CATALOG_ID FROM model.ALL_CATALOGS

         WHERE CATALOG_NAME = 'name' AND
//name of virtual catalog
       PARENT_PATH = '/services/databases/publishedDB');
//parent path of the published catalog
```

Considerations

Listed below are some points to consider while working with Virtual Catalogs:

- Virtual Catalog DSL can be executed using

  For create/alter/delete - http://localhost:9400/rest/execute/v1/actions/dsl/invoke

  For read - http://localhost:9400/rest/execute/v1/actions/query/invoke

- For annotation QUOTED_STRING, an explicit NULL keyword or a quoted string 'NULL' can be specified. This is to allow annotation to be unset or set to null explicitly.
- Altering a virtual catalog can alter the annotation or rename/relocate the virtual catalog.

Examples

Create a Virtual Catalog

```
curl -d "[\"CREATE VIRTUAL CATALOG /services/databases/PUB3/cat1 SET
ANNOTATION 'this is a published catalog'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Create a Virtual Catalog If not exists

```
curl -d "[\"CREATE VIRTUAL CATALOG IF NOT EXISTS
/services/databases/PUB3/cat1 SET ANNOTATION 'this is a published
catalog'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"
```

Rename & Relocate a Virtual catalog

```
//only rename
curl -d "[\"ALTER VIRTUAL CATALOG /services/databases/PUB3/cat1 RELOCATE TO
/services/databases/PUB3/cat2 SET ANNOTATION 'renamed virtual catalog'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
    -H "Content-Type:application/json"

//Both rename and relocate
curl -d "[\"ALTER VIRTUAL CATALOG /services/databases/PUB3/cat2 RELOCATE TO
/services/databases/publishedDB/cat3 SET ANNOTATION 'altered virtual
catalog'\"]"
    -u "admin:admin"
    -X POST "http://localhost:9400/rest/execute/v1/actions/dsl/invoke"
```

```
-H "Content-Type:application/json"
```

System Tables

Virtual catalogs are listed in:

- /services/databases/system/model/ALL_CATALOGS
- /services/databases/system/ALL_CATALOGS.

The properties are saved in

- /services/databases/system/model/ALL_RESOURCE_PROPERTIES

Logging

To turn on debug logging for these commands, please set the following in conf/server/log4j.properties

```
log4j.logger.com.compositesw.server.qe.physical.ddl=DEBUG
```

During these operations, a DEBUG message is logged in the cs_server.log when debugging is enabled.

DSL Support in SQL Scripts

Multiple DSL statements of the following types may be specified in any order of CREATE, ALTER, DELETE statements. If any errors are found, the IMPACT_MESSAGE will show a non-null error message. Once the errors are corrected and saved, the IMPACT_MESSAGE will become NULL again.

- Virtual Database
- Virtual Table/Procedure
- Virtual Schema
- Table

Examples

Script containing multiple DSL statements

```
CREATE SCRIPT /shared/examples/testscript DEFINE AS PROCEDURE pubdbScr3()
    BEGIN
        CREATE VIRTUAL DATABASE 'PUB5';
```

```
        CREATE VIRTUAL TABLE /services/databases/PUB5/link2 SET TARGET
/shared/examples/ds_orders/tutorial/view1;
    END
```

Script containing multiple DSL statements - Impacted

```
CREATE SCRIPT /shared/examples/testscript DEFINE AS PROCEDURE pubdbScr3()
    BEGIN
        CREATE VIRTUAL DATABASE 'PUB5';
        DROP VIRTUAL DATABASE 'PUB5';
        CREATE VIRTUAL TABLE /services/databases/PUB5/link2 SET TARGET
/shared/examples/ds_orders/tutorial/view1;
    END
```

The above example shows an impacted procedure because the virtual database 'PUB5' is being deleted before virtual table is created under 'PUB5'. Hence the IMPACT_MESSAGE of "testscript" procedure will show the error message.

REST API

REST APIs provide a different means of executing most of the tasks that can be accomplished in Studio.These tasks can be duplicated with the appropriate web service operation invocation, or sequence of operation invocations from the Admin API. This gives the administrators the flexibility to script common tasks rather than using the Studio or Manager GUI.

This chapter describes all the built-in REST APIs in the TDV Server.

TDV Server REST APIs

TDV provides a list of APIs to manage the resources. This section explains the all the operations that can be performed on the resources by using REST APIs for the following TDV features:

Catalog

Using the Catalog REST API, you can create, delete, update and read virtual (published) catalogs. The operations that can be performed on the virtual catalogs are:

- GET/catalog
- PUT/catalog
- POST/catalog
- DELETE/catalogs
- DELETE/catalog

GET/catalog

This API is used to get virtual catalog summary and/or detailed information.

Parameters

Name	Deacription	Parameter Type	Data Type
path	Virtual catalog path	query	string
summary	Fetch virtual catalog summary	query	boolean

Example to get summary of virtual catalog "/services/databases/publishedDB/cat1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual?path=%2Fservices%2F
databases%2FpublishedDB%2Fcat1&summary=true" -H
"Content-Type:application/json"
```

Example to get detailed information about a virtual catalog "/services/databases/publishedDB/cat1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual?path=%2Fservices%2F
databases%2FpublishedDB%2Fcat1" -H "Content-Type:application/json"
```

PUT/catalog

This API is used to update virtual catalogs.

Parameters

None.

Request Body

Example Value - Schema

```
[
{
"path": "string",
"annotation": "string",
"newPath": "string",
"ifNotExists": true
}
]
```

Example to update annotation and rename a published catalog "/services/databases/publishedDB/cat1 to /services/databases/publishedDB/cat2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/cat1\",
\"annotation\":\"Edited published catalog created using REST api\",
\"newPath\":\"/services/databases/publishedDB/cat2\"  }]"
```

Example to update annotation and relocate a published catalog "/services/databases/publishedDB/cat2 to /services/databases/publishedDB1/cat2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/cat2\",
\"annotation\":\"Edited published catalog created using REST api\",
\"newPath\":\"/services/databases/publishedDB1/cat2\"  }]"
```

Example to unset annotation of a published catalog "/services/databases/publishedDB/cat1"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/cat1\",
\"annotation\":\"null\"}]"
```

POST/catalog

This API is used to create virtual catalogs.

Parameters

None.

Request Body

Example Value - Schema

```
[
{
"path": "string",
"annotation": "string",
"newPath": "string",
"ifNotExists": true
}
]
```

Example to create virtual catalog "/services/databases/publishedDB/cat1"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/cat1\",
\"annotation\":\"This is a published catalog created using REST
api\" }]"
```

Example to create virtual catalog "/services/databases/publishedDB/cat1" with "ifNotExists" syntax

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/cat1\",
\"ifNotExists\":true, \"annotation\":\"This is a published catalog
created using REST api\" }]"
```

DELETE/catalogs

This API is used to delete virtual catalogs.

Parameters

Name	Deacription	Parameter Type	Data Type
ifExists	If the catalogs exist	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to delete virtual catalogs "/services/databases/publishedDB/cat1" and "/services/databases/publishedDB/cat2"

Sample CURL Invocation

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual" -H
"Content-Type:application/ json" -d
"[\"/services/databases/publishedDB/cat1\", \"/services/databases/
publishedDB/cat2\"]"
```

Sample CURL Invocation with ifExists

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual?ifExists=true" -H
"Content-Type:application/json" -d
"[\"/shared/examples/publishedDB/cat2\"]"
```

DELETE/catalog

This API is used to delete a specific virtual catalog.

Parameters

Name	Deacription	Parameter Type	Data Type
ifExists	If the catalogs exist	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to delete virtual catalog

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual/{catalogPath}?ifExi
sts=true"  -H "Content-Type: application/json"
```

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/catalog/v1/virtual/%2Fservices%2Fdatab
ases%2Fpubdb%2Fcatalog1?ifExists=true" -H
"Content-Type:application/json"
```

Column-Based Security

Column-Based Security policies can be assigned to/deleted/updated from the TDV resources. The operations that can be performed on the resources are:

- GET /assignments
- POST /assignments
- PUT /assignments
- DELETE /assignments
- GET /enable
- PUT /enable
- GET /policies
- POST /policies
- PUT /policies
- DELETE /policies
- GET /policyDataTypeMap
- GET /policyDataTypes
- GET /ruleDataTypeMap

GET /assignments

This API is used to get cbs assignments.

Parameters:

Name	Value	Parameter Type	Data Type
policyPath		query	string
resourcePath		query	string
resourceType		query	string
columnName		query	string

Example to Get all cbs assignments:

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments"
```

Example to Get cbs policy “policy/cbs/cbs1” assignments

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?policyPath=/policy/
cbs/cbs1"
```

Example to Get resource “/shared/examples/ViewOrder” assignments

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?resourcePath=/share
d/examples/ViewOrder&resourceType=TABLE"
```

Example to Get cbs policy“/policy/cbs/cbs1” assignment to resource “/shared/examples/ViewOrder” assignments

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?policyPath=/policy/
cbs/cbs1&resourcePath=/shared/examples/ViewOrder&resourceType=TABL
E"
```

Example to Get cbs policy“/policy/cbs/cbs1” assignment to resource “/shared/examples/ViewOrder” column “companyName” assignments

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?policyPath=/policy/
cbs/cbs1&resourcePath=/shared/examples/ViewOrder&resourceType=TABL
E&columnName=companyName"
```

POST /assignments

This API is used to add cbs assignments.

Parameters:

None

Request Body

Example Value: Schema

```
[
{
"columnName": "string",
"paramMap": {
"additionalProp1": "string",
"additionalProp2": "string",
"additionalProp3": "string"
},
"resourceType": "string",
"resourcePath": "string",
"policyPath": "string"
}
]
```

Example

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments" -H
"Content-Type:application/json" -d "[{ \"policyPath\":
\"/policy/cbs/cbs1\",\"resourcePath\":\"/shared/examples/ViewOrder
\",
\"resourceType\":\"TABLE\",\"columnName\":\"CustomerContactPhone\"
,\"paramMap\":{}}]"
```

PUT /assignments

This API is used to update cbs assignments.

Parameters:

None

Request Body

Example Value: Schema

```
[
{
"columnName": "string",
"paramMap": {
"additionalProp1": "string",
"additionalProp2": "string",
"additionalProp3": "string"
},
"resourceType": "string",
"resourcePath": "string",
"policyPath": "string"
}
]
```

Example

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments" -H
"Content-Type:application/json" -d "[{ \"policyPath\":
\"/policy/cbs/cbs1\",\"resourcePath\":\"/shared/examples/ViewOrder
\",
\"resourceType\":\"TABLE\",\"columnName\":\"CustomerContactPhone\"
,\"paramMap\":{}}]"
```

DELETE /assignments

This API is used to delete cbs assignments.

Parameters:

Name	Description	Parameter Type	Data Type
resourcePath		query	String
resourceType		query	String
columnName		query	String

Example to delete cbs assignments associated with view

"/shared/examples/ViewOrder" column "CustomerContactPhone"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?resourcePath=/share
d/examples/ViewOrder&resourceType=TABLE&columnName=CustomerContact
Phone"
```

Example to delete cbs assignments associate with view "/shared/examples/ViewOrder"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments?resourcePath=/share
d/examples/ViewOrder&resourceType=TABLE"
```

Example to delete all cbs assignments

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/cbs/v1/assignments"
```

GET /enable

This API is used to Get cbs status.

Parameters:

None

Example

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/enable"
```

PUT /enable

This API is used to enable or disable cbs.

Parameters:

Name	Description	Parameter Type	Data Type
body		body	integer

Example to enable cbs

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/cbs/v1/enable" -H
"Content-Type:application/json" -d "1"
```

Example to disable cbs

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/cbs/v1/enable" -H
"Content-Type:application/json" -d "0"
```

GET /policies

This API is used to get the cbs policies.

Parameters:

Name	Description	Parameter Type	Data Type
policyPath		query	string

Example to get all cbs policies

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies"
```

Example to get cbs policy "/policy/cbs/cbs1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies?policyPath=/policy/cbs
/cbs1"
```

POST /policies

This API is used to create cbs policies.

Parameters:

None

Request Body

Example Value - Schema

```
[
{
"path": "string",
"dataType": "string",
"maskingRules": [
{
"domainName": "string",
"userGroupName": "string",
"isGroup": true,
"isDefaultRule": true,
"ruleType": "string",
"selectableString": "string"
}
],
"annotation": "string",
"parameters": [
"string"
],
"newPath": "string",
"isEnabled": true
}
]
```

Example to create cbs policy "/policy/cbs/cbs1"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies" -H
"Content-Type:application/json" -d
"[{\"path\":\"/policy/cbs/cbs1\",\"dataType\":\"integer\",
\"maskingRules\":[{\"isDefaultRule\": \"true\", \"ruleType\":
\"PASS_THROUGH\",\"selectableString\":
\"\"}],\"isEnabled\":\"true\" ,\"annotation\":\"the 1st cbs\" }]"
```

PUT /policies

This API is used to update cbs policies.

Parameters:

None

Request Body

Example Value - Schema

```
[
{
"path": "string",
"dataType": "string",
"maskingRules": [
{
"domainName": "string",
"userGroupName": "string",
"isGroup": true,
"isDefaultRule": true,
"ruleType": "string",
"selectableString": "string"
}
],
"annotation": "string",
"parameters": [
"string"
],
"newPath": "string",
"isEnabled": true
}
]
```

Example to update cbs policy "/policy/cbs/cbs1"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies" -H
"Content-Type:application/json" -d
"[{\"path\":\"/policy/cbs/cbs1\",\"dataType\":\"integer\",
\"maskingRules\":[{\"isDefaultRule\": \"true\", \"ruleType\":
\"PASS_THROUGH\",\"selectableString\":
\"\"}],\"isEnabled\":\"true\" ,\"annotation\":\"update the 1st
cbs\" }]"
```

Example to rename cbs policy "/policy/cbs/cbs1" to "/policy/cbs/cbs2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies" -H
"Content-Type:application/json" -d
"[{\"path\":\"/policy/cbs/cbs1\",\"newPath\":\"/policy/cbs/cbs2\",
\"dataType\":\"integer\",  \"maskingRules\":[{\"isDefaultRule\":
\"true\", \"ruleType\": \"PASS_THROUGH\",\"selectableString\":
\"\"}],\"isEnabled\":\"true\" ,\"annotation\":\"update the 1st
cbs\" }]"
```

DELETE /policies

This API is used to delete cbs policies.

Parameters:

Name	Description	Parameter Type	Data Type
policyPath	cbs policy path	query	string

Example to delete cbs policy "/policy/cbs/cbs1"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies?policyPath=/policy/cbs
/cbs1"
```

Example to delete all cbs policies

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/cbs/v1/policies"
```

GET /policyDataTypeMap

This API is used to get compatible policy and column data type map.

Parameters:

None

Example to get compatible policy and column data type map

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/policyDataTypeMap"
```

GET /policyDataTypes

This API is used to get supported cbs rule types.

Parameters:

None

Example to get supported cbs rule types

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/policyDataTypes"
```

GET /ruleDataTypeMap

This API is used to get compatible rule and policy data type map.

Parameters:

None

Example to get compatible rule and policy data type map

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/cbs/v1/ruleDataTypeMap"
```

Datasource

The datasource API provides the ability to create, alter, delete and read a relational, file-delimited, MS Excel (non ODBC) or MS Excel datasources in the non-published area and virtual (published) databases. The following operations can be performed:

GET/datasource

PUT/datasource

POST/datasource

DELETE/datasource

GET/datasource/adapter/definitions

GET/datasource/virtual

PUT/datasource/virtual

POST/datasource/virtual

DELETE/datasource/virtual

DELETE/datasource/virtual/dsName

GET/datasource/virtual/adapter/definitions

GET/datasource

This API is used to get datasource summary and/or detailed information.

Parameters:

Name	Description	Parameter Type	Data Type
path	datasource path	query	string
summary	fetch datasource summary	query	boolean

Example to get datasource summary "/shared/examples/ds1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1?path=%2Fshared%2Fexample
s%2Fds_orders&summary=true" -H "Content-Type:application/json"
```

Example to get detailed information of datasource "/shared/examples/ds1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1?path=%2Fshared%2Fexample
s%2Fds_orders" -H "Content-Type:application/json"
```

PUT/datasource

This API is used to update datasources.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"adapterName": "string",
"nativeProperties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
```

```
"newName": "string",
"ifNotExists": true
}
]
```

Example to rename and Relocate a relational datasource

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d "[{\"parentPath\":\"/shared\",
\"name\":\"ds1\", \"adapterName\":\"PostgreSQL 9.1\",
\"newPath\":\"/shared/examples/dsnew\"  }]"
```

Example to update annotation and database name of a relational datasource "/shared/examples/ds1"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"ds1\",
\"adapterName\":\"PostgreSQL 9.1\", \"annotation\":\"Edited
annotation of a postgres datasource created using REST api\",
\"nativeProperties\":{\"urlDatabaseName\":\"inventory\"} }]"
```

Example to update annotation and delimiter of a file-delimited datasource "/shared/examples/csv_ds"

Sample CURL Invocation of a file-delimited datasource :

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"csv_ds\",
\"adapterName\":\"File-Delimited\", \"annotation\":\"Edited
annotation of a csv datasource created using REST api\",
\"nativeProperties\":{\"delimiter\":\":\"} }]"
```

Example to update annotation and data range of a MS excel (non-ODBC) datasource "/shared/examples/excel_ds"

Sample CURL Invocation of a MS excel (non-ODBC) datasource :

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"excel_ds\",
```

```
\"adapterName\":\"Microsoft Excel (non-ODBC)\",
\"annotation\":\"Edited annotation of a MS excel datasource created
using REST api\", \"nativeProperties\":{\"dataRange\":\"A2\"} }]"
```

Example to update annotation and dsn of a MS excel (ODBC) datasource "/shared/examples/excel_ds"

Sample CURL Invocation of a MS excel (ODBC) datasource :

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"excel_ds\",
\"adapterName\":\"Microsoft Excel\", \"annotation\":\"Edited
annotation of a MS excel datasource created using REST api\",
\"nativeProperties\":{\"dsn\":\"excelodbc1\"} }]"
```

api\", \"nativeProperties\":{\"dataRange\":\"A2\"} }]"

Example to Unset annotation of a relational datasource "/shared/examples/ds1"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"ds1\",
\"adapterName\":\"PostgreSQL 9.1\", \"annotation\":\"null\"}]"
```

POST/datasource

This API is used to creates datasources for specified database adapter, path and properties

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"adapterName": "string",
```

```
"nativeProperties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
"newName": "string",
"ifNotExists": true
}
]
```

Example to create relational datasource "/shared/examples/ds1"

Sample CURL Invocation

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"ds1\",
\"adapterName\":\"PostgreSQL 9.1\", \"annotation\":\"This is a
postgres datasource created using REST api\",
\"nativeProperties\":{\"urlIP\":\"localhost\", \"urlPort\":5432,
\"urlDatabaseName\":\"orders\", \"login\":\"tutorial\",
\"password\":\"password\"} }]"
```

Example to create file-delimited datasource "/shared/examples/csv_ds"

Sample CURL Invocation

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"csv_ds\",
\"adapterName\":\"File-Delimited\", \"annotation\":\"This is a csv
datasource created using REST api\",
\"nativeProperties\":{\"root\":\"/Users/Shared\",
\"filters\":\"*.csv\", \"charset\":\"utf-8\"} }]"
```

Example to Creates MS excel (non-ODBC) datasource "/shared/examples/excel_ds"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"excel_ds\",
```

```
\"adapterName\":\"Microsoft Excel (non-ODBC)\",
\"annotation\":\"This is a Microsoft excel datasource created using
REST api\", \"nativeProperties\":{\"root\":\"/Users/Shared\",
\"filters\":\"*.xls\", \"charset\":\"utf-8\"} }]"
```

Example to Creates MS excel (ODBC) datasource "/shared/examples/excel_ds"

Sample CURL Invocation of a MS excel (non-ODBC) datasource :

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"excel_ds\",
\"adapterName\":\"Microsoft Excel\", \"annotation\":\"This is a
Microsoft excel datasource created using REST api\",
\"nativeProperties\":{\"dsn\":\"excelodbc\",
\"charset\":\"utf-8\"} }]"
```

DELETE/datasource

This API is used to delete datasources.

Parameters:

None

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Delete datasource "/shared/examples/ds1"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/datasource/v1" -H
"Content-Type:application/json" -d "[\"/shared/examples/ds1\"]"
```

GET/datasource/adapter/definitions

This API is used to get datasource attribute definitions. Schema of returned result is [ADAPTER_NAME, ADAPTER_TYPE, ADAPTER_TYPE_CATEGORY, DEFINITION_NAME, DISPLAY_NAME, DEFINITION_TYPE, REQUIRED, DEFAULT_VALUE, ALLOWED_VALUES, EDITOR_HINT, IS_ADVANCED,DISPLAY_PARENT_NAME, DEPENDENCY_EXPRESSION, UPDATE_RULE, ANNOTATION, DEFINITION_PARENT_NAME]

Parameters:

Name	Description	Parameter Type	Data Type
adapterName	Name of the adapter	query	string

Example to Get datasource attribute defs for "PostgreSQL 9.1"

Sample CURL Invocation

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=PostgreSQL%209.1" -H "Content-Type:application/json"
```

Example to Get datasource attribute defs for "File-Delimited"

Sample CURL Invocation

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=File-Delimited" -H "Content-Type:application/json"
```

Example to Get datasource attribute defs for "Microsoft Excel (non-ODBC)"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=Microsoft%20Excel%20(non-ODBC)" -H
"Content-Type:application/json"
```

Example to Get datasource attribute defs for "Microsoft Excel"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=Microsoft%20Excel" -H "Content-Type:application/json"
```

GET/datasource/virtual

This API is used to get virtual database summary and/or detailed information.

Parameters:

Name	Description	Parameter Type	Data Type
name	name of the virtual database	query	string
summary	fetch virtual database summary	query	boolean

Example to Get summary for virtual database "/services/databases/publishedDB"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual?name=publishedDB
&summary=true" -H "Content-Type:application/json"
```

Example to Get detailed information for virtual database "/services/databases/publishedDB"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=File-Delimited" -H "Content-Type:application/json"
```

Example to Get datasource attribute defs for "Microsoft Excel (non-ODBC)"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/adapter/definitions?adap
terName=Microsoft%20Excel%20(non-ODBC)" -H
"Content-Type:application/json"
```

Example to Get datasource attribute defs for "Microsoft Excel"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual?name=publishedDB
" -H "Content-Type:application/json"
```

PUT/datasource/virtual

This API is used to update virtual datasources.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"adapterName": "string",
"nativeProperties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
"newName": "string",
"ifNotExists": true
}
]
```

Example to Update annotation and rename a published datasource "/services/databases/publishedDB to /services/databases/publishedNew"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual" -H
"Content-Type:application/json" -d "[{\"name\":\"publishedDB\",
\"annotation\":\"Edited published datasource created using REST
api\", \"newName\":\"publishedNew\"  }]"
```

Example to Unset annotation of a published datasource "/services/databases/publishedDB"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual" -H
"Content-Type:application/json" -d "[{\"name\":\"publishedDB\",
\"annotation\":\"null\"}]"
```

POST/datasource/virtual

This API is used to create virtual datasources.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"adapterName": "string",
"nativeProperties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
"newName": "string",
"ifNotExists": true
}
]
```

Example to create virtual datasource "/services/databases/publishedDB"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual" -H
"Content-Type:application/json" -d "[{\"name\":\"ds1\",
\"annotation\":\"This is a published datasource created using REST
api\" }]"
```

Sample CURL Invocation with ifNotExists:

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual" -H
"Content-Type:application/json" -d "[{\"name\":\"ds1\",
\"ifNotExists\":true,\"annotation\":\"This is a published
datasource created using REST api\" }]"
```

DELETE/datasource/virtual

This API is used to delete virtual datasources.

Parameters:

Name	Description	Parameter Type	Data Type
ifExists	Flaf to indicate if datasource exists	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Delete published datasource "/services/databases/publishedDB"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual" -H
"Content-Type:application/json" -d "[\"publishedDB\"]"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual?ifExists=true"
-H "Content-Type:application/json" -d "[\"publishedDB\"]"
```

DELETE/datasource/virtual/dsName

This API is used to delete virtual database.

Parameters:

Name	Description	Parameter Type	Data Type
dsName	Name of the database	query	string
ifExists	Flag to indicate if database exists	query	boolean

Example to delete virtual database

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual/{dsName}?ifExist
s=true"  -H "Content-Type: application/json"
```

GET/datasource/virtual/adapter/definitions

This API is used to get datasource attribute definitions. Schema of returned result is [ADAPTER_NAME, ADAPTER_TYPE, ADAPTER_TYPE_CATEGORY, DEFINITION_NAME, DISPLAY_NAME, DEFINITION_TYPE, REQUIRED, DEFAULT_VALUE, ALLOWED_VALUES, EDITOR_HINT, IS_ADVANCED,DISPLAY_PARENT_NAME, DEPENDENCY_EXPRESSION, UPDATE_RULE, ANNOTATION, DEFINITION_PARENT_NAME]

Parameters:

None.

Example to get datasource attribute defs

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/datasource/v1/virtual/adapter/definiti
ons" -H "Content-Type:application/json"
```

Dataview

The dataview API provides the ability to create, alter, delete and read dataviews. The following operations can be performed:

GET/dataview

PUT/dataview

POST/dataview

DELETE/dataview

DELETE/dataview/{dataviewPath}

GET/dataview

This API is used to get data view summary and/or detailed information.

Parameters:

Name	Description	Parameter Type	Data Type
path	data view path	query	string
summary	fetch data view summary	query	boolean

Example to Get data view summary "/shared/examples/sampleView"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/dataview/v1?path=%2Fshared%2Fexamples%
2FsampleView&summary=true" -H "Content-Type:application/json"
```

Example to Get detailed information of data view "/shared/examples/sampleView"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/dataview/v1?path=%2Fshared%2Fexamples%
2FsampleView" -H "Content-Type:application/json"
```

PUT/dataview

This API is used to update data views.

Parameters:

None

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"sql": "string",
"annotation": "string",
"newPath": "string",
"ifNotExists": true,
"properties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
}
}
]
```

Example to update sql and annotation of data view "/shared/examples/sampleView"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/dataview/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"sampleView\",
```

```
\"sql\":\"SELECT OrderID,CompanyName FROM
/shared/examples/ViewOrder\", \"annotation\":\"Edited annotation of
a data view created using REST api\" }]"
```

Example to Unset annotation of data view "/shared/examples/sampleView"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/dataview/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"sampleView\",
\"annotation\":\"null\" }]"
```

POST/dataview

This API is create data views with specified definition sql.

Parameters:

None

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"sql": "string",
"annotation": "string",
"newPath": "string",
"ifNotExists": true,
"properties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
}
}
]
```

Example to Create data view "/shared/examples/sampleView"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/dataview/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"sampleView\",
```

```
\"sql\":\"SELECT OrderID FROM /shared/examples/ViewOrder\",
\"annotation\":\"This view is created using REST api\"}]"
```

Example to Create data view "/shared/examples/dataview1" with "ifNotExists" syntax

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/dataview/v1" -H
"Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\", \"name\" : \"dataview1\",
\"ifNotExists\":true, \"annotation\":\"This is a data view created
using REST api\" }]"
```

DELETE/dataview

This API is used to delete data views.

Parameters:

Name	Description	Parameter Type	Data Type
ifExists	flag to indicate whether the data view exists.	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Delete data view "/shared/examples/sampleView"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/dataview/v1" -H
"Content-Type:application/json" -d
"[\"/shared/examples/sampleView\"]"
```

Sample CURL Invocation with "ifExists"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/dataview/v1?ifExists=true" -H
"Content-Type:application/json" -d
"[\"/shared/examples/sampleView\"]"
```

DELETE/dataview/{dataviewPath}

This API is used to delete data view. Optionally specify "if exists".

Parameters:

Name	Description	Parameter Type	Data Type
dataViewPath	Path of the data view	query	string
ifExists	flag to indicate if the dataview exists	query	boolean

Example to delete data view

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/dataview/v1/{dataViewPath}?ifExists=tr
ue"  -H "Content-Type: application/json"
```

Sample CURL Invocation with "ifExists"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/dataview/v1/%2Fshared%2Fexamples%2Fdat
aview1?ifExists=true" -H "Content-Type:application/json"
```

Deployment Manager

This section describes the following Deployment Manager Procedures:

- POST /executeQuery
- POST /executeDDL
- POST /executePlan
- GET /export_dm_metadata
- GET /export_plan_package
- POST /import_dm_metadata
- DELETE /purgeLog
- GET /validateSite

POST /executeQuery

This API is used to execute query.

Parameters:

None

Request Body

Example Value - Schema

```
[
“query”: "string",
“standardSQL”: “string”
]
```

Example

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeQuery" -H "accept:
application/json" -H "Content-Type:
application/x-www-form-urlencoded" -d "query=select * from
SYS_SITES&standardSql=true"
```

POST /executeDDL

This API is used to execute DDL commands.

Parameters:

None

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Create Site

```
curl -X POST  -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
RESOURCE /mySite OF TYPE "site" SET ANNOTATION "my site annotation"
SET PROPERTIES { "host" : "localhost", "port": 9400,
"user":"admin", "password":"password", "domain":"composite"}'
```

Example to Update Site

```
curl -X POST  -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/mySite OF TYPE "site" SET ANNOTATION "my site annotation update "'
```

Example to Delete Site

```
curl -X POST  -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/mySite OF TYPE "site"'
```

Example to Create Resource Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
RESOURCE /mySite/myResourceSet OF TYPE "resource_set" SET
ANNOTATION "my resource set annotation" SET PROPERTIES {
"definition" :
"{\""includeDependencies\"":\""true\"",\""resourceTrees\"":[{\""pa
th\"":\""/shared\"",\""type\"":\""CONTAINER\""}]}"}'
```

Example to Update Resource Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/mySite/myResourceSet OF TYPE "resource_set" SET ANNOTATION "my
resource set annotation update" '
```

Example to Delete Resource Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/mySite/myResourceSet OF TYPE "resource_set"'
```

Example to Create Resource Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
```

```
RESOURCE
/targetSite/sourceSite/shared/examples/ds_orders#DATA_SOURCE OF
TYPE "attribute_mapping" SET PROPERTIES { "definition" : {
"urlPort" : "1000"}}'
```

Example to Update Resource Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/targetSite/sourceSite/shared/examples/ds_orders#DATA_SOURCE OF
TYPE "attribute_mapping" SET PROPERTIES { "definition" : {
"urlPort" : "2000"}}'
```

Example to Delete Resource Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/targetSite/sourceSite/shared/examples/ds_orders#DATA_SOURCE OF
TYPE "attribute_mapping"'
```

Example to Create Principal Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
RESOURCE /mySourceSite/myPrincipalSet OF TYPE "principal_set" SET
PROPERTIES { "definition" : [ "/composite/user/dev"] }'
```

Example to Update Principal Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/mySourceSite/myPrincipalSet OF TYPE "principal_set" SET ANNOTATION
"my principal set annotation update"'
```

Example to Delete Principal Set

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/mySourceSite/myPrincipalSet OF TYPE "principal_set"'
```

Example to Create Principal Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
RESOURCE /myTargetSite/mySourceSite OF TYPE "principal_set_mapping"
SET PROPERTIES { "definition" : { "/composite/user/dev" :
"/composite/user/test"}}'
```

Example to Update Principal Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/myTargetSite/mySourceSite OF TYPE "principal_set_mapping" SET
PROPERTIES { "definition" : { "/composite/user/dev" :
"/composite/user/dev1"}}'
```

Example to Delete Principal Set Mapping

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/myTargetSite/mySourceSite OF TYPE "principal_set_mapping"'
```

Example to Create Deployment Plan

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'CREATE
RESOURCE
/myTargetSite/mySourceSite/MyDeploymentPlan_PrincipalSet_ADD OF
TYPE "deployment_plan" SET ANNOTATION "my deployment plan
annotation" SET PROPERTIES { "definition" :
"{\""operations\"":{\""principalSet\"":\""myPrincipalSet\"",\""tar
getOperation\"":\""ADD\""}}"}'
```

Example to Update Deployment Plan

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'ALTER RESOURCE
/myTargetSite/mySourceSite/MyDeploymentPlan_PrincipalSet_ADD OF
TYPE "deployment_plan"'
```

Example to Delete Deployment Plan

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executeDDL" -H
"Content-Type:text/plain" -H "Accept:text/plain" -d 'DROP RESOURCE
/myTargetSite/mySourceSite/MyDeploymentPlan_PrincipalSet_ADD OF
TYPE "deployment_plan"'
```

POST /executePlan

This API is used to execute Deployment Plan.

Parameters:

None

Request Body

Example Value - Schema

```
[
"string"
]
```

Example

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/executePlan" -d
'/myTargetSite/mySourceSite/DeploymentPlanName'
```

GET /export_dm_metadata

This API is used to export Deployment Management Artifacts.

Parameters:

Name	Description	Parameter Type	Data Type
encryptionPassword	Encryption password	query	string

Example

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/deploy/export_dm_metadata" -H
"Content-Type:application/binary"
```

GET /export_plan_package

This API is used to export Deployment Plan.

Parameters:

Name	Description	Parameter Type	Data Type
plan	The plan name that will be exported	query	string
encryptionPassword	Encryption password	query	string

Example

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/deploy/export_plan_package?plan=planNa
me" -H "Content-Type:application/binary"
```

POST /import_dm_metadata

This API is used to import Deployment Management Artifacts.

Parameters

Name	Description	Parameter Type	Data Type
file	The name of the file to be imported.	query	object
overwrite	Option to indicate whether to overwrite.	query	boolean
encryptionPassword	Encryption Password	query	string
ignoreEncryption	Option to indicate whether to ignore encryption errors.	query	boolean

Note: If the option *ignoreEncryption* is used, then all backup data will be imported regardless of whether a valid encryption key was provided. This means that the import will not fail. This option can be used to allow partially importing any backed up data. However, the import process will only import data that is not encrypted or can be decrypted using the provided encryption key. All encrypted portions of the backup data that cannot be decrypted will be imported as empty values and the import will otherwise succeed.

This affects all encrypted values in the backup data, which includes, but is not limited to data source and LDAP domain connection passwords.

Example

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/deploy/import_dm_metadata" -H
"Content-Type:multipart/form-data" -v -F 'file=@localfilename' -F
“ignoreEncryption=true” -X 'overwrite=true/false'
```

DELETE /purgeLog

This API is used to purge Deployment Logs.

Parameters

Name	Description	Parameter Type	Data Type
ids	The plan ids	query	string
begin	The log begin time	query	string
end	The log end time	query	string

Example

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/deploy/purgeLog" -H "begin:0" -H
"end:2408386295553"
```

GET /validateSite

This API is used to validate Site Information.

Parameters:

Name	Description	Parameter Type	Data Type
siteName	Name of the site	query	string

Example

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/deploy/validateSite?siteName=sourceSit
e"
```

Execute

The execute API provides the ability to execute queries, procedures and multiple DSL statements.

POST/execute/query

POST/execute/procedure

POST/execute/cancel

GET/execute/nextBatch

POST/execute/DSL

POST/execute/sqlscript

POST/execute/query

This API is used to execute a SQL query against the TDV server.

Parameters:

None

Request Body

Example Value - Schema

```
{"standardSQL": true,
“query”: “string”,
“skipRows”: 0,
“maxRows”: 0
“dataServiceName”: “string”
“blocking”: true
}
```

standardSQL - The default value is true. Set this to false for performing a data preview. The query must be a composite query (non-standard).

isBlocking - The default value is true. Set this to false to execute query in an asynchronous fashion.

skipRows - If this value is set, then that many number of rows will be skipped in the execution output before returning any results. If 'skipRows' is greater than the total possible number of rows, then no rows will be returned. The default value is 0.

maxRows - If this value is set, then the result will contain at most 'maxRows' number of rows. If 'maxRows' is fewer than the total number of rows of data available, then additional calls to "getNextBatch" will need to be made to get the rest of the available data. If 'maxRows' is not set, then the result row count is request memory-bound.

Note: skipRows and maxRows are used for a blocking query execution.

Following are the status codes that are returned:

- 206 (Partial Content) - This exception occurs when partial results are obtained due to memory constraints. The result is set in the entity.
- 409 (Conflict) - This exception occurs when a request is canceled before being consumed completely.
- 406 (Not Acceptable) - This exception occurs when the input request id represents a blocking request.
- 500 (Internal Server Error) or 400 (Bad Request) - This occurs when a request fails.
- 200 (OK) - When the maxRows is set and is greater than the number of available rows in the result, then all the available rows are returned with this status code.

Example to execute a query to get a property of a sql script procedure.

```
curl -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d "{\"query\":\"SELECT * FROM
model.ALL_RESOURCE_PROPERTIES WHERE property_name = 'script' AND
metadata_id = (SELECT PROCEDURE_ID FROM model.ALL_PROCEDURES WHERE
PROCEDURE_NAME = 'ctasScript' AND parent_path =
'/shared/examples')\", \"standardSQL\":true}"
```

Example to search for “product”

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d '{\"standardSQL\":false,
\"query\":\"select * from
/lib/resource/\"Search\"('product',null,null,null,null,null,null,n
ull)\", \"blocking\":false}'
```

Example to search for “product”, filter “column” and “parameter” field types and filter resource types “table” and “procedure”'

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d '{\"standardSQL\":false,
\"query\":\"select * from
/lib/resource/\"Search\"('product','column,parameter','table,proce
dure',null,null,null,null,null)\", \"blocking\":false}'
```

Example to search for “product”, filter “annotation” field type and mark start of search as { and stop of search as }

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d '{\"standardSQL\":false,
\"query\":\"select * from
/lib/resource/\"Search\"('product','annotation',null,'StartSel:<,S
topSel:>',null,null,null,null)\", \"blocking\":false}'
```

Example to search for “product” and report data flows

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d '{\"standardSQL\":false,
\"query\":\"select * from
/lib/resource/\"Search\"('product',null,null,null,1,null,null,null
)\", \"blocking\":false}'
```

Example to search for “product”, assign fieldRanks for “column” and “parameter” fields, set rowOffset to 10 to skip 10 rows in the result,set rowLimit to 30 rows in the result.

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/query/invoke" -H
"Content-Type:application/json" -d '{\"standardSQL\":false,
```

```
\"query\":\"select * from
/lib/resource/\"Search\"('product',null,null,null,null,'column:3,p
arameter:3',10,30)\", \"blocking\":false}'
```

POST/execute/procedure

This API is used to execute procedure specified by path / type.

Parameters:

None

Request Body

Example Value - Schema

```
[
"includeMetadata": true,
“parameterBeanList”: [
{
“definition”: “string”,
“value”: “string”
}
],
“path”: “string”,
“type”: “string”,
“blocking”: true
]
```

Example to execute procedure "/shared/examples/LookupProduct"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/execute/v1/actions/procedure/invoke"
-H "Content-Type:application/json"  -d
"{\"blocking\":\"true\",\"includeMetadata\":\"true\",
\"parameterBeanList\":[{\"definition\":\"INTEGER\",
\"value\":\"1\"}],\"path\":\"/shared/examples/LookupProduct\",
\"type\":\"PROCEDURE\"}"
```

POST/execute/cancel

This API is used to cancel asynchronous/synchronous sql request with the specified execution ids. Procedure requests are unsupported for cancellation.h

Parameters:

None

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to cancel a request

```
curl -X POST
"http://localhost:9400/rest/execute/v1/actions/cancel/invoke" -H
"Content-Type: application/json" -d "[\"400014-0\"]"
```

GET/execute/nextBatch

This API is used to get next batch of data. Only Asynchronous sql request execution ids may be specified. When batch size is unspecified, it is defaulted to the server configuration 'defaultFetchRows'. The various status codes returned are:

- 206 (Partial Content) - This exception occurs when partial results are obtained due to memory constraints. The result is set in the entity.
- 409 (Conflict) - This exception occurs when a request is canceled before being consumed completely.
- 406 (Not Acceptable) - This exception occurs when the input request id represents a blocking request.
- 500 (Internal Server Error) or 400 (Bad Request) - This occurs when a request fails.
- 200 (OK) - This status code indicates that there has been no errors. Following are some scenarios:
  - When the batchSize is greater than the number of available rows in the result, then all the available rows are returned with this status code.
  - When the result is completely consumed, an empty result set is returned with this status code.
  - When no request is found for the specified request id, an empty result set is returned with this status code.

Parameters:

Name	Description	Parameter Type
executionId	Execution ID	string
batchSize	Batch size to fetch. When 'batchSize' is not set, the value set in server configuration 'defaultFetchRows' is used.	integer

Example to get next batch of data

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/execute/v1/actions/nextBatch/invoke?ex
ecutionId=400014-0&batchSize=5" -H "Content-Type:application/json"
```

POST/execute/DSL

This API is used to execute the specified list of DSL statements.

Parameters:

None

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Drop a Sql script procedure and create one using two DSL statements.

```
curl -d "[\"CREATE SCRIPT /shared/examples/ctasScript DEFINE AS
PROCEDURE ctasScript() BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/sampleTable as select
OrderId, ProductID, Discount, OrderDate, CompanyName,
CustomerContactFirstName, CustomerContactLastName,
CustomerContactPhone FROM /shared/examples/ViewOrder; END\",\"DROP
SCRIPT /shared/examples/ctasScript\"]" -u "admin:admin" -X POST
"http://localhost:9400/rest/execute/v1/actions/dsl/invoke" -H
"Content-Type:application/json"
```

POST/execute/sqlscript

This API is used to execute procedure specified by scriptText.

Parameters:

None

Request Body

Example Value - Schema

```
[
"includeMetadata": true,
“parameterBeanList”: [
{
“definition”: “string”,
“value”: “string”
}
],
“scriptText”: “string”,
“blocking”: true
]
```

Example to execute procedure using script text

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/execute/v1/actions/sqlscript/invoke"
-H "Content-Type:application/json"  -d
"{\"blocking\":\"true\",\"includeMetadata\":\"true\",
\"parameterBeanList\":[{\"definition\":\"INTEGER\",
\"value\":\"1\"}],\"scriptText\":\"PROCEDURE LookupProduct(IN
desiredProduct INTEGER,OUT result CURSOR (ProductName
VARCHAR(50),ProductID INTEGER,ProductDescription VARCHAR(255)))
BEGIN OPEN result FOR SELECT products.ProductName,
products.ProductID, products.ProductDescription FROM
/shared/examples/ds_inventory/tutorial/products products WHERE
products.ProductID = desiredProduct; END\"}"
```

Folders

The folder API provides the ability to create, alter or delete multiple folders and read a folder.

GET/folder

PUT/folder

GET/folder

This API is used to get folder.

Parameters:

Name	Description	Parameter Type	Data Type
path	folder path	query	string
summary	fetch folder summary	query	boolean

Example to Get summary of folder "/shared/examples/folder1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/folder/v1?path=%2Fshared%2Fexamples%2F
folder1&summary=true" -H "Content-Type:application/json"
```

Example to Get summary and detailed information of folder "/shared/examples/folder1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/folder/v1?path=%2Fshared%2Fexamples%2F
folder1" -H "Content-Type:application/json"
```

PUT/folder

This API is used to update folders.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
```

```
"name": "string",
"properties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
"ifNotExists": true
}
]
```

Example to Update folder with a modified annotation "/shared/examples/folder1"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"folder1\",
\"annotation\":\"Edited annotation of a folder created using REST
api\" }]"
```

Example to Rename folder "/shared/examples/folder1" to "/shared/examples/folder2"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"folder1\",
\"newPath\":\"/shared/examples/folder2\" }]"
```

Example to Relocate & Rename folder "/shared/examples/folder1" to "/users/composite/admin/folder2"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"folder1\",
\"newPath\":\"/users/composite/admin/folder2\" }]"
```

Example to Unset annotation on folder "/shared/examples/folder1"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"folder1\",
\"annotation\":\"null\" }]"
```

POST/folder

This API is used to Create folders using specified paths.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
"parentPath": "string",
"name": "string",
"properties": {
"additionalProp1": {},
"additionalProp2": {},
"additionalProp3": {}
},
"annotation": "string",
"newPath": "string",
"ifNotExists": true
}
]
```

Example to Create folder "/shared/examples/folder1"

```
curl -X POST -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"folder1\",
\"annotation\":\"This folder is created using REST api\"}]"
```

Example to Create folder "/shared/examples/folder1" with "ifNotExists" syntax

```
curl -X POST -u admin:admin "http://localhost:9400/rest/folder/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\", \"name\" : \"folder1\",
\"ifNotExists\":true, \"annotation\":\"This is a folder created
using REST api\" }]"
```

DELETE/folder

This API is used to delete folders.

Parameters:

Name	Description	Parameter Type	Data Type
ifExists	flag to indicate whether a folder exists	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Delete folder "/shared/examples/folder1"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/folder/v1" -H
"Content-Type:application/json" -d
"[\"/shared/examples/folder1\"]"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/folder/v1" -H
"Content-Type:application/json" -d
"[\"/shared/examples/folder1\"]"
```

DELETE/folder/{folderPath}

This API is used to delete folder in the path. Delete folder. Optionally specify "if exists".

Parameters:

Name	Description	Parameter Type	Data Type
folderPath	Path of the folder	query	string

Name	Description	Parameter Type	Data Type
ifExists	flag to indicate whether a folder exists	query	boolean

Example to Delete folder.

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/folder/v1/{folderPath}?ifExists=true"
-H "Content-Type: application/json"
```

Sample CURL Invocation with "ifExists":

curl -X DELETE -u admin:admin
"http://localhost:9400/rest/folder/v1/%2Fshared%2Fexamples%2Ffolder1?ifExists=true" -H "Content-Type:application/json"

Link

The Link REST API provides the ability to create, alter or delete multiple virtual tables/procedures and read a virtual table/procedure.

GET/link

PUT/link

POST/link

DELETE/link

DELETE/link/{linkPath}

GET/link

This API is used to get summary and/or detailed information of virtual table or procedure.

Parameters:

Name	Description	Parameter Type	Data Type
path	virtual table or procedure path	query	string
summary	fetch virtual table or procedure summary	query	boolean

Example to Get summary of published table "/services/databases/publishedDB/pubTable"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/link/v1?path=%2Fservices%2Fdatabases%2
FpublishedDB%2FpubTable&summary=true" -H
"Content-Type:application/json"
```

Example to Get detailed information about published table "/services/databases/publishedDB/pubTable"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/link/v1?path=%2Fservices%2Fdatabases%2
FpublishedDB%2FpubTable" -H "Content-Type:application/json"
```

Example to Get detailed information about published procedure "/services/databases/publishedDB/pubProcedure"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/link/v1?path=%2Fservices%2Fdatabases%2
FpublishedDB%2FpubProcedure" -H "Content-Type:application/json"
```

PUT/link

This API is used to update virtual tables or procedures.

Parameters:

None.

Request Body

Example Value - Schema

```
[
“path”: "string",
“isTable”: true,
“targetPath”: “string”,
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
]
```

Example to Update annotation and rename a published table "/services/databases/publishedDB/link1 to /services/databases/

publishedDB/pubTable"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/link1\", \"isTable\"
: true, \"annotation\":\"Edited published table created using REST
api\", \"newPath\":\"/services/databases/publishedDB/pubTable\"
}]"
```

Example to Update annotation and relocate a published procedure "/services/databases/publishedDB/link2 to /services/databases/publishedDB1/pubProcedure"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/link2\", \"isTable\"
: false, \"annotation\":\"Edited published procedure created using
REST api\",
\"newPath\":\"/services/databases/publishedDB1/pubProcedure\"  }]"
```

Example to Update target of a published table "/services/databases/publishedDB/pubTable"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/pubTable\",
\"isTable\" : true, \"annotation\":\"Edited target of published
table created using REST api\",
\"targetPath\":\"/shared/examples/ds_orders/tutorial/orderdetails\
"  }]"
```

Example to Unset annotation of a published table "/services/databases/publishedDB/pubTable"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/pubTable\",
\"isTable\" : true, \"annotation\":\"null\"}]"
```

POST/link

This API is used to creates virtual tables or procedures.

Parameters:

None.

Request Body

Example Value - Schema

```
[
“path”: "string",
“isTable”: true,
“targetPath”: “string”,
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
]
```

Example to Create virtual table "/services/databases/publishedDB/link1"

```
curl -X POST -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/link1\",
\"isTable\":true, \"targetPath\" :
\"/shared/examples/ds_orders/tutorial/orders\",
\"annotation\":\"This is a published table created using REST api\"
}]"
```

Example to Creates virtual procedure "/services/databases/publishedDB/link2"

curl -X PUT -u admin:admin "http://localhost:9400/rest/link/v1" -H "Content-Type:application/json" -d "[{\"path\":\"/services/databases/publishedDB/link2\", \"isTable\" : false, \"annotation\":\"Edited published procedure created using REST api\", \"newPath\":\"/services/databases/publishedDB1/pubProcedure\" }]"

Example to Update target of a published table "/services/databases/publishedDB/pubTable"

```
curl -X POST -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/link2\",
\"isTable\":false, \"targetPath\" :
\"/shared/examples/LookupProduct\", \"annotation\":\"This is a
published procedure created using REST api\" }]"
```

Example to Creates virtual table "/services/databases/publishedDB/link1" with "ifNotExists" syntax

```
curl -X POST -u admin:admin "http://localhost:9400/rest/link/v1" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/link1\",
\"ifNotExists\":true, \"isTable\":true, \"targetPath\" :
\"/shared/examples/ds_orders/tutorial/orders\",
\"annotation\":\"This is a published table created using REST api\"
}]"
```

DELETE/link

This API is used to delete virtual tables or procedures.

Parameters:

None.

Request Body

Example Value - Schema

```
[
“path”: "string",
“isTable”: true
]
```

Example to Deletes virtual table and procedure "/services/databases/publishedDB/pubTable" and "/services/databases/publishedDB/pubProcedure"

```
curl -X DELETE -u admin:admin "http://localhost:9400/rest/link/v1"
-H "Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/pubTable\",
\"isTable\":true},
{\"path\":\"/services/databases/publishedDB/pubProcedure\",
\"isTable\":false}]"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/link/v1?ifExists=true" -H
"Content-Type:application/json"  -d
"[{\"path\":\"/services/databases/publishedDB/pubTable\",
\"isTable\":true},
```

```
{\"path\":\"/services/databases/publishedDB/pubProcedure\",
\"isTable\":false}]"
```

DELETE/link/{linkPath}

This API is used to delete virtual tables or procedures.

Parameters:

Name	Description	Parameter Type	Data Type
linkPath	virtual table or procedure path	query	string
isTable	flag to indicate if resource is a table	query	boolean
ifExists	flag to indicate if resource exists	query	boolean

Example to Deletes virtual table

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/link/v1/{linkPath}?isTable=true&ifExis
ts=true"  -H "Content-Type: application/json"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/link/v1/%2Fservices%2Fdatabases%2Fpubl
ishedDB%2FpubProcedure?isTable=false&ifExists=true" -H
"Content-Type:application/json"
```

Resource

The REST API operations that can be performed on the resources are:

- GET /children
- GET /custom_functions
- GET /columns

GET /children

This API is used to get the children of resources.

Parameters:

Name	Description	Parameter Type	Data Type
path	resource path	query	string
type	resource type	query	string
metadataVersion	resource version	query	long
start	the start index in children list	query	integer
count	the count of children	query	integer
component	component name, (only supports “cbs” now.)	query	string

Example to get children of resource "/shared/examples"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/resource/v1/children?path=/shared/exam
ples&type=CONTAINER"
```

Example to get children of resource "/shared/examples" by page

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/resource/v1/children?path=/shared/exam
ples&type=CONTAINER&metadataVersion=&start=0&cound=10"
```

GET /custom_functions

This API is used to get all custom functions.

Parameters:

None

Example to get all custom functions

curl -X GET -u admin:admin
"http://localhost:9400/rest/resource/v1/custom_functions"

GET /columns

This API is used to get table/view columns.

Parameters:

Name	Description	Parameter Type	Data Type
path	table path	query	string
metadataVersion	table version	query	long
start	the start index in children list	query	integer
count	the count of children	query	integer

Example to get view "/shared/examples/ViewOrder" columns

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/resource/v1/table/columns?path=/shared
/examples/ViewOrder"
```

Example to get view "/shared/examples/ViewOrder" columns by page

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/resource/v1/table/columns?path=/shared
/examples/ViewOrder&metadataVersion=&start=0&cound=10"
```

Schema

The schema REST API provides the ability to create, alter or delete multiple (published) virtual schemas. The operations that can be performed on the resources are:

- GET /schema/virtual
- PUT /schema/virtual
- POST /schema/virtual
- DELETE /schema/virtual
- DELETE /schema/virtual/{schemaPath}

GET /schema/virtual

This API is used to get summary and/or detailed information about a virtual schema.

Parameters:

Name	Description	Parameter Type	Data Type
path	virtual schema path	query	string
summary	fetch virtual schema summary	query	boolean

Example to Get summary of virtual schema "/services/databases/publishedDB/sch1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual?path=%2Fservices%2Fd
atabases%2FpublishedDB%2Fsch1&summary=true" -H
"Content-Type:application/json"
```

Example to Get detailed information about virtual schema "/services/databases/publishedDB/sch1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual?path=%2Fservices%2Fd
atabases%2FpublishedDB%2Fsch1" -H "Content-Type:application/json"
```

PUT /schema/virtual

This API is used to Update virtual schemas.

Parameters:

None.

Request Body

Example Value - Schema

```
[
“path”: "string",
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
]
```

Example to Update annotation and rename a published schema "/services/databases/publishedDB/sch1 to /services/databases/publishedDB/sch2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/sch1\",
\"annotation\":\"Edited published schema created using REST api\",
\"newPath\":\"/services/databases/publishedDB/sch2\"  }]"
```

Example to Update annotation and relocate a published schema "/services/databases/publishedDB/sch2 to /services/databases/publishedDB1/sch2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/sch2\",
\"annotation\":\"Edited published schema created using REST api\",
\"newPath\":\"/services/databases/publishedDB1/sch2\"  }]"
```

Example to Unset annotation of published schema "/services/databases/publishedDB/sch1"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/sch1\",
\"annotation\":\"null\" }]"
```

POST /schema/virtual

This API is used to create virtual schemas.

Parameters:

None.

Request Body

Example Value - Schema

```
[
“path”: "string",
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
]
```

Example to Create virtual schema "/services/databases/ publishedDB/sch1"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/sch1\",
\"annotation\":\"This is a published schema created using REST
api\" }]"
```

Example to create virtual schema "/services/databases/ publishedDB/sch1" with "ifNotExists" syntax

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual" -H
"Content-Type:application/json" -d
"[{\"path\":\"/services/databases/publishedDB/sch1\",
\"ifNotExists\":true, \"annotation\":\"This is a published schema
created using REST api\" }]"
```

DELETE /schema/virtual

This API is used to delete virtual schemas.

Parameters:

Name	Description	Parameter Type	Data Type
ifExists	flag to indicate if resource exists	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to Delete virtual schema "/services/databases/publishedDB/sch1" and "/services/databases/ publishedDB/sch2"

```
curl -X DELETE -u admin:admin "http://localhost:9400/rest/
schema/v1/virtual" -H "Content-Type:application/ json" -d
"[\"/services/databases/ publishedDB/sch1\",
\"/services/databases/ publishedDB/sch2\"]"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual?ifExists=true" -H
"Content-Type:application/json" -d
"[\"/shared/examples/publishedDB/sch2\"]"
```

DELETE /schema/virtual/{schemaPath}

This API is used to delete virtual schema given in the path.

Parameters:

Name	Description	Parameter Type	Data Type
schemaPath	Path of the schema	query	string
ifExists	flag to indicate if resource exists	query	boolean

Example to Delete virtual schema

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual/{schemaPath}?ifExist
s=true"  -H "Content-Type: application/json"
```

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/schema/v1/virtual/%2Fservices%2Fdataba
```

```
ses%2Fpubdb%2Fschema1?ifExists=true" -H
"Content-Type:application/json"
```

Script

The script REST API provides the ability to create, alter, delete, read sql script procedures. The operations that can be performed on the resources are:

- GET /script
- PUT /script
- POST /script
- DELETE /script
- DELETE /script/{scriptPath}

GET /script

This API is used to get summary of sql script procedure.

Parameters:

Name	Description	Parameter Type	Data Type
path	sql script procedure path	query	string
summary	fetch sql script procedure summary	query	boolean

Example to get summary of sql script procedure "/shared/examples/LookupProduct"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/script/v1?path=%2Fshared%2Fexamples%2F
LookupProduct&summary=true" -H "Content-Type:application/json"
```

Example to Get detailed information about sql script procedure "/shared/examples/LookupProduct"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/script/v1?path=%2Fshared%2Fexamples%2F
LookupProduct" -H "Content-Type:application/json"
```

PUT /script

This API is used to Update sql script procedures.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
“parentPath”: "string",
“name”: “string”,
“script”: “string”,
“properties”:
{
“additionalProp1”:{},
“additionalProp2”:{},
“additionalProp3”:{}
},
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
}
]
```

Example to update script with a modified CTAS and annotation of script "/shared/examples/script1"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/script/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"script1\",
\"script\":\"PROCEDURE script1() BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/sampleTable as select
OrderId, ProductID, OrderDate, CustomerContactFirstName,
CustomerContactLastName FROM /shared/examples/ViewOrder; END\",
\"annotation\":\"Edited annotation of a script created using REST
api\" }]"
```

Example to Update script to unset annotation of script "/shared/examples/script1"

```
curl -X PUT -u admin:admin "http://localhost:9400/rest/script/v1"
-H "Content-Type:application/json" -d
```

```
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"script1\",
\"annotation\":\"null\" }]"
```

POST /script

This API is used to create sql script procedures with specified definition script.

Parameters:

None.

Request Body

Example Value - Schema

```
[
{
“parentPath”: "string",
“name”: “string”,
“script”: “string”,
“properties”:
{
“additionalProp1”:{},
“additionalProp2”:{},
“additionalProp3”:{}
},
“annotation”: “string”,
“newPath”: “string”,
“ifNotExists”: true
}
]
```

Example to create script "/shared/examples/script1"

```
curl -X POST -u admin:admin "http://localhost:9400/rest/script/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"script1\",
\"script\":\"PROCEDURE script1() BEGIN CREATE TABLE
/shared/examples/ds_inventory/tutorial/sampleTable as select
OrderId, ProductID, Discount, OrderDate, CompanyName,
CustomerContactFirstName, CustomerContactLastName,
CustomerContactPhone FROM /shared/examples/ViewOrder; END\",
\"annotation\":\"This script is created using REST api\" }]"
```

Example to create script "/shared/examples/script1" with "ifNotExists"

syntax

```
curl -X POST -u admin:admin "http://localhost:9400/rest/script/v1"
-H "Content-Type:application/json" -d
"[{\"parentPath\":\"/shared/examples\",  \"name\":\"script1\",
\"ifNotExists\":true, \"script\":\"PROCEDURE script1() BEGIN CREATE
TABLE /shared/examples/ds_inventory/tutorial/sampleTable as select
OrderId, ProductID, Discount, OrderDate, CompanyName,
CustomerContactFirstName, CustomerContactLastName,
CustomerContactPhone FROM /shared/examples/ViewOrder; END\",
\"annotation\":\"This script is created using REST api\" }]"
```

DELETE /script

This API is used to delete sql script procedures.

Parameters:

Name	Description	Parameter Type	Data Type
ifExists	Flag to indicate if resource exists	query	boolean

Request Body

Example Value - Schema

```
[
"string"
]
```

Example to delete script "/shared/examples/script1"

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/script/v1" -H
"Content-Type:application/json" -d
"[\"/shared/examples/script1\"]"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/script/v1?ifExists=true" -H
"Content-Type:application/json" -d
"[\"/shared/examples/script1\"]"
```

DELETE /script/{scriptPath}

This API is used to Delete sql script procedure given in the script path. Optionally specify "if exists".

Parameters:

Name	Description	Parameter Type	Data Type
scriptPath	path of the script	query	string
ifExists	Flaf to indicate if resource exists	query	boolean

Example to delete sql script procedure

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/script/v1/{scriptPath}?ifExists=true"
-H "Content-Type: application/json"
```

Sample CURL Invocation with "ifExists":

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/script/v1/%2Fshared%2Fexamples%2Fscrip
t1?ifExists=true" -H "Content-Type:application/json"
```

Security

The following Security operations can be performed:

- GET /backup_encryption_settings
- GET /systemEncryption
- GET /domains
- GET /domains/groups, page 465
- POST/domains/groups/sync, page 465
- GET /backup_encryption_settings, page 464
- GET /domains/domain_users, page 466
- GET /generateUUID
- GET /systemEncryption

- PUT /systemEncryption

GET /backup_encryption_settings

This API is used to backup the encryption settings to a password protected file for server recovery in case of emergency.

Parameters:

Name	Description	Parameter Type	Data Type
encryptionPassword	Encryption Password	query	string

Example to backup the encryption settings

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/backup_encryption_settings
?encryptionPassword=testPassword" -o
backup_encryption_settings.txt
```

POST /import_encryption_settings

This API is used to restore the encryption settings from the backup file. You must know the password that was used to protect the backup file.

Parameters:

Name	Description	Parameter Type	Data Type
file	Name of the file to be imported	query	object
encryptionPassword	Encryption Password	query	string

Example to restore the encryption settings

```
curl -u "admin:admin" -i -F "encryptionPassword=testPassword" -F
"file=@backup_encryption_settings.txt" -X POST
"http://localhost:9400/rest/security/v1/import_encryption_settings
"
```

GET /domains

This API is used to get all domains.

Parameters:

None

Example to get all domains

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/domains"
```

GET /domains/groups

This API is used to get all domain groups.

Parameters:

Name	Description	Parameter Type	Data Type
domain	The name of the domain	path	string

Example to get all “composite” domain groups

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/domains/composite/groups"
```

POST/domains/groups/sync

This API is used to synchronize the given users in all external ldap domain groups for an external ldap domain.

Parameters

Name	Description	Parameter Type	Data Type
domain	The name of domain	path	string

Request Body

Example Value - Schema

```
[
 "string"
]
```

Example to synchronize the groups of "ldap" domain users "user1" and "user2"

```
curl -X POST
"https://localhost:9402/rest/security/v1/domains/ldap/groups/sync"
-H "accept: */*" -H "Content-Type: application/json" -d
"[\"user1\",\"user2\"]"
```

GET /domains/domain_users

This API is used to get all domain users.

Parameters:

Name	Description	Parameter Type	Data Type
domain	The name of the domain	path	string

Example to get all “composite” domain users

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/domains/composite/users"
```

GET /generateUUID

This API is used to get the randomly generated system UUID.

Parameters:

None

Example to get the system randomly generated UUID.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/generateUUID"
```

GET /systemEncryption

This API is used to get the system encryption settings.

Parameters:

None

Example to get the system encryption settings.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/security/v1/systemEncryption"
```

PUT /systemEncryption

This API is used to update the system encryption settings. Can take a long time in large database to re-encrypt data.

Parameters:

Name	Description	Parameter Type	Data Type
body	Encryption settings (algorithm, password, uuid, key size	body	Model schema

Request Body

Example Value - Schema

```
{
“algorithm”: "string",
“password”: “string”,
“uuid”: “string”,
“keySize”: “string”
]
```

Example to update the system encryption settings

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/security/v1/systemEncryption" -H
"Content-Type:application/json" -d
"{\"password\":\"MyTestEncryptionPassword\",\"uuid\":\"0b352e1e-ab
56-4271-a813-31183df63788\"}"
```

Session

The session operations that can be performed are:

- GET /session
- PUT /session

- DELETE /session

GET /session

This API is used to get information about the currently open session. This may include updates to a user's rights.

Parameters

None

Example to get session information

```
curl -X GET -u admin:admin "https://localhost:9400/rest/session"
```

Example to get session information as ldap user

```
curl -X GET -u user@ldapDomain:password "https://localhost:9400/
rest/session"
```

PUT /session

This API is used to initiate a long running session with the TDV server. Returns information about the current session including a session token and the current user object. The session token should be used for all following Rest API calls and be placed within the session HTTP cookie.

Parameters

None

Request Body

Example Value - Schema

```
{
“user”: {
“name”: “string”,
“domainName”: “string”,
“id”: 0,
“annotation”: “string”,
“memberReferences”: [
{
“memberName”: “string”
“domainName”: “string”
```

```
}
],
“rights”: 0,
“effectiveRights”: 0,
“inheritedRights”: 0,
“attributes”: {
“empty”: true
},
“locked”: true
},
“sessionToken”: “string”,
“autoCloseMode”: true
}
```

Example to begin a new session

```
curl -X PUT -u admin:admin "https://localhost:9400/rest/session"
```

Example to begin a new session as ldap user

```
curl -X PUT -u user@ldapDomain:password "https://localhost:9400/
rest/session"
```

DELETE /session

This API is used to end the current session and invalidate the session token that was previously returned when creating the session.

Parameters

None

Example to end new session

```
curl -X DELETE -u admin:admin "https://localhost:9400/rest/session"
```

Example to end new session as ldap user

```
curl -X DELETE -u user@ldapDomain:password "https://localhost:9400/
rest/session"
```

Version Control System

The VCS operations that can be performed on the TDV resources are:

- GET /branches
- GET /branches/{name}
- POST /checkin/{name}
- GET /connection
- GET /connection/{name}
- GET /content/{name}
- POST /discard/{name}
- GET /enable
- POST /fetch/{name}
- GET /history/{name}
- GET /latestcontent/{name}
- GET /localcontent/{name}
- GET /root
- GET /root/{name}
- POST /root/{name}
- DELETE /root/{name}
- POST /setCredential/{name}
- GET /status/{name}
- GET /vcsAdapter/{adapter_name}
- GET /vcsAdapters
- POST /vcsInstance
- GET /vcsInstance/{name}
- PUT /vcsInstance/{name}
- DELETE /vcsInstance/{name}
- GET /vcsInstances

GET /branches

This API is used to get branches by VCS adapter name and remote VCS url. it is only for GIT

Parameters:

Name	Description	Parameter Type	Data Type
adapterName	The name of the adapter	query	string
url		header	string
user		header	string
password		header	string
base64Cer	It is an optional parameter used on git https with ssl certificate support. for other usage, this can be Null.	query	string

Example to get branches by VCS adapter name and remote VCS url.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/branches?adaptername={adapterNa
me}&base64Cer={pem format certificate}"  -H "Content-Type:
application/json" -H "user: {user}" -H "password: {password}" -H
"url: {url}"
```

GET /branches/{name}

This API is used to get branches by VCS instance name. it is only for GIT.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to get branches by VCS instance name

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/branches/{vcsInstanceName}"  -H
"Content-Type: application/json"
```

POST /checkin/{name}

This API is used to checkin the resource into vcs.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
body		body	Model schema

Request Body

Example Value - Schema

```
{
“createdResources”:[
{
“resourcePath”: “string”,
“type”: “string”
}
],
“deletedResources”:[
{
“resourcePath”: “string”,
“type”: “string”
}
],
“updatedResources”:[
{
“resourcePath”: “string”,
“type”: “string”
}
],
“comment”: "string",
“email”: “string”,
“committerFullName”: “string”
}
```

Example to checkin

```
curl -X POST -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/checkin/{vcsInstanceName}}" -H
"Content-Type: application/json"  -d '{"comment": "my comment",
"createdResources":[{"resourcePath":"/users/composite/admin/abc/fo
o", "type":"PROCEDURE"}],
```

```
"updatedResources":[{"resourcePath":"/users/composite/admin/abc/ba
r", "type":"PROCEDURE"}],
"deletedResources":[{"resourcePath":"/users/composite/admin/abc/ka
y", "type":"PROCEDURE"}]}'
```

GET /connection

This API is used to test connection by vcsAdapter name, url, username, password.

Parameters:

Name	Description	Parameter Type	Data Type
adapterName	Name of the adapter	query	string
url		header	string
user		header	string
password		header	string
base64Cer	It is an optional parameter used on git https with ssl certificate support. for other usage, this can be Null.	query	string

Example to test connection by vcsAdapter name, url, username, password

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/connection?adapterName=svnAdapt
er&base64Cer={pem format certificate}"  -H "Content-Type:
application/json" -H "url:svn://172.23.5.76:3690" -H
"user:svnuser1" -H "password:foa23f9u"
```

GET /connection/{name}

This API is used to verify vcs url connection.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to verify vcs url connection

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/connection/{vcsInstanceName}"
-H "Content-Type: application/json"
```

GET /content/{name}

This API is used to get cmf content for special resource and revision in special vcsInstance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
resourceType		query	string
resourcePath		query	string
revision		query	string

Example to get cmf content for special resource and revision in special vcsInstance.

```
curl -X GET -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/content/{vcsInstanceName}?resou
rcePath=/users/composite/admin/abc/LookupProduct&resourceType=PROC
EDURE&revision=r109"  -H "Content-Type: application/json"
```

POST /discard/{name}

This API is used to discard local changes for special resource and get latest revision.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
includePrivileges		query	boolean
body		body	Model schema

Request Body

Example Value - Schema

```
{
“resources”:[
{
“resourcePath”: “string”,
“type”: “string”
}
]
}
```

Example to dicard local changes

```
curl -X POST -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/discard/{vcsInstanceName}?inclu
dePrivileges=true"  -H "Content-Type: application/json"  -d
'{"resources":[{"resourcePath":"/users/composite/admin/def/LookupP
roduct", "type":"PROCEDURE"}]}'
```

GET /enable

This API is used to check whether vcs feature is enabled.

Parameters:

None

Example to check whether vcs feature is enabled

```
curl -X GET  -u admin:admin
"http://localhost:9400/rest/vcs/v1/enable"  -H "Content-Type:
application/json"
```

POST /fetch/{name}

This API is used to fetch the resource subtree from a particular revision. For single resource(either a folder or a file), resource path and type should be provided. For vcs connection wide fetch, resource path and type should be set to null, (empty values for the resource path and type results in reader error currently).

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
revision		query	string
includePrivileges		query	boolean
body		body	Model schema

Request Body

Example Value - Schema

```
{
“resourcePath”: “string”,
“type”: “string”
}
```

Example to fetch the resource subtree from a particular revision.

```
curl -X POST -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/fetch/{vcsInstanceName}?revisio
n=r112&includePrivileges=true" -H "Content-Type: application/json"
-d”{"resourcePath":"/users/composite/admin/examples/view4",
"type":"TABLE"}’
```

GET /history/{name}

This API is used to get whole history or history for special resource in special vcsInstance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
resourceType		query	string
resourcePath		query	string

Example to get whole history or history for special resource in special vcsInstance.

```
curl -X GET -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/history/{vcsInstnace}[?resource
Path=/users/composite/admin/abc/LookupProduct&resourceType=PROCEDU
RE]"  -H "Content-Type: application/json"
```

GET /latestcontent/{name}

This API is used to get latest cmf content for special resource in special vcsInstance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
resourceType		query	string
resourcePath		query	string

Example to get latest cmf content for special resource in special vcsInstance.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/latestcontent/{vcsInstanceName}
?resourcePath=/users/composite/admin/abc/LookupProduct&resourceTyp
e=PROCEDURE"  -H "Content-Type: application/json"
```

GET /localcontent/{name}

This API is used to get local cmf content for special resource in special vcsInstance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
resourceType		query	string
resourcePath		query	string

Example to get local cmf content for special resource in special vcsInstance.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/localcontent/{vcsInstanceName}?
resourcePath=/users/composite/admin/abc/LookupProduct&resourceType
=PROCEDURE"  -H "Content-Type: application/json"
```

GET /root

This API is used to check whether special path is a root in vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
cis_path		query	string
rootType		query	string

Example to check whether special path is a root in vcs instance.

```
curl -X GET  -u admin:admin
"http://localhost:9400/rest/vcs/v1/root?cis_path=/users/composite/
admin/abc/kkkdd&rootType={type}"  -H "Content-Type:
application/json"
```

GET /root/{name}

This API is used to get all of roots for special vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to get all of roots for special vcs instance

```
curl -X GET  -u admin:admin
"http://localhost:9400/rest/vcs/v1/root/{vcsInstanceName}}" -H
"Content-Type: application/json"
```

POST /root/{name}

This API is used to add root for special vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
mode		query	string
rootType		query	string
includePriveleges		query	boolean
body		body	string

Example to add root for special vcs instance

```
curl -X POST -u admin:admin  --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/root/{vcsInstanceName}?mode={pu
sh/pull}&includePrivileges=true&rootType={type}"  -H "Content-Type:
application/json" -d "/users/composite/admin/gitabc"
```

DELETE /root/{name}

This API is used to delete root from special vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
cis_path		query	string
rootType		query	string

Example to delete root from special vcs instance

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/vcs/v1/root/{vcsInstanceName}?cis_path
=/users/composite/admin/abc&rootType={type}"  -H "Content-Type:
application/json"
```

POST /setCredential/{name}

This API is used to set credial for the vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string
body		body	Model schema

Request Body

Example Value - Schema

```
{
“username”: “string”,
“password”: “string”
}
```

Example to set credential

```
curl -X POST -u admin:admin --cookie "session9400={sessionToken}"
"http://localhost:9400/rest/vcs/v1/setCredential/{vcsInstanceName}
}" -H "Content-Type: application/json" -d '{"username":"gituser2",
"password":"foa23f9u"}'
```

GET /status/{name}

This API is used to get resource status for special vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to get resource status for special vcs instance

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/status/{vcsInstanceName}}" -H
"Content-Type: application/json"
```

GET /vcsAdapter/{adapter_name}

This API is used to get VCS adapter by adapter name.

Parameters:

Name	Description	Parameter Type	Data Type
adapterName	name of the adapter	path	string

Example to Get VCS adapter by adapter name

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsAdapter/{adapterName}"  -H
"Content-Type: application/json"
```

GET /vcsAdapters

This API is used to get VCS adapters.

Parameters:

None

Example to get VCS adapters

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsAdapters"  -H "Content-Type:
application/json"
```

POST /vcsInstance

This API is used to create vcs instance.

Parameters:

None

Request Body

Example Value - Schema

```
{
“cmdCheckIn”: “string”,
“cmdCheckOut: “string”,
“cmdAdd”: “string”,
“cmdRemove”: “string”,
“cmdRevert”: “string”,
“cmdStatus”: “string”,
“cmdDiff”: “string”,
“name”: “string,
“description”: “string”,
“providerId”: “string”,
“repository”: “string”,
“localWorkspace”: “string”,
“encryptionPassword”, “string”,
“branch”, “string”,
“ignoreEncryption”: “string”,
“base64Cer”: “string”,
“cmdHistory”: “string”
}
```

Example to create vcs instance

```
curl  -X POST -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsInstance"  -H "Content-Type:
application/json" -d '{"name":"vcssource1", "description": "dd",
"providerId":"svnAdapter", "repository":
"svn://172.23.5.76:3690/project6",  "branch":"master",
```

```
"encryptionPassword":"testPassword", "base64Cer":"Pem format
certificate" }'
```

GET /vcsInstance/{name}

This API is used to get vcs instance by name.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to get vcs instance by name

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsInstance/{vcsInstanceName}"
-H "Content-Type: application/json"
```

PUT /vcsInstance/{name}

This API is used to update vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Request Body

Example Value - Schema

```
{
“cmdCheckIn”: “string”,
“cmdCheckOut: “string”,
“cmdAdd”: “string”,
“cmdRemove”: “string”,
“cmdRevert”: “string”,
“cmdStatus”: “string”,
“cmdDiff”: “string”,
“name”: “string,
“description”: “string”,
“providerId”: “string”,
“repository”: “string”,
```

```
"localWorkspace": "string",
"encryptionPassword", "string",
"branch", "string",
"ignoreEncryption": "string",
"base64Cer": "string",
"cmdHistory": "string"
}
```

Example to update vcs instance

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsInstance/{vcsInstanceName}"
-H "Content-Type: application/json" -d '{"description": "dddddd",
"providerId":"svnAdapter", "repository": "svn://172.23.5.76:3690",
"branch":"master", "encryptionPassword":"testPassword",
"base64Cer":"Pem format certificate" }'
```

DELETE /vcsInstance/{name}

This API is used to delete vcs instance.

Parameters:

Name	Description	Parameter Type	Data Type
name		path	string

Example to delete vcs instance

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsInstance/{vcsInstanceName}"
-H "Content-Type: application/json"
```

GET /vcsInstances

This API is used to get all vcs instances.

Parameters:

None

Example to get all vcs instances

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/vcs/v1/vcsInstances" -H "Content-Type:
application/json"
```

Workload Management

The following operations can be performed with the Workload Rules:

- GET /enable
- PUT /enable
- GET /rules
- POST /rules
- PUT /rules
- DELETE /rules
- GET /rules/effective
- GET /rules/effective/member
- GET /rules/effective/member/resource

GET /enable

This API is used to get enabled status of workload management.

Parameters:

None

Example to get status of workload management

curl -X GET -u admin:admin "http://localhost:9400/rest/workload/v1/enable"

PUT /enable

This API is used to enable or disable workload management.

Parameters:

Name	Description	Parameter Type	Data Type
body		body	integer

Example to enable workload

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/workload/v1/enable" -H
"Content-Type:application/json" -d "1"
```

Example to disable workload

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/workload/v1/enable" -H
"Content-Type:application/json" -d "0"
```

GET /rules

This API is used to get workload rules.

Parameters:

Name	Description	Parameter Type	Data Type
ruleName		query	string

Example to get all workload rules that exist in the server.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules"
```

Example to get workload rule "rule1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules?ruleName=rule1"
```

POST /rules

This API is used to create workload rules by specifying the filters, resource and member assignments.

Parameters:

Name	Description	Parameter Type	Data Type
body	Workload rule	body	Array[#/definitions/WorkloadRuleBean]

Request Body

Example Value - Schema

```
{
“ruleName”: “string”,
“ruleType: “ROW_LIMIT”,
“maxRowCount”: 0,
“enabled”: true,
“annotation”: “string”,
“memberAssignments”: [
{
“domainName”: “string”,
“memberName”: “string”,
“group”: true
}
],
“resourceAssignments”: [
{
“resourcePath”: “string”,
“resourceType”: “string”
}
],
“memoryLimitPercentage”: 0,
“actionTypes”: “NOOP”
],
“emailActionData”: {
“from”: “string”,
“replyTo”: “string”,
“to”: [
“string”
],
“cc”: [
“string”
],
“bcc”: [
“string”
],
```

```
"subject": "string",
"content": "string"
},
"maxRequestTime" 0,
"maxRequestTimeUnit": "NANOSECONDS",
"newRuleName": "string",
"customMessage": "string"
}
]
```

Example to create workload rule "rule1"

```
curl -X POST -u admin:admin
"http://localhost:9400/rest/workload/v1/rules" -H
"Content-Type:application/json" -d
"[{\"ruleName\":\"rule1\",\"enabled\":\"true\",
\"actionTypes\":[\"NOOP\"], \"ruleType\":\"FULL_TABLE_SCAN\",
\"annotation\":\"Rule to disallow select *\",
\"memberAssignments\":[{\"domainName\":\"composite\",
\"memberName\":\"customuser\", \"group\":\"false\"}],
\"resourceAssignments\":[{\"resourcePath\":\"/services/databases/f
oo/customers\", \"resourceType\":\"TABLE\"}] }]"
```

PUT /rules

This API is used to update workload rules.

Parameters:

None

Request Body

Example Value - Schema

```
{
"ruleName": "string",
"ruleType: "ROW_LIMIT",
"maxRowCount": 0,
"enabled": true,
"annotation": "string",
"memberAssignments": [
{
"domainName": "string",
"memberName": "string",
"group": true
```

```
}
],
“resourceAssignments”: [
{
“resourcePath”: “string”,
“resourceType”: “string”
}
],
“memoryLimitPercentage”: 0,
“actionTypes”: “NOOP”
],
“emailActionData”: {
“from”: “string”,
“replyTo”: “string”,
“to”: [
“string”
],
“cc”: [
“string”
],
“bcc”: [
“string”
],
“subject”: “string”,
“content”: “string”
},
“maxRequestTime” 0,
“maxRequestTimeUnit”: “NANOSECONDS”,
“newRuleName”: “string”,
“customMessage”: “string”
}
]
```

Example to update workload rule "/policy/workload/rule1" enabled status and rule type

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/workload/v1/rules" -H
"Content-Type:application/json" -d
"[{\"ruleName\":\"rule1\",\"enabled\":\"false\",
\"actionTypes\":[\"NOOP\"], \"ruleType\":\"ROW_LIMIT\",
\"annotation\":\"Rule to limit rows\", \"maxRowCount\":\"10\",
\"memberAssignments\":[{\"domainName\":\"composite\",
\"memberName\":\"customuser\", \"group\":\"false\"}],
\"resourceAssignments\":[{\"resourcePath\":\"/services/databases/f
oo/customers\", \"resourceType\":\"TABLE\"}] }]"
```

Example to rename workload rule "/policy/workload/rule1" to "/policy/workload/rule2"

```
curl -X PUT -u admin:admin
"http://localhost:9400/rest/workload/v1/rules" -H
"Content-Type:application/json" -d
"[{\"ruleName\":\"rule1\",\"newRuleName\":\"rule2\",
\"enabled\":\"true\",  \"actionTypes\":[\"NOOP\"],
\"ruleType\":\"ROW_LIMIT\", \"annotation\":\"Rule to limit rows\",
\"maxRowCount\":\"10\",
\"memberAssignments\":[{\"domainName\":\"composite\",
\"memberName\":\"customuser\", \"group\":\"false\"}],
\"resourceAssignments\":[{\"resourcePath\":\"/services/databases/f
oo/customers\", \"resourceType\":\"TABLE\"}] }]"
```

DELETE /rules

This API is used to delete workload rules.

Parameters:

None

Request Body

Example Value - Schema

```
[
“string”
]
```

Example to delete workload rules

```
curl -X DELETE -u admin:admin
"http://localhost:9400/rest/workload/v1/rules" -H
"Content-Type:application/json" -d "[\"rule1\", \"rule2\"]"
```

GET /rules/effective

This API is used to get effective rules of a resource.

Parameters:

Name	Description	Parameter Type	Data Type
resourceIds	resource ids	query	Array[integer]

Example to get effective rules of a resource

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules/effective?resourceId
s=10128&resourceIds=10138"
```

GET /rules/effective/member

This API is used to get workload rules of a user/group.

Parameters:

Name	Description	Parameter Type	Data Type
memberDomain	The domain of the user.	query	string
memberName	The name of the user	query	string

Example to get all workload rules that exist in the server.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules/effective/member"
```

Example to get effective workload rules of a user "u1"

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules/effective/member?mem
berDomain=composite&memberName=u1"
```

GET /rules/effective/member/resource

This API is used to get effective workload rules of a user/group for specified resource path types.

Parameters:

Name	Description	Parameter Type	Data Type
memberDomain	The domain name of user/group.	query	string
memberName	The name of the user/group	query	string
resourcePath	Resource path	query	string
resourceType	Resource type	query	string

Example to get all workload rules that exist in the server.

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules/effective/member/res
ource" -H "Content-Type:application/json" -d "[]"
```

Example to get all effective workload rules for specified resource path types

```
curl -X GET -u admin:admin
"http://localhost:9400/rest/workload/v1/rules/effective/member/res
ource?memberDomain=composite&memberName=u1&resourcePath=/services/
databases/foo/customers&resourceType=TABLE""
```

Auth

You can issue/request/revoke a token for authentication and authorization using API calls. The operations that can be performed are:

POST/auth/refreshToken, page 492

DELETE/auth/revokeToken, page 493

POST/auth/requestToken, page 493

POST/auth/spnegoRequestToken, page 494

POST/auth/refreshToken

This API is used to issue a new token using refresh token. It is assumed that the refresh token is in cookie.

Parameters

None.

Example to issue a new token using the refresh token in the user's cookie

```
curl -X POST -b cookie.txt
"http://localhost:9400/rest/auth/v1/refreshToken"
```

DELETE/auth/revokeToken

Assuming the refresh token is in cookie, this API is used to revoke the token and terminate the session.

Parameters

None

Example to revoke the refresh token in the cookie.

```
curl -X DELETE -b cookie.txt
"http://localhost:9400/rest/auth/v1/revokeToken"
```

POST/auth/requestToken

This API is used to request a token.

Parameters

None

Request Body

Example Value - Schema

```
[
{
"string"
}
]
```

Example to request a token for user "admin" in the request body

```
curl -X POST -u admin:admin -c cookie.txt
"http://localhost:9400/rest/auth/v1" -H
"Content-Type:application/json" -d "{\"appId\":\"Contrail\"}"
```

POST/auth/spnegoRequestToken

This API is used to request a token using Kerberos (GSS) authentication.

Parameters

None

Request Body

Example Value - Schema

```
[
{
"string"
}
]
```

Example to request a token for a Kerberos user in the request body.

```
curl -X POST --negotiate -u : -c cookie.txt
"http://localhost:9400/rest/auth/v1/spnegoRequestToken" -H
"Content-Type:application/json" -d "{\"appId\":\"Contrail\"}"
```

TIBCO Product Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join the TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO Data Virtualization page.

- Users
  - TDV Getting Started Guide
  - TDV User Guide
  - TDV Web UI User Guide
  - TDV Client Interfaces Guide
  - TDV Tutorial Guide
  - TDV Northbay Example
- Administration
  - TDV Installation and Upgrade Guide
  - TDV Administration Guide
  - TDV Active Cluster Guide
  - TDV Security Features Guide
- Data Sources
  - TDV Adapter Guides
  - TDV Data Source Toolkit Guide (Formerly Extensibility Guide)
- References
  - TDV Reference Guide
  - TDV Application Programming Interface Guide

- **Other**

  TDV Business Directory Guide

  TDV Discovery Guide

- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, visit TIBCO Community.

Legal and Third-Party Notices