Informatica® Multidomain MDM
10.3 HotFix 1

Business Entity Services Guide

Informatica Multidomain MDM Business Entity Services Guide
10.3 HotFix 1
June 2020

Publication Date: 2020-06-27

Table of Contents

Preface 7

Informatica Resources 7

- Informatica Network 7
- Informatica Knowledge Base 7
- Informatica Documentation 7
- Informatica Product Availability Matrices 8
- Informatica Velocity 8
- Informatica Marketplace 8
- Informatica Global Customer Support 8

Chapter 1: Introduction to Business Entity Services 9

Business Entity Services Overview 9

Business Entity Services 10

- ReadBE Business Entity Service 10
- WriteBE Business Entity Service 10
- SearchBE Business Entity Service 10

Business Entity Service Endpoints 11

- Enterprise JavaBeans Endpoint for Business Entity Services 11
- REST Endpoint for Business Entity Services 11
- REST and EJB Business Entity Service Calls 11
- SOAP Endpoint for Business Entity Services 12

Identifying a Root Record 12

Security and Data Filters 12

Chapter 2: Enterprise Java Bean Business Entity Service Calls 13

Enterprise Java Bean Business Entity Service Calls Overview 13

Java Code Example with Standard SDO Classes 13

Java Code Example with Generated SDO Classes 17

Chapter 3: Representational State Transfer Business Entity Service Calls 21

REST APIs for Business Entity Services Overview 21

Supported REST Methods 22

Authentication Method 22

Authentication Cookies for Login from Third-Party Applications 22

Web Application Description Language File 23

REST Uniform Resource Locator 24

Header and Body Configuration 24

- Request Header 25
- Request Body 25

Standard Query Parameters 27

Formats for Dates and Time in UTC. . . . 27
Configuring WebLogic to Run Business Entity Service REST Calls. . . . 28
Viewing Input and Output Parameters. . . . 29
JavaScript Template. . . . 29
JavaScript Example. . . . 30
REST API Reference for Business Entity Services 32
Get Metadata 32
List Metadata. . . . 35
List Match Columns. . . . 40
Read Record. . . . 41
Create Record. . . . 47
Update Record. . . . 50
Delete Record. . . . 53
List Record. . . . 54
Search Record. . . . 56
Suggester. . . . 61
SearchQuery. . . . 62
SearchMatch. . . . 66
Get BPM Metadata. . . . 71
List Tasks. . . . 72
Read Task. . . . 77
Create Task. . . . 79
Update Task. . . . 81
Task Complete. . . . 84
Execute Task Action. . . . 86
List Assignable Users. . . . 89
List File Metadata. . . . 89
Create File Metadata. . . . 90
Get File Metadata. . . . 91
Update File Metadata. . . . 92
Upload File Content. . . . 93
Get File Content. . . . 94
Delete File. . . . 94
Preview Promote. . . . 95
Promote. . . . 97
Delete Pending. . . . 97
Preview Merge. . . . 98
Update Pending Merge. . . . 101
Pending Merge. . . . 103
PromoteMerge. . . . 104
Merge Records. . . . 105
Unmerge Records. . . . 107

Read a Relationship. . . . 108
Create a Relationship. . . . 110
Update a Relationship. . . . 112
Delete a Relationship. . . . 113
Get Related Records. . . . 114
Read Matched Records. . . . 118
Update Matched Records. . . . 119
Delete Matched Records. . . . 120
Get Record History Events. . . . 121
Get Event Details. . . . 124
Get DaaS Metadata. . . . 126
DaaS Search 126
DaaS Read. . . . 131
WriteMerge. . . . 133
DaaS Import. . . . 134
DaaS Update. . . . 137

Chapter 4: Simple Object Access Protocol Business Entity Service Calls. . . . 140

Simple Object Access Protocol Calls for Business Entity Services. . . . 140
Authentication method. . . . 141
Authentication Cookies for Login from Third-Party Applications. . . . 141
Web Services Description Language File. . . . 142
SOAP URL. . . . 143
SOAP Requests and Responses. . . . 144
Viewing Input and Output Parameters. . . . 145
SOAP API Reference. . . . 146
Sample SOAP Request and Response. . . . 147

Chapter 5: Services for Cross-reference Records and BVT Calculations. 149

Overview of Services for Cross-reference Records and BVT Calculations. . . . 149
Getting Cross-reference Data and Investigating BVT Calculations. . . . 149
Get Cross-reference Records. . . . 150
Determine Contributors to the Master Record. . . . 150
Get the Trust Scores of Contributing Cross-reference Record Fields. . . . 151
Getting the Trust Scores of All Cross-reference Record Fields. . . . 151
Get Information about Source Systems. . . . 152
Get Information about Source Systems Example. . . . 152
Filtering and Paginating Responses. . . . 153
Filtering Request Examples. . . . 153
Establish the Best Version of the Truth. . . . 154
Select the Correct Contributing Field. . . . 154
Select the Correct Contributing Field Example. . . . 154
Write the Correct Value to the Master Record. . . . 155

Write the Correct Value to the Master Record Example. . . . 156
Remove Mismatched Source Data. . . . 157
Remove Mismatched Source Data Example. . . . 158
Unmerge Response. . . . 159

Chapter 6: Supporting Corporate Linkage Service. . . . 160

Overview. . . . 160
Business Entity Services for DaaS Import and Update. . . . 160
Configuring Linkage Support. . . . 161
Custom Application for Linkage Data Splitting. . . . 161

Chapter 7: External Calls to Cleanse, Analyze, and Transform Data. . . . 162

Overview. . . . 162
Supported Events. . . . 163
How to Configure External Calls. . . . 163
Example: Custom Validation and Logic for Business Entity Services. . . . 164
Prerequisites. . . . 164
Step 1. Test Custom Validation. . . . 164
Step 2. Test Custom Logic. . . . 165

Appendix A: Using REST APIs to Add Records. . . . 170

Using REST APIs to Add Records Overview. . . . 170
Person Business Entity Structure. . . . 171
Step 1. Get Information about the Schema. . . . 171
Get Metadata Response. . . . 172
Step 2. Create a Record. . . . 177
Create Record Response. . . . 178
Step 3. Read the Record. . . . 179
Read the Record Response. . . . 180

Appendix B: Using REST APIs to Upload Files. . . . 184

Using REST APIs to Upload Files Overview. . . . 184
REST APIs for Files. . . . 184
File Components. . . . 185
Storage Types. . . . 185
Attaching Files to Records. . . . 186
Attaching Files to Tasks. . . . 188
Uploading Resource Bundle Files. . . . 190

Index. . . . 192

Preface

Welcome to the *Multidomain MDM Business Entity Services Guide*. This guide explains how to make business entity service calls to operate on business entities in the Informatica® MDM Hub.

This guide is intended for technical specialists who are responsible for configuring custom user interfaces to make business entity service calls to the MDM Hub.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction to Business Entity Services

This chapter includes the following topics:

- Business Entity Services Overview, 9
- Business Entity Services, 10
- Business Entity Service Endpoints, 11
- Identifying a Root Record, 12
- Security and Data Filters, 12

Business Entity Services Overview

A business entity service is a set of operations that run MDM Hub code to create, update, delete, and search for base object records in a business entity. You can develop a custom user interface that can run Java code or JavaScript code to make business entity service calls.

For example, you can create a business entity service to augment a supplier record with Dun and Bradstreet data. Configure the business entity service to take the supplier record as an input, retrieve some information from Dun and Bradstreet, update the record, and then output the updated supplier record.

Base objects in a business entity have the following business entity services:

Read

Each business entity has a business entity service to perform a read operation.

Write

Each business entity has a business entity service to perform a write operation.

Search

Any business entity that has searchable fields has a business entity service to perform a search operation.

For example, a Person business entity has searchable fields. The MDM Hub generates a ReadPerson, WritePerson, and SearchPerson business entity service. The read, write, and search business entity service steps allow you to read, create, update, delete, and search for records in a business entity.

Business Entity Services

A business entity service performs an operation. You can use the ReadBE, WriteBE, and SearchBE business entity services.

A business entity service has service steps. An incoming request passes through each service step. The output of one step is an input for the next step. The output of a step can pass information to the input of the following step. All business entity service steps run as one Enterprise Java Bean call in a single transaction. The MDM Hub handles exceptions.

Note: Before you use the business entity services, validate the Operational Reference Store.

ReadBE Business Entity Service

The ReadBE business entity service reads data from a base object record in a business entity.

You can specify pagination parameters with the ReadBE step to set the number of records to return and the page of results to view.

The results of the ReadBE service do not include soft-deleted records.

If you do not pass the EffectiveDate parameter in the business entity service request, the MDM Hub assumes a NULL effective date and the business entity service reads data from the base object. If you pass the EffectiveDate parameter, the MDM Hub calculates the best version of the truth from cross-reference records and the read business entity service returns the up-to-date best version of the truth.

WriteBE Business Entity Service

The WriteBE business entity service can update the data in a business entity element, create a child business entity element, or delete child business entity elements.

Note: The WriteBE business entity service uses existing trust settings to calculate trust on base objects. You cannot perform a trust override with this service.

Optional Parameters

The following table describes optional parameters that you can use with the WriteBE business entity service:

Parameter	Description
recordState	Set the record state to ACTIVE, PENDING, or DELETED. **Note:** When you set `recordState=ACTIVE` and you run the service on a soft-deleted record, the service restores the record to the active state.
EffectivePeriod	Specify an effective period. If you do not pass the `EffectivePeriod` parameter, the MDM Hub assumes an unbounded period. The MDM Hub does not check that there is alignment for effective periods between the root objects and child objects. When you create or update records, ensure the effective periods of the parent record and child records align.

SearchBE Business Entity Service

Use the SearchBE business entity service to search for a root record in a business entity.

For information on configuring business entities for smart search, see the *Multidomain MDM Configuration Guide*.

Business Entity Service Endpoints

You can access business entity services through an Enterprise JavaBeans (EJB) endpoint, a Representational State Transfer (REST) endpoint, or a Simple Object Access Protocol (SOAP) endpoint.

REST endpoints are built upon an EJB standpoint. The REST business entity service configuration defines how the REST URLs are mapped to the EJB business entity service calls.

Enterprise JavaBeans Endpoint for Business Entity Services

The Enterprise JavaBeans (EJB) endpoint is the underlying endpoint for all types of business entity service calls. All other endpoints are mapped to the EJB endpoint

Business entity services are exposed as a stateless EJB. A stateless EJB container can pool instances, allocate instances, and apply load balancing strategies to distribute the load across different servers within the domain.

An EJB endpoint accepts a user name and password for authentication.

REST Endpoint for Business Entity Services

Representational State Transfer Endpoint (REST) calls make business entity services available as web services.

A Web Application Description Language (WADL) files contain XML descriptions of the REST web services, all the REST URLs, and all REST parameters. The MDM Hub generates a WADL file for each Operational Reference Store.

You can download the WADL files for each Operational Reference Store from the following location:

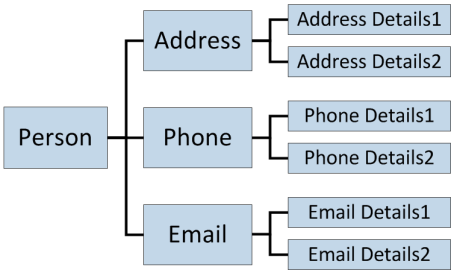
```
http://<host>:<port>/cmx/csfiles
```

REST and EJB Business Entity Service Calls

When you make a business entity service call, you might specify certain child branches instead of requesting the whole business entity.

For example, you want to perform a read operation on a business entity that has a Person root node and multiple child branches. The Person base object has Address, Phone, and Email child base objects. Each child base object has two grandchild base objects.

The following image shows the structure of a business entity with multiple branches:



You can read from multiple child branches at various depths in a single request. For example, you can read Person, Phone, Phone Details1, Phone Details 2, Email, and Email Details 2 in a single request.

The following URL sample shows how to make a REST read request to get the Person record with row ID 1242, in addition to the Address Details 1 and Email child records:

```
http://localhost:8080/cmx/cs/localhost-ORCL-DS_UI1/Person/1242?children=Address/
Address_Details_1,Email
```

SOAP Endpoint for Business Entity Services

Simple Object Access Protocol (SOAP) endpoint calls make business entity services available as web services.

Web Services Description Language (WSDL) files contain the XML descriptions of the web services, formats of the SOAP requests and responses, and all parameters. The MDM Hub generates a WSDL file for each Operational Reference Store.

Identifying a Root Record

You can use one of the following approaches to identify a root record:

- rowid. The value in the ROWID_OBJECT column of the record.
- systemName and sourceKey. The systemName is the name of the system to which the record belongs. The sourceKey is the value in the PKEY_SRC_OBJECT column of the record.
- Global Business Identifier (GBID) of an object. A GBID can be a compound value, in which case you must pass all the values.

  Note: The GBID approach works only with the ReadBE service.

The following sample code uses the systemName and sourceKey to identify a record:

```
String systemName = "SFA";

Properties config = new Properties();
config.put(SiperianClient.SIPERIANCLIENT_PROTOCOL, EjbSiperianClient.PROTOCOL_NAME);
CompositeServiceClient client = CompositeServiceClient.newCompositeServiceClient(config);
CallContext callContext = new CallContext(orsId, user, pass);
helperContext = client.getHelperContext(callContext);
DataFactory dataFactory = helperContext.getDataFactory();

//String personRowId = "1097";
String pkeySrcObject = "CST1379";

//Set custom key pkey
pkey = (Key) dataFactory.create(Key.class);
pkey.setSystemName(systemName);
pkey.setSourceKey(val);
writePerson.setKey(pkey);
```

Security and Data Filters

When base objects and resources have user role privileges, the business entity inherits those privileges. To access business entity records, the user role must have the appropriate privileges to the root base object for the business entity as well as other resources.

The business entity services also inherit any data filters that you set on business entity fields.

For more information about security and data filters, see the *Multidomain MDM Provisioning Tool Guide*.

CHAPTER 2

Enterprise Java Bean Business Entity Service Calls

This chapter includes the following topics:

- Enterprise Java Bean Business Entity Service Calls Overview, 13
- Java Code Example with Standard SDO Classes, 13
- Java Code Example with Generated SDO Classes, 17

Enterprise Java Bean Business Entity Service Calls Overview

You can make Enterprise Java Bean (EJB) business entity service calls to create, update, delete, and search for base object records in a business entity. You can create Java code to run EJB business entity service calls.

You can create Java code based on standard Service Data Objects (SDO) classes or you can create Java code based on java classes that the MDM Hub generates based on the business entity and business entity services configuration.

Java Code Example with Standard SDO Classes

The example shows Java code to run Enterprise Java Bean (EJB) calls based on standard Service Data Objects (SDO) classes.

The example is in the following file in the resource kit: `C:\<MDM Hub installation directory>\hub\resourcekit\samples\COS\source\java\com\informatica\mdm\sample\cs\DynamicSDO.java`

The following Java code is based on standard SDO classes and runs EJB business entity service calls to create a Person base object record, add multiple child records, delete one child record, and then delete the person record and all child records:

```
package com.informatica.mdm.sample.cs;

import com.informatica.mdm.cs.CallContext;
import com.informatica.mdm.cs.api.CompositeServiceException;
import com.informatica.mdm.cs.client.CompositeServiceClient;
```

```
import com.siperian.sif.client.EjbSiperianClient;
import com.siperian.sif.client.SiperianClient;
import commonj.sdo.DataObject;
import commonj.sdo.Property;
import commonj.sdo.Type;
import commonj.sdo.helper.DataFactory;
import commonj.sdo.helper.HelperContext;

import java.io.PrintStream;
import java.util.Arrays;
import java.util.Properties;

public class DynamicSDO {

    public static void main(String[] args) throws CompositeServiceException {

        if(args.length != 3) {
            System.err.println("USAGE: DynamicSDO <ors> <user> <pass>");
            return;
        }

        new DynamicSDO(args[0], args[1], args[2]).execute();
    }


    private String orsId;
    private String user;
    private String pass;
    private HelperContext helperContext;
    private PrintStream out = System.out;

    public DynamicSDO(String orsId, String user, String pass) {
        this.orsId = orsId;
        this.user = user;
        this.pass = pass;
    }

    public void execute() throws CompositeServiceException {

        String systemName = "Admin";

        Properties config = new Properties();
        config.put(SiperianClient.SIPERIANCLIENT_PROTOCOL,
EjbSiperianClient.PROTOCOL_NAME);
        CompositeServiceClient client =
CompositeServiceClient.newCompositeServiceClient(config);

        CallContext callContext = new CallContext(orsId, user, pass);

        helperContext = client.getHelperContext(callContext);

        DataFactory dataFactory = helperContext.getDataFactory();

        // types for Read requests
        Type coFilterType = helperContext.getTypeHelper().getType("urn:cs-
base.informatica.mdm", "CoFilter");
        Type coFilterNodeType = helperContext.getTypeHelper().getType("urn:cs-
base.informatica.mdm", "CoFilterNode");
        Type keyType = helperContext.getTypeHelper().getType("urn:cs-
base.informatica.mdm", "Key");

        // ReadCO & WriteCO request types
        Type readPersonType = helperContext.getTypeHelper().getType("urn:cs-
ors.informatica.mdm", "ReadPerson");
        Type writePersonType = helperContext.getTypeHelper().getType("urn:cs-
ors.informatica.mdm", "WritePerson");

        // 1. Create new person
        DataObject createPerson = dataFactory.create(writePersonType);
        DataObject createPersonParameters = createPerson.createDataObject("parameters");
        createPersonParameters.setString("systemName", systemName);
```

```
DataObject person = createPerson.createDataObject("object");

person.getChangeSummary().beginLogging();

DataObject personRoot = person.createDataObject("Person");
personRoot.setString("firstName", "John");
personRoot.setString("lastName", "Smith");

person.getChangeSummary().endLogging();

dump("*** CREATE NEW PERSON ...", createPerson);

DataObject createPersonResponse = client.process(callContext, createPerson);

dump("*** PERSON CREATED:", createPersonResponse);

String personRowId = createPersonResponse.getString("object/Person/rowidObject");

DataObject readPerson = dataFactory.create(readPersonType);
DataObject readPersonParameters = readPerson.createDataObject("parameters");
DataObject coFilter = readPersonParameters.createDataObject("coFilter");
DataObject coFilterNode = coFilter.createDataObject("object");
coFilterNode.set("name", "Person");
DataObject key = coFilterNode.createDataObject("key");
key.set("rowid", personRowId);

dump("*** READ CREATED PERSON...", readPerson);

DataObject readPersonResponse = client.process(callContext, readPerson);

dump("*** READ RESULT:", readPersonResponse);

person = readPersonResponse.getDataObject("object");
person.detach();

person.getChangeSummary().beginLogging();

personRoot = person.getDataObject("Person");
// add new 'one' child
DataObject genderCd = personRoot.createDataObject("genderCd");
genderCd.setString("genderCode", "M");

// add two 'many' children
DataObject phonePager = personRoot.createDataObject("TelephoneNumbers");
Property item = phonePager.getInstanceProperty("item");
Type phoneType = item.getType();

DataObject phone1 = dataFactory.create(phoneType);
phone1.setString("phoneNumber", "111-11-11");
DataObject phone2 = dataFactory.create(phoneType);
phone2.setString("phoneNumber", "222-22-22");

phonePager.setList(item, Arrays.asList(phone1, phone2));

person.getChangeSummary().endLogging();

DataObject updatePerson = dataFactory.create(writePersonType);
updatePerson.setDataObject("object", person);
DataObject updatePersonParameters = updatePerson.createDataObject("parameters");
updatePersonParameters.setString("systemName", systemName);
updatePersonParameters.setString("interactionId", "");

dump("*** UPDATE PERSON...", updatePerson);

DataObject updatePersonResponse = client.process(callContext, updatePerson);

dump("*** PERSON UPDATED:", updatePersonResponse);

coFilterNode.set("depth", 3);

readPersonParameters.setBoolean("readSystemFields", true);
```

```
dump("*** READ UPDATED PERSON WITH CHILDREN...", readPerson);

readPersonResponse = client.process(callContext, readPerson);

dump("*** READ RESULT:", readPersonResponse);

person = readPersonResponse.getDataObject("object");
person.detach();

person.getChangeSummary().beginLogging();

genderCd = person.getDataObject("Person").createDataObject("genderCd");
genderCd.setString("genderCode", "F");

// delete one phone
DataObject phoneItem = person.getDataObject("Person/TelephoneNumbers/item[1]");
phoneItem.delete();

person.getChangeSummary().endLogging();

DataObject deletePhone = dataFactory.create(writePersonType);
deletePhone.setDataObject("object", person);
DataObject deletePhoneParameters = deletePhone.createDataObject("parameters");
deletePhoneParameters.setString("systemName", systemName);

dump("*** DELETE CHILD...", deletePhone);

DataObject deletePhoneResponse = client.process(callContext, deletePhone);

dump("*** CHILD DELETED:", deletePhoneResponse);

readPersonParameters.setBoolean("readSystemFields", false);

dump("*** READ PERSON AFTER CHILD WAS DELETEED...", readPerson);

readPersonResponse = client.process(callContext, readPerson);

dump("*** READ RESULT:", readPersonResponse);

person = readPersonResponse.getDataObject("object");
person.detach();

person.getChangeSummary().beginLogging();

person.getDataObject("Person").detach();

person.getChangeSummary().endLogging();

DataObject deletePerson = dataFactory.create(writePersonType);
deletePerson.setDataObject("object", person);
DataObject deletePersonParameters = deletePerson.createDataObject("parameters");
deletePersonParameters.setString("systemName", systemName);

dump("*** DELETE PERSON...", deletePerson);

DataObject deletePersonResponse = client.process(callContext, deletePerson);

dump("*** PERSON DELETED:", deletePersonResponse);

dump("*** TRY TO READ PERSON AFTER DELETE", readPerson);

try {
    readPersonResponse = client.process(callContext, readPerson);

    dump("*** READ RESULT:", readPersonResponse);
} catch (CompositeServiceException e) {
    out.println("*** READ RESULT: " + e.getLocalizedMessage());
}
```

```
    }

    private void dump(String title, DataObject dataObject) {
        String xml = helperContext.getXMLHelper().save(
                dataObject,
                dataObject.getType().getURI(),
                dataObject.getType().getName());
        out.println(title);
        out.println(xml);
        out.println();
    }

}
```

Java Code Example with Generated SDO Classes

The example shows Java code to run Enterprise Java Bean (EJB) calls based on Java classes that the MDM Hub generates based on the business entity and business entity services configuration.

The example is in the following file in the resource kit: `C:\<MDM Hub installation directory>\hub\resourcekit\samples\COS\source\java\com\informatica\mdm\sample\cs\GeneratedSDO.java`

The following Java code is based on generated classes and runs EJB business entity service calls to create a person base object record, add multiple child records, delete one child record, and then delete the person record and all child records:

```
package com.informatica.mdm.sample.cs;

import com.informatica.mdm.cs.CallContext;
import com.informatica.mdm.cs.api.CompositeServiceException;
import com.informatica.mdm.cs.client.CompositeServiceClient;
import com.informatica.mdm.sdo.cs.base.CoFilter;
import com.informatica.mdm.sdo.cs.base.CoFilterNode;
import com.informatica.mdm.sdo.cs.base.Key;
import com.siperian.sif.client.EjbSiperianClient;
import com.siperian.sif.client.SiperianClient;
import commonj.sdo.DataObject;
import commonj.sdo.helper.DataFactory;
import commonj.sdo.helper.HelperContext;
import mdm.informatica.co_ors.*;
import mdm.informatica.cs_ors.*;

import java.io.PrintStream;
import java.util.Arrays;
import java.util.Properties;

public class GeneratedSDO {

    public static void main(String[] args) throws CompositeServiceException {

        if(args.length != 3) {
            System.err.println("USAGE: GeneratedSDO <ors> <user> <pass>");
            return;
        }

        new GeneratedSDO(args[0], args[1], args[2]).execute();
    }


    private String orsId;
    private String user;
    private String pass;
    private HelperContext helperContext;
    private PrintStream out = System.out;
```

```
    public GeneratedSDO(String orsId, String user, String pass) {
        this.orsId = orsId;
        this.user = user;
        this.pass = pass;
    }

    public void execute() throws CompositeServiceException {

        String systemName = "Admin";

        Properties config = new Properties();
        config.put(SiperianClient.SIPERIANCLIENT_PROTOCOL,
EjbSiperianClient.PROTOCOL_NAME);
        CompositeServiceClient client =
CompositeServiceClient.newCompositeServiceClient(config);

        CallContext callContext = new CallContext(orsId, user, pass);

        helperContext = client.getHelperContext(callContext);

        DataFactory dataFactory = helperContext.getDataFactory();

        // 1. Create new person
        WritePerson createPerson = (WritePerson)dataFactory.create(WritePerson.class);
        WritePersonParameters createPersonParameters =
(WritePersonParameters)dataFactory.create(WritePersonParameters.class);
        createPersonParameters.setSystemName(systemName);
        createPerson.setParameters(createPersonParameters);

        Person person = (Person)dataFactory.create(Person.class);
        createPerson.setObject(person);

        person.getChangeSummary().beginLogging();

        PersonRoot personRoot = (PersonRoot)dataFactory.create(PersonRoot.class);
        personRoot.setFirstName("John");
        personRoot.setLastName("Smith");
        person.setPerson(personRoot);

        person.getChangeSummary().endLogging();

        dump("*** CREATE NEW PERSON ...", createPerson);

        WritePersonReturn createPersonResponse =
(WritePersonReturn)client.process(callContext, (DataObject)createPerson);

        dump("*** PERSON CREATED:", createPersonResponse);

        String personRowId =
createPersonResponse.getObject().getPerson().getRowidObject();

        Key key = (Key)dataFactory.create(Key.class);
        key.setRowid(personRowId);
        CoFilterNode coFilterNode = (CoFilterNode)dataFactory.create(CoFilterNode.class);
        coFilterNode.setName(Person.class.getSimpleName());
        coFilterNode.setKey(key);
        CoFilter coFilter = (CoFilter)dataFactory.create(CoFilter.class);
        coFilter.setObject(coFilterNode);
        ReadPersonParameters readPersonParameters =
(ReadPersonParameters)dataFactory.create(ReadPersonParameters.class);
        readPersonParameters.setCoFilter(coFilter);

        ReadPerson readPerson = (ReadPerson)dataFactory.create(ReadPerson.class);
        readPerson.setParameters(readPersonParameters);

        dump("*** READ CREATED PERSON...", readPerson);

        ReadPersonReturn readPersonResponse =
(ReadPersonReturn)client.process(callContext, (DataObject)readPerson);
```

```
        dump("*** READ RESULT:", readPersonResponse);

        person = readPersonResponse.getObject();
        ((DataObject)person).detach();

        person.getChangeSummary().beginLogging();

        personRoot = person.getPerson();
        // add new 'one' child
        LUGenderLookup genderCd =
(LUGenderLookup)dataFactory.create(LUGenderLookup.class);
        genderCd.setGenderCode("M");
        personRoot.setGenderCd(genderCd);

        // add two 'many' children
        PersonTelephoneNumbersPager phonePager =
(PersonTelephoneNumbersPager)dataFactory.create(PersonTelephoneNumbersPager.class);

        PersonTelephoneNumbers phone1 =
(PersonTelephoneNumbers)dataFactory.create(PersonTelephoneNumbers.class);
        phone1.setPhoneNumber("111-11-11");
        PersonTelephoneNumbers phone2 =
(PersonTelephoneNumbers)dataFactory.create(PersonTelephoneNumbers.class);
        phone2.setPhoneNumber("222-22-22");

        phonePager.setItem(Arrays.asList(phone1, phone2));
        personRoot.setTelephoneNumbers(phonePager);

        person.getChangeSummary().endLogging();

        WritePerson updatePerson = (WritePerson)dataFactory.create(WritePerson.class);
        updatePerson.setObject(person);
        WritePersonParameters updatePersonParameters =
(WritePersonParameters)dataFactory.create(WritePersonParameters.class);
        updatePersonParameters.setSystemName(systemName);
        updatePersonParameters.setInteractionId("");
        updatePerson.setParameters(updatePersonParameters);

        dump("*** UPDATE PERSON...", updatePerson);

        WritePersonReturn updatePersonResponse =
(WritePersonReturn)client.process(callContext, (DataObject)updatePerson);

        dump("*** PERSON UPDATED:", updatePersonResponse);

        coFilterNode.setDepth(3);

        readPersonParameters.setReadSystemFields(true);

        dump("*** READ UPDATED PERSON WITH CHILDREN (with system fields)...",
readPerson);

        readPersonResponse = (ReadPersonReturn)client.process(callContext,
(DataObject)readPerson);

        dump("*** READ RESULT:", readPersonResponse);

        person = readPersonResponse.getObject();
        ((DataObject)person).detach();

        person.getChangeSummary().beginLogging();

        // delete one phone
        person.getPerson().getTelephoneNumbers().getItem().remove(0);

        // change gender
        genderCd = (LUGenderLookup)dataFactory.create(LUGenderLookup.class);
        genderCd.setGenderCode("F");
        personRoot.setGenderCd(genderCd);

        person.getChangeSummary().endLogging();
```

```
        WritePerson deletePhone = (WritePerson)dataFactory.create(WritePerson.class);
        deletePhone.setObject(person);
        WritePersonParameters deletePhoneParameters =
(WritePersonParameters)dataFactory.create(WritePersonParameters.class);
        deletePhoneParameters.setSystemName(systemName);
        deletePhone.setParameters(deletePhoneParameters);

        dump("*** DELETE CHILD...", deletePhone);

        WritePersonReturn deletePhoneResponse =
(WritePersonReturn)client.process(callContext, (DataObject)deletePhone);

        dump("*** CHILD DELETED:", deletePhoneResponse);

        readPersonParameters.setReadSystemFields(false);

        dump("*** READ PERSON AFTER CHILD WAS DELETEED...", readPerson);

        readPersonResponse = (ReadPersonReturn)client.process(callContext,
(DataObject)readPerson);

        dump("*** READ RESULT:", readPersonResponse);

        person = readPersonResponse.getObject();
        ((DataObject)person).detach();

        person.getChangeSummary().beginLogging();

        ((DataObject)person.getPerson()).delete();

        person.getChangeSummary().endLogging();

        WritePerson deletePerson = (WritePerson)dataFactory.create(WritePerson.class);
        deletePerson.setObject(person);
        WritePersonParameters deletePersonParameters =
(WritePersonParameters)dataFactory.create(WritePersonParameters.class);
        deletePersonParameters.setSystemName(systemName);
        deletePerson.setParameters(deletePersonParameters);

        dump("*** DELETE PERSON...", deletePerson);

        WritePersonReturn deletePersonResponse =
(WritePersonReturn)client.process(callContext, (DataObject)deletePerson);

        dump("*** PERSON DELETED:", deletePersonResponse);

        dump("*** TRY TO READ PERSON AFTER DELETE", readPerson);

        try {
            readPersonResponse = (ReadPersonReturn)client.process(callContext,
(DataObject)readPerson);

            dump("*** READ RESULT:", readPersonResponse);
        } catch (CompositeServiceException e) {
            out.println("*** READ RESULT: " + e.getLocalizedMessage());
        }

    }

    private void dump(String title, Object object) {
        DataObject dataObject = (DataObject)object;
        String xml = helperContext.getXMLHelper().save(
                dataObject,
                dataObject.getType().getURI(),
                dataObject.getType().getName());
        out.println(title);
        out.println(xml);
        out.println();
    }
}
```

CHAPTER 3

Representational State Transfer Business Entity Service Calls

This chapter includes the following topics:

- REST APIs for Business Entity Services Overview, 21
- Supported REST Methods, 22
- Authentication Method, 22
- Authentication Cookies for Login from Third-Party Applications, 22
- Web Application Description Language File, 23
- REST Uniform Resource Locator, 24
- Header and Body Configuration, 24
- Standard Query Parameters, 27
- Formats for Dates and Time in UTC, 27
- Configuring WebLogic to Run Business Entity Service REST Calls, 28
- Viewing Input and Output Parameters, 29
- JavaScript Template, 29
- JavaScript Example, 30
- REST API Reference for Business Entity Services , 32

REST APIs for Business Entity Services Overview

REST endpoint calls make all business entity services available as web services.

You can make REST calls to create, update, delete, and search for base object records and related child records in a business entity. You can perform operations, such as merge, unmerge, and match records. You can make REST calls to create, update, and search for tasks and perform tasks. You can also make REST calls to create, update, and delete files, such as attachments for tasks or records.

A REST business entity service call is a web service request in the form of a Uniform Resource Locator (URL). The MDM Hub assigns a unique URL for each base object in a business entity. You can use the unique URL to identify which base object to update or delete.

Note: Before you use the REST APIS to call the business entity services, validate the Operational Reference Store.

Supported REST Methods

The REST APIs for business entity services use the standard HTTP methods to perform operations on the resources, such as records, tasks, and files.

The REST APIs for business entity services support the following HTTP request methods:

Method	Description
GET	Retrieves information about a record, a task, or a file.
POST	Creates a task, a root record, a child record, or a file. **Note:** The Operational Reference Store (ORS) name in a POST request is case sensitive. If the case of the ORS name does not match the name in the MDM Hub, an error occurs.
PUT	Updates a root record, a child record, a task, or a file.
PATCH	Updates a task partially.
DELETE	Deletes a root record, a child record, or a file.

Authentication Method

The REST endpoints for business entity services use the basic HTTP authentication method to authenticate users. When you first connect to a business entity service with your browser, you must provide your MDM Hub user name and password. After successful authentication, you can use the business entity services REST APIs to perform operations.

The browser caches the user credentials and uses them with every subsequent request to a business entity service.

Authentication Cookies for Login from Third-Party Applications

Use authentication cookies to authenticate the MDM Hub users and call the business entity services from third party applications. You can obtain a cookie based on the credentials of an authenticated user. Save the cookie and use it to call the REST APIs. You need not hard-code the user name and password.

Make the following POST request to log in to the Entity 360 View with your user name and password:

```
POST http://<host>:<port>/e360/com.informatica.tools.mdm.web.auth/login
{
    user: 'admin',
    password: 'user password'
}
```

When the login operation is successful, the server returns the authentication cookie in the `set-cookie` header field. The following sample code shows a `set-cookie` in the response header:

```
Set-Cookie: auth_hash_cookie="admin===QTc1RkNGQkNCMzc1RjIyOQ==";
Version=1; Path=/
```

Store the hash and use it in the request header of your API calls. You need not provide a user name and a password for the API calls.

The following example shows how to use an authentication cookie in your API request header:

```
GET http://<IP of host>:8080/cmx/cs/localhost-orcl-DS_UI1/Person?action=meta
Cookie: auth_hash_cookie="admin===QTc1RkNGQkNCMzc1RjIyOQ=="
```

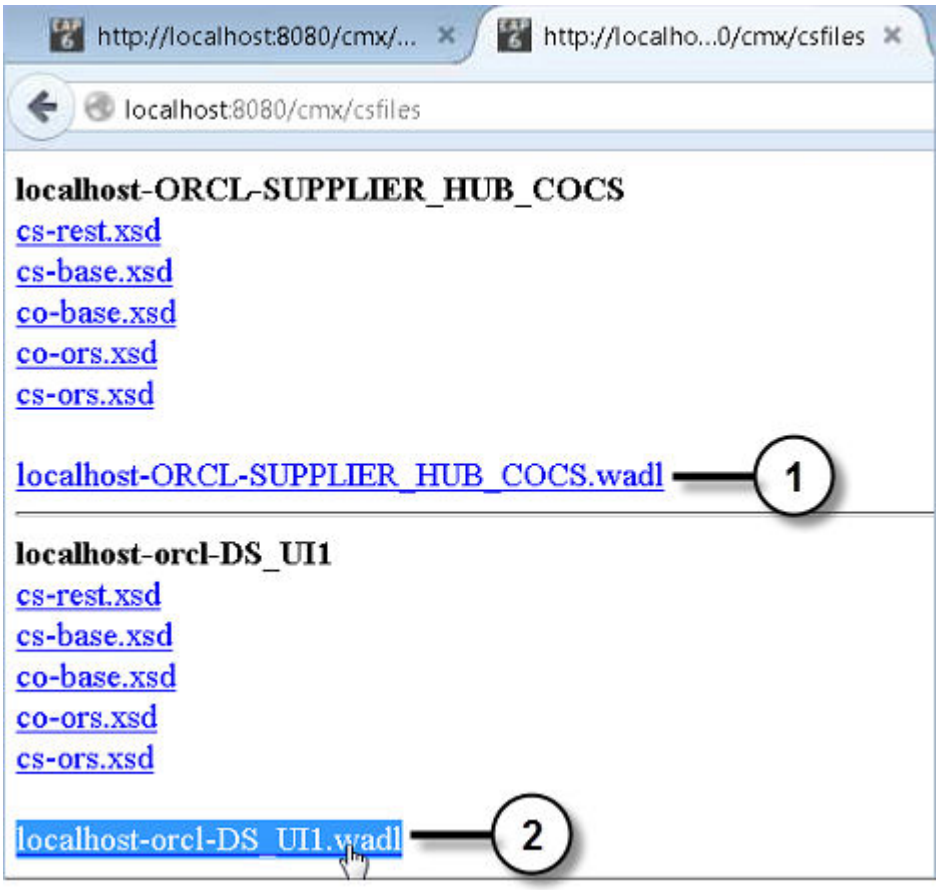
Web Application Description Language File

Web Application Description Language (WADL) files contain XML descriptions of the REST web services, all the REST URLs, and all REST parameters. The MDM Hub generates a WADL file for each Operational Reference Store.

The WADL files for each Operational Reference Store are in the following location:

```
http:://<host>:<port>/cmx/csfiles
```

The following image shows the location where you can download the WADL file for the Operational Reference Stores:



1. Link to download the WADL file for the SUPPLIER_HUB_COCS Operational Reference Store
2. Link to download the WADL file for the DS_UI1 Operational Reference Store

REST Uniform Resource Locator

You use a REST URL to make REST calls for business entity services.

The REST URL has the following syntax:

```
http://<host>:<port>/<context>/<database ID>/<path>
```

The URL has the following fields:

host

The host that is running the database.

port

Port number that the database listener uses.

context

The context for the business entity, search, query, match, and task APIs is `cmx/cs`.

The context for the match columns API is `cmx`.

The context for file APIs is `cmx/file`.

Note: In a Hosted MDM environment, include the tenant name in the context. For example, the context can be `<tenant name>/cmx/cs` or `<tenant name>/cmx/file`.

database ID

ID of the ORS as registered in the Databases tool in the Hub Console.

path

The object that you want to use the API on, such as records, tasks, or files.

If the URL is for a root record, the path is the root object name followed by a unique identifier.

An example of a path for a Person root record is `Person/798243.json`.

If the URL is for a record that is a direct child of the root object, the path also includes the child record name and a unique identifier.

An example of a path for a billing address record that is a child of a Person root record is:

`Person/798243/BillAddresses/121522.json`.

If the URL is for a child record that is at a depth of two or greater, the path also includes the depth.

The following URL is an example of a REST URL for a child record with a depth of 2:

```
http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/798243/BillAddresses/121522.json?depth=2
```

Note: Parameters are case sensitive. Ensure the case of the parameter names in the REST URL matches the case of the parameter names in the REST configuration.

Header and Body Configuration

A REST operation combines an HTTP method with the full URL to the resource. For a complete request, combine the REST operation with the appropriate HTTP headers and any required data. A REST request has a header and a body component. You can use the JSON or the XML format to define a request.

Request Header

Use a request header to define the operating parameters or the metadata of the REST operation. The header consists of a series of field-value pairs. The API request line contains the method and the URL. Specify the header fields after the request line.

To construct the REST API request header, add the header fields after the `<METHOD> <<host>:<port>/<context>/<database ID>/<Path>` request line, as shown in the following example:

```
<METHOD> <<host>:<port>/<context>/<database ID>/<Path>
Content-Type: application/<json/xml>
Accept: application/<json/xml>
```

The following table describes some of the commonly used request header fields:

Request component	Description
Content-Type	Media type of the data in the request. When you include a body in the REST request, you must specify the media type of the body in the Content-Type header field. Include the Content-Type header field in PUT and POST requests.
Accept	Media type of the data in the response. To specify the request format, use `application/<json/xml>` in the header or add `.json` or `.xml` to the URL. Default is XML.

Request Body

Use the REST API request body to send data in the request. A request body is used with a method that can have a body attached to it, such as the POST and the PUT methods. The body of data is written after the header lines. If the request message has a body, use the Content-Type header field to specify the format of the body in the request header.

The XML Schema Definition (XSD) files describe what elements and attributes you can use. The content of the request body depends on the element types that you define in the XSD files.

The XSD files are in the following location:

```
http:://<host>:<port>/cmx/csfiles
```

XML Format

When you use the XML request format, define the request object as an enclosing set of tags.

Use the following XML format to define a request object:

```
<request object>
        <attribute1>value1</attribute1>
        <attribute2>value2</attribute2>
</request object>
```

The following example shows the XML representation of a request object:

```
<task>
    <taskType>
                <name>UpdateWithApprovalWorkflow</name>
    </taskType>
    <taskId>urn:b4p2:5149</taskId>
    <owner>manager</owner>
    <title>Smoke test task 222</title>
    <comments>Smoke testing</comments>
    <dueDate>2015-06-15T00:00:00</dueDate>
    <status>OPEN</status>
```

```
    <priority>NORMAL</priority>
    <creator>admin</creator>
    <createDate>2015-06-15T00:00:00</createDate>
    <updatedBy>admin</updatedBy>
    <lastUpdateDate>2015-06-15T00:00:00</lastUpdateDate>
    <businessEntity>Person</businessEntity>
    <orsId>localhost-orcl-DS_UI1</orsId>
    <processId>IDDUpdateWithApprovalTask</processId>
    <taskRecord>
                <businessEntity>
                        <name>Person</name>
                        <key>
                                    <rowid>123</rowid>
                                    <systemName></systemName>
                                    <sourceKey></sourceKey>
                        </key>
                </businessEntity>
    </taskRecord>
</task>
```

JSON Format

When you use the JSON request format, define the request object with the type attribute.

Use the following JSON format to specify a request object:

```
{
 "type"="<request object>"
    "<attribute1>": "<value1>",
    "<attribute2>": "<value2>",
}
```

The following example shows the JSON representation of a request object:

```
{
     "type"="task"
     taskType: {name:"UpdateWithApprovalWorkflow"},
     taskId: "urn:b4p2:5149",
     owner: "manager",
     title: "Smoke test task 222",
     comments: "Smoke testing",
     dueDate: "2015-06-15T00:00:00",
     status: "OPEN",
     priority: "NORMAL",
     creator: "admin",
     createDate: "2015-06-15T00:00:00",
     updatedBy: "admin",
     lastUpdateDate: "2015-06-15T00:00:00",
     businessEntity: "Person",
     orsId: "localhost-orcl-DS_UI1",
     processId: 'IDDUpdateWithApprovalTask',
     taskRecord: [{
       businessEntity:{
         name: 'Person',
         key:{
            rowid: '123',
            systemName: '',
            sourceKey: ''
         }
       }
     }]
}
```

Standard Query Parameters

The business entity services REST APIs use standard query parameters to filter, paginate, and expand the results.

Use a question mark (?) to separate the query parameters from the other parameters. Query parameters are key-value pairs separated by the equal sign. Use an ampersand (&) to separate a sequence of query parameters.

The following REST request URL shows how to use query parameters:

```
/Person/123/Phone/SFA:456/PhoneUse?recordsToReturn=100&recordStates=ACTIVE,PENDING
```

Use the following standard query parameters:

Parameter	Description
recordsToReturn	Specifies the number of rows to return. Default is 10.
firstRecord	Specifies the first row in the result. Default is 1. Used in subsequent calls to read more pages.
searchToken	Specifies the search token returned with previous request. You can use the search token to fetch subsequent pages of search results. For example, the following query lists the first page: `/Person/123/Phone` The following query returns the second page: `/Person/123/Phone?searchToken=SVR1.AZAM5&firstRecord=10`
returnTotal	If set to true, returns the number of records in the result. Default is false.
depth	Specifies the number of child levels to include in the result.

Formats for Dates and Time in UTC

In the request and response, all dates and times are specified in UTC (Coordinated Universal Time), with or without an offset for a specific time zone.

When you specify a date and time in a request body, use one of the formats defined in Date and Time Formats (NOTE-datetime) for ISO specification 8601.

The following guidelines are taken from the NOTE-datetime document:

Type	Syntax	Example
Date: Year	YYYY	1997
Date: Year and month	YYYY-MM	1997-07
Date: Year, month, and day	YYYY-MM-DD	1997-07-16
Date plus hours and minutes	YYYY-MM-DDThh:mmTZD	1997-07-16T19:20+01:00

Type	Syntax	Example
Date plus hours, minutes, and seconds	YYYY-MM-DDThh:mm:ssTZD	1997-07-16T19:20:30+01:00
Date plus hours, minutes, seconds, and fractional seconds	YYYY-MM-DDThh:mm:ss.sTZD	1997-07-16T19:20:30.45+01:00

where:

- YYYY = a four-digit year
- MM = a two-digit month, from 01 to 12
- DD = a two-digit day of the month, from 01 to 31
- T = a literal value that follows the date and introduces the time
- hh = two digits for the hour, from 00 to 23
- mm = two digits for the minutes, from 00 to 59
- ss = two digits for the seconds, from 00 to 59
- s = one or more digits representing a decimal fraction of a second
- TZD = time zone designator (Z or +hh:mm or -hh:mm)
  - Z for UTC time
  - +hh:mm for a local time zone that is ahead of UTC
  - -hh:mm for a local time zone that is behind UTC

Configuring WebLogic to Run Business Entity Service REST Calls

Because business entity services REST calls use basic HTTP authentication, you must disable the WebLogic Server authentication for the REST calls. To configure WebLogic to run business entity service REST calls, edit the WebLogic `config.xml` file.

1. Navigate to the following WebLogic directory:

   On UNIX.

   ```
   <WebLogic installation directory>/user_projects/domains/base_domain/config
   ```

   On Windows.

   ```
   <WebLogic installation directory>\user_projects\domains\base_domain\config
   ```

2. Open the following file in a text editor:

   ```
   config.xml
   ```

3. Before the closing </security-configuration> tag, add the following XML code:

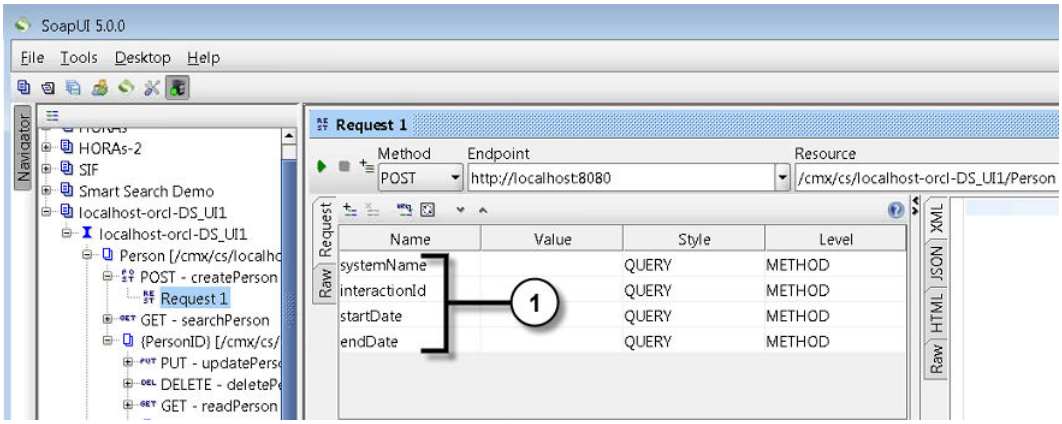
   ```
   <enforce-valid-basic-auth-credentials>
      false
   </enforce-valid-basic-auth-credentials>
   ```

Viewing Input and Output Parameters

You can use a functional testing tool such as SoapUI to view the REST API input and output parameters.

Download the WADL file and then import the file into the functional testing tool to create a REST project.

The following image shows the input parameters in the SoapUI for the createPerson REST API:



1. Input parameters for createPerson REST API

Note: The WADL file produced in WebSphere environments might not import correctly into SoapUI. If the input parameters do not appear in SoapUI, edit the WADL file to remove the `xmlns` attribute from each `param` element and then import the WADL file again.

JavaScript Template

The following code sample shows a basic template that you can modify to create JavaScript code for REST business entity service calls. You need the jQuery java script library.

```
(function ($) {
    window.CSClient = window.CSClient || {
        baseUrl: "/cmx/cs/" + "[siperian-client.orsId]",
        user: "[siperian-client.username]",
        pass: "[siperian-client.password]",

        process: function (method, url, body, params) {
            var fullUrl = this.baseUrl + url + ".json?" + $.param(params);
            return $.ajax({
                method: method,
                contentType: "application/json",
                url: fullUrl,
                data: JSON.stringify(body),
                beforeSend: function (xhr) {
                    xhr.setRequestHeader("Authorization", "Basic " + btoa(CSClient.user
+ ":" + CSClient.pass));
                }
            });
        },

        readCo: function (url, params) {
            return this.process("GET", url, null, params);
        },
        createCo: function (url, body, params) {
            return this.process("POST", url, body, params);
        },
```

```
        updateCo: function (url, body, params) {
            return this.process("PUT", url, body, params);
        },
        deleteCo: function (url, params) {
            return this.process("DELETE", url, null, params);
        }
    };
})(jQuery);
```

JavaScript Example

The resource kit has sample Java source code that shows how to make REST business entity service calls.

The sample code is in the following file:

`<MDM Hub installation directory>\hub\resourcekit\samples\COS\source\resources\webapp\rest-api.html`

The following code shows REST API calls to create a person root record, add multiple child records, delete one child record, and then delete the person record and all child records:

```
<html>
<head>
    <script type="text/javascript" src="jquery-1.11.1.js"></script>
    <script type="text/javascript" src="cs-client.js"></script>
</head>
<body>

<script type="text/javascript" language="javascript">
    $(document).ready(function () {

        $("#run").click(function () {

            log = function(msg, json) {
                $('#log').before("<hr/><b>" + msg + "</b>");
                $('#log').before("<pre>" + JSON.stringify(json, undefined, 2) + "</
pre>");
            };

            CSClient.createCo(
                    "/Person",
                    {
                        firstName: "John",
                        lastName: "Smith"
                    },
                    {
                        systemName: "Admin"
                    }
            ).then(
                    function (result) {
                        log("PERSON CREATED:", result);
                        return CSClient.readCo(
                                "/Person/" + result.Person.rowidObject.trim(),
                                {
                                    depth: 1
                                }
                        );
                    }
            ).then(
                    function (result) {
                        log("READ CREATED PERSON:", result);
                        return CSClient.updateCo(
                                "/Person/" + result.rowidObject.trim(),
                                {
                                    genderCd: {
```

```
                                        genderCode: "M"
                                    },
                                    TelephoneNumbers: {
                                        item: [
                                            {
                                                phoneNumber: "111-11-11"
                                            },
                                            {
                                                phoneNumber: "222-22-22"
                                            }
                                        ]
                                    }
                                },
                                {
                                    systemName: "Admin"
                                }
                            );
                        }
                ).then(
                        function (result) {
                            log("PERSON UPDATED:", result);
                            return CSClient.readCo(
                                    "/Person/" + result.Person.rowidObject.trim(),
                                    {
                                        depth: 3,
                                        readSystemFields: true
                                    }
                            );
                        }
                ).then(
                        function (result) {
                            log("READ UPDATED PERSON:", result);
                            return CSClient.deleteCo(
                                    "/Person/" + result.rowidObject.trim() + "/
TelephoneNumbers/" + result.TelephoneNumbers.item[0].rowidObject.trim(),
                                    {
                                        systemName: "Admin"
                                    }
                            );
                        }
                ).then(
                        function (result) {
                            log("TELEPHONE DELETED:", result);
                            return CSClient.readCo(
                                    "/Person/" + result.Person.rowidObject.trim(),
                                    {
                                        depth: 3
                                    }
                            );
                        }
                ).then(
                        function (result) {
                            log("READ PERSON AFTER TELEPHONE IS DELETED:", result);
                            return CSClient.deleteCo(
                                    "/Person/" + result.rowidObject.trim(),
                                    {
                                        systemName: "Admin"
                                    }
                            );
                        }
                ).then(
                        function (result) {
                            log("PERSON DELETED:", result);
                            return CSClient.readCo(
                                    "/Person/" + result.Person.rowidObject.trim(),
                                    {
                                        depth: 1,
                                        recordStates: "ACTIVE,PENDING,DELETED",
                                        readSystemFields: true
                                    }
                            );
```

```
                }
            ).then(
                function (result) {
                    log("READ PERSON AFTER DELETE (HSI -1):", result);
                }
            );
        });

    });
</script>

<input type="button" id="run" value="Run..."/>
<p/>
<div id="log"></div>
</body>
</html>
```

REST API Reference for Business Entity Services

The REST API reference for business entity services lists the REST APIs and provides a description for each API. The API reference also contains information on the URLs, the query parameters, the sample requests, and the sample responses.

Get Metadata

The Get Metadata REST API returns the data structure of a business entity or a business entity relationship.

The API uses the GET method to return the following metadata of a business entity:

- Structure of the business entity
- List of fields
- Field types such as the data type and size
- Security settings for operations, such as create, update, and merge
- Localized labels for nodes or fields
- Name of the code and display fields for lookup fields
- Default values for fields and lookup fields

  Note: Multiple default values are not supported with dependent lookup fields.

The API returns the following details of a business entity relationship:

- Name of the relationship and its label.
- From and To business entities.
- Direction of the relationship.

Request URL

The Get Metadata URL has the following format for a business entity:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?action=meta
```

The Get Metadata URL has the following format for a relationship:

```
http://<host>:<port>/<context>/<database ID>/<relationship>?action=meta
```

Use the query parameter "action=meta" to retrieve the metadata information. Ensure that you specify the name of the business entity correctly.

Make an HTTP GET request to the Get Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity/relationship>?
action=meta
```

You can add HTTP headers to the request.

Sample API Request

The following sample request retrieves metadata information for the Person business entity and root node:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?action=meta
```

The following sample request includes a header to retrieve the metadata information for the Person business entity in the JSON format:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?action=meta
Accept: application/json
```

The following sample request retrieves metadata information for the HouseholdContainsMemberPerson relationship:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductGroupProductGroup?action=meta
```

Sample API Response

The sample responses for an entity and a relationship contain security permissions. The first `operations` section defines the possible permissions. The `object` section lists your permissions for the entire business entity or relationship. The `fields` section defines your permissions at the field level.

The following example shows a partial data structure for the Person business entity in the JSON format.

```
{
  "operations": {
    "read": {
      "allowed": true
    },
    "search": {
      "allowed": true
    },
    "create": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:28:13.455-04:00",
          "taskType": "AVOSBeUpdate",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "update": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:28:13.455-04:00",
          "taskType": "AVOSBeUpdate",
          "comment": "This is urgent. Please review ASAP"
        },
```

```
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "merge": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:28:13.455-04:00",
          "taskType": "AVOSBeMerge",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "delete": {
      "allowed": true
    },
    "unmerge": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:28:13.455-04:00",
          "taskType": "AVOSBeUnmerge",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    }
  },
  "objectType": "ENTITY",
  "timeline": true,
  "object": {
    "operations": {
      "read": {
        "allowed": true
      },
      "create": {
        "allowed": true
      },
      "update": {
        "allowed": true
      },
      "merge": {
        "allowed": true
      },
      "delete": {
        "allowed": true
      },
      "unmerge": {
        "allowed": true
      }
    },
    "field": [
      {
        "operations": {
          "read": {
            "allowed": true
          },
          "create": {
            "allowed": true
          },
          "update": {
            "allowed": true
          }
```

```
      },
      "allowedValues": [
        "Person"
      ],
      "searchable": {
        "filterable": true,
        "facet": true
      },
      "name": "partyType",
      "label": "Party Type",
      "dataType": "String",
      "length": 255
    },
    {
      "operations": {
        "read": {
          "allowed": true
        },
        "create": {
          "allowed": true
        },
        "update": {
          "allowed": true
        }
      },
      "name": "lastName",
      "label": "Last Name",
      "dataType": "String",
      "length": 50
    },
    {
      "operations": {
        "read": {
          "allowed": true
        },
        "create": {
          "allowed": true
        },
        "update": {
          "allowed": true
        }
      },
      "searchable": {
        "filterable": true,
        "facet": true
      },
      "name": "displayName",
      "label": "Display Name",
      "dataType": "String",
      "length": 200
    },
  ...
  ],
  "name": "Person",
  "label": "Person",
  "many": false
 }
}
```

List Metadata

The List Metadata REST API returns a list of business entities or relationships that you have defined. The response includes timeline information and security information, if the business entities contain that information. You can use the API to retrieve the list of relationships that start from an entity, end at an entity, or start from and end at specified entities.

The API uses the GET method.

List Metadata URL

The List Metadata URL has the following format for entity metadata:

```
http://<host>:<port>/<context>/<database ID>/meta/entity
```

The List Metadata URL has the following format for relationship metadata:

```
http://<host>:<port>/<context>/<database ID>/meta/relationship
```

Make the following HTTP GET request to the List Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/meta/entity|relationship
```

Query Parameters

You can append the query parameters to the request URL to filter the search results for the business entity relationships. You can specify the direction and search for relationships to and from a business entity.

The following table lists the query parameters:

Parameter	Description
start	Optional. Specifies the entity from which a relationship originates. For example, the `meta/relationship?start=Organization` query returns all relationships that start from the `Organization` business entity.
finish	Optional. Specifies the entity at which a relationship ends. For example, the `meta/relationship?finish=Person` query returns all relationships that end at the `Person` business entity.

You can specify both the parameters to retrieve all the relationships between two business entities. For example, the `meta/relationship?start=Organization&finish=Person` query returns all relationships that start from the `Organization` business entity and end at the `Person` business entity.

Sample API Request

The following sample request retrieves the list of business entities configured:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/meta/entity
```

The following sample request retrieves the list of relationships configured:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/meta/relationship
```

The following sample request retrieves the list of relationships that start from the `Organization` business entity and end at the `Person` business entity:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/meta/relationship?
start=Organization&finish=Person
```

Sample API Response

The following example shows an excerpt of the list of configured relationships:

```
{
  "link": [],
  "firstRecord": 1,
  "pageSize": 10,
  "item": [
    {
      "operations": {
        "read": {
```

```
      "allowed": true
    },
    "search": {
      "allowed": false
    },
    "create": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:31:48.167-04:00",
          "taskType": "AVOSBeUpdate",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "update": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:31:48.167-04:00",
          "taskType": "AVOSBeUpdate",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "merge": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:31:48.167-04:00",
          "taskType": "AVOSBeMerge",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    },
    "delete": {
      "allowed": true
    },
    "unmerge": {
      "allowed": true,
      "task": {
        "template": {
          "title": "Review changes in {taskRecord[0].label}",
          "priority": "NORMAL",
          "dueDate": "2018-04-24T09:31:48.167-04:00",
          "taskType": "AVOSBeUnmerge",
          "comment": "This is urgent. Please review ASAP"
        },
        "comment": "AS_REQUIRED",
        "attachment": "OPTIONAL"
      }
    }
  },
  "objectType": "ENTITY",
  "timeline": false,
  "object": {
    "link": [
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
```

```
            CreditCard.json?action=meta",
      "rel": "entity"
    }
  ],
  "field": [
    {
      "name": "issuingCompany",
      "label": "Issuing Company",
      "dataType": "String",
      "length": 100
    },
    {
      "name": "expirationYear",
      "label": "Expiration Year",
      "dataType": "String",
      "length": 4
    },
    {
      "allowedValues": [
        "Credit Card"
      ],
      "name": "accountType",
      "label": "Account Type",
      "dataType": "String",
      "length": 255
    },
    {
      "name": "accountNumber",
      "label": "Account Number",
      "dataType": "String",
      "length": 20
    },
    {
      "name": "securityCode",
      "label": "Security Code",
      "dataType": "String",
      "length": 4
    },
    {
      "name": "expirationMonth",
      "label": "Expiration Month",
      "dataType": "String",
      "length": 2
    },
    {
      "name": "cardholderName",
      "label": "Card Holder Name",
      "dataType": "String",
      "length": 100
    },
    {
      "name": "consolidationInd",
      "label": "Consolidation Ind",
      "dataType": "Integer",
      "length": 38,
      "readOnly": true,
      "system": true
    },
    {
      "name": "creator",
      "label": "Creator",
      "dataType": "String",
      "length": 50,
      "readOnly": true,
      "system": true
    },
    {
      "name": "interactionId",
      "label": "Interaction Id",
      "dataType": "Integer",
      "length": 38,
```

```
    "readOnly": true,
    "system": true
  },
  {
    "name": "updatedBy",
    "label": "Updated By",
    "dataType": "String",
    "length": 50,
    "readOnly": true,
    "system": true
  },
  {
    "name": "lastUpdateDate",
    "label": "Last Update Date",
    "dataType": "Date",
    "readOnly": true,
    "system": true
  },
  {
    "name": "lastRowidSystem",
    "label": "Last Rowid System",
    "dataType": "String",
    "length": 14,
    "readOnly": true,
    "system": true
  },
  {
    "name": "dirtyIndicator",
    "label": "Dirty Indicator",
    "dataType": "Integer",
    "length": 38,
    "readOnly": true,
    "system": true
  },
  {
    "name": "deletedBy",
    "label": "Deleted By",
    "dataType": "String",
    "length": 50,
    "readOnly": true,
    "system": true
  },
  {
    "name": "deletedInd",
    "label": "Deleted Indicator",
    "dataType": "Integer",
    "length": 38,
    "readOnly": true,
    "system": true
  },
  {
    "name": "hubStateInd",
    "label": "Hub State Ind",
    "dataType": "Integer",
    "length": 38,
    "readOnly": true,
    "system": true
  },
  {
    "name": "deletedDate",
    "label": "Deleted Date",
    "dataType": "Date",
    "readOnly": true,
    "system": true
  },
  {
    "name": "rowidObject",
    "label": "Rowid Object",
    "dataType": "String",
    "length": 14,
    "readOnly": true,
```

```
            "system": true
          },
          {
            "name": "cmDirtyInd",
            "label": "Content metadata dirty Ind",
            "dataType": "Integer",
            "length": 38,
            "readOnly": true,
            "system": true
          },
          {
            "name": "createDate",
            "label": "Create Date",
            "dataType": "Date",
            "readOnly": true,
            "system": true
          }
        ],
        "name": "CreditCard",
        "label": "Credit Card",
        "many": false
      }
    },
    ...
  ]
}
```

List Match Columns

The List Match Columns REST API can return either a list of match rule sets for a specified business entity or a list of match columns for a specified match rule set. You can generate a list of match columns and use the match columns with the SearchMatch REST API.

For information on configuring match columns and match rule sets, see the *Multidomain MDM Configuration Guide*.

The API uses the GET method.

Request URL

The context for the List Match Columns URL is `cmx`. In a Hosted MDM environment, include the tenant name in the context, such as `<tenant name>/cmx`.

The List Match Columns URL has the following formats:

URL to return all the match rule sets

Use the following URL to list all the match rule sets for a specific business entity:

```
http://<host>:<port>/<context>/queryTemplate/<database ID>/<business entity>
```

Make the following HTTP GET request to the List Match Columns URL:

```
GET http://<host>:<port>/<context>/queryTemplate/<database ID>/<business entity>
```

URL to return the match columns used in a specified match rule set

Use the following URL to list all the match columns in a specified match rule set:

```
http://<host>:<port>/<context>/queryTemplate/<database ID>/<business entity>/<match
rule set>
```

Sample API Request

Request for match rule sets

The following sample request lists the match rule sets for the Person business entity:

```
GET http://localhost:8080/cmx/queryTemplate/localhost-orcl-DS_UI1/Person
```

Request for match columns

The following sample request lists the match columns that are used in the match rule set IDL2:

```
GET http://localhost:8080/cmx/queryTemplate/localhost-orcl-DS_UI1/Person/IDL2
```

Sample API Response

The following sample response contains the match columns that are included in the IDL2 match rule set:

```
{
 "queryTemplates":[
 {
  "businessEntity": "Person",
  "matchRuleSet": "IDL2",
  "type": "extended",
  "searchFields": [
    {
      "name": "displayName",
      "mandatory": true
    },
    {
      "name": "BillAddresses.Address.addressLine1",
      "mandatory": false
    },
    {
      "name": "ShipAddresses.Address.addressLine2",
      "mandatory": false
    },
    {
      "name": "ShipAddresses.Address.addressLine1",
      "mandatory": false
    }
  ]
 }
]
}
```

Read Record

The Read Record REST API returns the details of a root record in the business entity. You can use the API to return the details of the child records of a root record. You can use the API to view the content metadata of a record.

The API uses the GET method.

You can sort the result set to view the information in ascending or descending order. Use the POST method when you need more and complex parameters. For example, when you want to retrieve the data where the child elements are ordered by a set of fields.

Request URL

Use the row ID or the name of the source system and the source key to specify the record in the request URL.

The Read Record URL can have the following formats:

URL with rowId

Use the following URL format when you specify the row ID:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root
record>
```

Make the following HTTP GET request to the URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the
root record>
```

URL with source system name and source key

Use the following URL format when you specify the source system name and the source key:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<system name>:<source
key>
```

URL with system name and global business identifier (GBID) of an object

Use the following URL format when you specify the source system name and the GBID:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<system
name>:uid:<gbid>
```

URL with only the GBID

Use the following URL format when you specify only the GBID:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/:uid:<gbid>
```

URL with more than one GBID

Use the following URL format when you specify more than one GBID:

```
http://<host>:<port>/<context>/<database ID>/<business
entity>/:one:<gbid>,another:<gbid>
```

URL to return the details of the child nodes

Use the following URL format to return the details of the child nodes:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the record>?
depth=n
```

URL to return the details of a child nodes

Use the following URL format to return the details of child nodes:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the record>?
children=<comma separated list of child node names or paths>
```

For example, `children= BillAddresses/Address,Email`

URL to return the details of a particular node

Use the following URL format to return the details of particular node:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the record>/
<node name>
```

URL to return the details of the children of a particular node

Use the following URL format to return he details of the children of a particular node:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the record>/
<node name>?children=<child node name>
```

URL to return the content metadata of a record

Use the following URL format to return the content metadata of a record:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root
record>?contentMetadata=<content metadata type>
```

For example, you can retrieve matches for the child records with the following GET request:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the
root record>?contentMetadata=MATCH
```

URL to sort the child elements by fields

Use the following URL format to sort the child elements by fields:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root
record>/<node name>?order=-<field name>
```

Use the character the character - as suffix to specify descending order.

Query Parameters

You can append the query parameters to the request URL to filter the details of the record.

The following table lists the query parameters:

Parameter	Description
depth	Number of child levels to return. Specify 2 to return the root node and its direct children, and 3 to return the root node, direct children, and grandchildren. Default is 1.
effectiveDate	Date for which you want to retrieve the data.
readSystemFields	Indicates whether to return the system fields in the result. Default is false.
recordStates	State of the record. Provide a comma-separated list of states. The supported record states are ACTIVE, PENDING, and DELETED. Default is ACTIVE.
contentMetadata	Metadata of the record. Provide a comma-separated list. For example, XREF, PENDING_XREF, DELETED_XREF, HISTORY, XREF_HISTORY, and MATCH. When you select MATCH, the response contains a list of matched records that are retrieved from the _MTCH table.
historyDate	Date for which you want to retrieve history data. The response contains record data for the specified date retrieved from the _HIST table. You can use historyDate together with the contentMetadata parameter to retrieve historical metadata. Set contentMetadata to XREF, BVT, or TRUST. - XREF. The response contains historical cross-reference data from the _HXRF table. - BVT. The response contains historical best version of the truth from the _HCTL table. - TRUST. The response contains historical trust settings from the _HCTL and _HVXR tables.
children	Comma-separated list of child node names or paths.

Parameter	Description
suppressLinks	Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all parent-child links in the response. Default is false. For example, the `Person/1242?depth=10&suppressLinks=true` query will display the record details up to 10 child levels, with no parent-child links visible in the response.
order	Comma-separated list of field names with an optional prefix of + or –. The prefix + indicates to sort the results in ascending order, and the prefix – indicates to sort the results in descending order. Default is +. When you specify more than one parameter, the result set is ordered by the parameter that is first in the list, followed by the next. For example, the `Person/1242/Names?order=-name` query will display the result with the names in descending order. The `Person/1242/BillAddresses?order=rowidObject,-effStartDate` query will display the billing addresses with the rowid in ascending order and then, the effective start date in descending order.

The following example shows how to filter the details of a record:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/123/Phone/SFA:456/PhoneUse?
recordsToReturn=100&recordStates=ACTIVE,PENDING&contentMetadata=XREF
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

POST Request to Specify Sort Order of Child Elements

Use a POST request to order the result set by multiple fields. Include the parameters or fields in the POST body.

The following sample request shows how to use the POST request for a read operation and sort the data by multiple fields:

```
http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/ReadPerson.json
{
 parameters:
 {
   coFilter: {
   object: {
     name:"Person",
     key: {
       rowid: 1242
     },
     order: "lastName",
     object:[
       {name:"Names", order:"-name"},
       {name:"Phone", order:"phoneNum, -phoneCountryCd",
       object:[{name:"PhonePermissions", order:"-column1"}]
       }
    ]}
  }
 }
}
```

Note: At each level of business entity, for each type of children only one type of sort order is allowed.

Sort Order Considerations

The Read Record API supports sorting order by one or more fields for each business entity child node. The following section describes certain considerations that you need to be aware of when you specify the sort order.

- If you specify the sort order for the grandchild and not for the child, the grandchild element is sorted as per the order specified. The child element is not sorted by the sort order specified for the grandchild. The following is a sample request:

  ```
  http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1242/Phone/861/
  PhonePermissions?order=-column1
  ```

  In the sample request, while a descending sort order is specified for grandchild PhonePermissions, no order is specified for the child element Phone. Phone is not sorted by the PhonePermissions sort order.

- If you specify the sort order for the child and not for the grandchild, the child is sorted by the sort order specified. The grandchild is not sorted by the sort order specified for the child. The following is a sample request:

  ```
  {parameters:
   {coFilter: {
     object: {
       name:"Person", key: { rowid: 1242 }, order: "lastName",
       object:[
         {name:"Names", order:"-name"},
         {name:"Phone", order:"-phoneCountryCd, -phoneNum", object:
  [{name:"PhonePermissions"}]},
      ]}
  }}}
  ```

  In the sample request, the sort order is specified for child Phone and not for grandchild PhonePermissions. The child Phone is sorted by the order specified.

- If you specify the sort order for the child and the grandchild, both are sorted as per the sort order. The following sample request specifies the sort order for the Phone (child) and the PhonePermissions (grandchild):

  ```
  {parameters:
   {coFilter: {
     object: {
       name:"Person", key: { rowid: 1242 }, order: "lastName",
       object:[
         {name:"Names", order:"-name"},
         {name:"Phone", order:"-phoneCountryCd, -phoneNum", object:
  [{name:"PhonePermissions", order:"-column1"}]},
      ]}
  }}}
  ```

- A child can be sorted only by the columns in the child itself, while the grandchild can be sorted by the columns in the grandchild. In the following sample requests, the Phone is sorted by PhoneType and the PhonePermissions is sorted by column 1. PhoneType is a column in Phone (child) and column 1 is a column in PhonePermissions (grandchild).

  ```
  http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1242/Phone?order=-PhoneType
  ```

  ```
  http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1242/Phone/861/
  phonePermissions?order=column1
  ```

- At each level of business entity, for each type of children only one type of sort order is allowed. In the following request, you have specified different sort orders for the PhonePermissions children of different parents. However, because first the sort order is specified as descending, the PhonePermissions children of both parents (rowid 861 and rowid 862) are sorted in descending order.

  ```
  {parameters:
   {coFilter: {
     object: {
       name:"Person", key: { rowid: 1242 }, order: "lastName",
       object:[
  ```

```
        {name:"Names", order:"-name"},
        {name:"Phone", key: { rowid:861 }, order:"+phoneCountryCd, -phoneNum", object:
[{name:"PhonePermissions", order:"-column1"}]},
        {name:"Phone", key: {rowid:862}, order:"phoneNum, -phoneCountryCd", object:
[{name:"PhonePermissions", order:"column1"}]}
    ]}
}}}
```

Sample API Requests

The following sample request returns the details of the root record in the Person business entity.

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102
```

The following sample request returns the details of a child record with rowId 2. The child base object is genderCd and the child record is at a depth of 2.

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/genderCd/2?depth=2
```

The following sample request uses the system name and source key to returns the details of the root record:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/SFA:9000000000
```

The following sample request returns the details of the root record and its XREF records:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102?contentMetadata=XREF
```

The following sample request returns the details of the root record, with the names in descending order:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1242/Names?order=-name
```

The folllowing sample request returns the billing addresses with the rowid in ascending order and then, the effective start date in descending order:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1242/BillAddresses?
order=rowidObject,-effStartDate
```

Sample API Response

The following example shows the details of the root record in the Person business entity.

```
{
    "link": [
        {
            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102",
            "rel": "self"
        },
        {
            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102?
depth=2",
            "rel": "children"
        }
    ],
    "rowidObject": "102           ",
    "label": "DARWENT,JIMMY",
    "partyType": "Person",
    "statusCd": "A ",
    "lastName": "DARWENT",
    "middleName": "N",
    "firstName": "JIMMY",
    "displayName": "JIMMY N DARWENT",
    "genderCd": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102",
                "rel": "parent"
            },
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
```

```
genderCd",
                    "rel": "self"
                },
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
genderCd?depth=2",
                    "rel": "children"
                }
            ],
            "genderCode": "M"
        },
        "generationSuffixCd": {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102",
                    "rel": "parent"
                },
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
generationSuffixCd?depth=2",
                    "rel": "children"
                },
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
generationSuffixCd",
                    "rel": "self"
                }
            ],
            "generationSuffixCode": "I"
        }
}
```


The following example shows the details of a child record in the genderCd base object.


```
{
    "link": [
        {
            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102",
            "rel": "parent"
        },
        {
            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
genderCd/2?depth=2",
            "rel": "children"
        },
        {
            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/102/
genderCd/2",
            "rel": "self"
        }
    ],
    "rowidObject": "2             ",
    "label": "LU Gender",
    "genderDisp": "MALE",
    "genderCode": "M"
}
```


Create Record

The Create Record REST API creates a record in the specified business entity. Send the record data in the request body. Use the Promote API to promote and add the record in the business entity.

The API uses the POST method to create a record.

Request URL

The Create Record URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?systemName=<name of the source system>
```

Note: The name of the source system is a required parameter in the URL.

Make the following HTTP POST request to the Create Record URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>?systemName=<name of the source system>
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>?systemName=<name of the source system>
Content-Type: application/<json/xml>
```

URL Parameters

The name of the source system is a required parameter in the request URL.

The following table lists the parameters that you can use in the URL:

Parameter	Description
systemName	Name of the source system.
interactionId	ID of the interaction. You can group multiple requests into a single interaction. All changes are done with the interaction ID.
startDate and endDate	Specifies the period of time for which the record is effective. Provide these parameters for a timeline-enabled base object.
validateOnly	Indicates whether the write business entity service validates incoming data. Default is false.
recordState	State of the record. Use the parameter to specify the initial state of the record. Use ACTIVE or PENDING. Default is ACTIVE.
taskComment	Add a comment to the workflow task triggered by the API.
taskAttachments	If task attachments are enabled, attach a file to the workflow task triggered by the API.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Request Body

Send data for the record in the REST request body. Use the JSON format or the XML format to send data. Use the Get Metadata API to get the structure of the business entity and provide the required parameter values in the request body.

Response Header and Body

When the response is successful, the API returns the interactionId and the processId in the response header and the record details in the response body.

If the process generates an interaction ID and uses it to create the record, the API returns the interaction ID. If the process starts a workflow instead of directly saving the record to the database, the API returns the process ID which is the ID of the workflow process.

The following example shows a response header with an interaction ID and a process ID:

```
BES-interactionId: 72200000242000
BES-processId: 15948
```

The response body contains the record with the generated row IDs.

Sample API Request

The following sample request creates a record in the Person business entity. The request adds a comment and attachment to the workflow task triggered by the API.

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
systemName=Admin&taskComment=Read my
comment&taskAttachments=TEMP_SVR1.29OT8,TEMP_SVR1.29OT9
Content-Type: application/json

{
    firstName: "John",
    lastName: "Smith",
    Phone: {
        item: [
            {
                phoneNumber: "111-11-11"
            }
        ]
    }
}
```

Sample API Response

The following sample response shows the response header and body after successfully creating a record:

```
BES-interactionId: 72200000242000
BES-processId: 15948
Content-Type: application/json
{
    "Person": {
        "key": {
            "rowid": "2198246",
            "sourceKey": "72200000241000"
        },
        "rowidObject": "2198246",
        "Phone": {
            "item": [
                {
                    "key": {
                        "rowid": "260961",
                        "sourceKey": "72200000243000"
                    },
                    "rowidObject": "260961"
                }
            ]
        }
    }
}
```

Update Record

The Update Record REST API updates the specified root record and its child records. Send the ID of the record in the request URL. Send the summary of the changes in the body of the request.

After the change, if the record is in a pending state, use the Promote API to promote the changes. For example, if the update triggered a review workflow, the record is in a pending state until the review is complete.

The API uses the POST method.

For root records, you can choose to use the simplified PUT version, where the body of the request contains the changed field, not a change summary. To update root records and related child or grandchild records, use the POST version. You also have the option to use the PUT version with the PKEY or the specify the rowidObject of the child record.

Request URL

The Update Record URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?systemName=<name
of the source system>
```

Note: The name of the source system is a required parameter in the URL.

Make the following HTTP POST request to the Update Record URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
systemName=<name of the source system>
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
PUT http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
systemName=<name of the source system>
Content-Type: application/<json/xml>
```

Query Parameters

The name of the source system is a required query parameter.

You can use the following query parameters in the request:

Parameter	Description
systemName	Name of the source system.
interactionId	ID of the interaction. You can group multiple requests into a single interaction. All changes are done with the interaction ID.
startdate and enddate	Specifies the period of time for which the record is effective. Provide these parameters for a timeline-enabled base object.
validateOnly	Indicates if the write business entity service validates incoming data. Default is false.
recordState	Sets the state of the record. Use ACTIVE, PENDING, or DELETED. For example, if you set`recordState=ACTIVE`, and request runs on a soft-deleted record, the request restores the record to the active state.

Parameter	Description
taskComment	Add a comment to the workflow task triggered by the API.
taskAttachments	If task attachments are enabled, attach a file to the workflow task triggered by the API.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Request Body

Send the data to update in the REST request body. Use the JSON format or the XML format to send the data.

Provide the new parameter values. Use the $original parameter to indicate the old value of a parameter you want to update.

You can also use the following properties with child records:

Properties / Elements	Type	Description
MATCH	object	If you want to add or remove match candidates from the match table for the child record, add the MATCH object to the child record.
MERGE	object	If you want to merge child records or remove candidates from the merge, add the MERGE object to the child record.

The following JSON code sample changes the first name in the root record to Bob:

```
{
  rowidObject: "123",
  firstName: "Bob",
  lastName: "Smith",
    $original: {
      firstName: "John"
                }
}
```

The following JSON code sample removes a match candidate for an Address child record, and defines the merge for two Telephone child records:

```
{
  rowidObject: "123",
  firstName: "Bob",
  lastName: "Smith",
    $original: {
      firstName: "John"
                }
  Address: {     // remove A3 from the matches for A2 in the Address_MTCH table
    item: [
      {
        rowidObject: "A2",
        MATCH: {
          item: [   // to remove matched child records for A2, specify null
            null
          ],
          $original: {
            item: [{key: {rowid: 'A3'}}]
          }
        }
      }
```

```
      ]
    }
    Telephone: {   // override the matches for the telephone child records
      item:[
        {
          rowid: "T1",
          MERGE: {
            item: [   // to remove merge candidates for T1, specify null
              null,
              null
            ],
            $original: {
              item: [
                {rowid: "T2"},
                {rowid: "T3"}
              ]
            }
          }
        },
        {
          rowid: "T4",
          MERGE: {
            item: [   // to add or override matches, specify matched records
              {rowid: "T2"}
            ],
          $original: {
            item: [
              null
            ]
          }
        }
      ]
    }
}
```

Response Header

When the response is successful, the API returns the interactionId and the processId in the response header and the record details in the response body.

If the process generates an interaction ID and uses it to create the record, the API returns the interaction ID. If the process starts a workflow instead of directly saving the record to the database, the API returns the process ID which is the ID of the workflow process

The following example shows a response header with an interaction ID and a process ID:

```
BES-interactionId: 72200000242000
BES-processId: 15948
```

The response body contains the record with the generated rowIds.

Sample API Request

The following sample request updates a root record and its child record in a business entity. Person is the business entity and Phone is a child base object. The request adds a comment and attachment to the workflow task triggered by the API.

```
POST  http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?systemName=Admin

{
    rowidObject: "233",
    firstName: "BOB",
    lastName: "LLOYD",
    Phone: {
        item: [
            {
                rowidObject: "164",
```

```
                phoneNumber: "777-77-77",
                $original: {
                    phoneNumber: "(336)366-4936"
                }
            }
        ]
    },
    $original: {
        firstName: "DUNN"
    }
}
```

Sample API Response

The following sample response shows the response header and body after successfully updating a record:

```
BES-interactionId: 72300000001000
BES-processId: 16302
{
    Person: {
        key: {
            rowid: "233",
            sourceKey: "SYS:233"
        },
        rowidObject: "233",
        preferredPhone: {
            key: {}
        }
    }
}
```

Delete Record

The Delete Record REST API deletes a root record in a business entity. Use the API to delete the child records of a root record.

The API uses the DELETE method to delete a record.

Request URL

The Delete record URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowID of the root
record>?systemName=Admin
```

Note: The name of the source system as a required parameter in the URL.

Make the following HTTP DELETE request to the Delete Record URL:

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowID of the
record>?systemName=Admin
```

Use the following URL format to delete a child record of a root record:

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowID of the
record>/<child base object>/<rowID of the child record>?systemName=Admin
```

Query Parameter

The name of the source system is a required URL parameter. Use the systemName parameter to specify the source system.

Sample API Request

The following sample request deletes a root record in the Person business entity:

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/292258?systemName=Admin
```

Sample API Response

The following sample response shows the response after successfully deleting a root record in the Person business entity:

```
{
    "Person": {
        "key": {
            "rowid": "292258",
            "sourceKey": "WRK50000_7016"
        },
        "rowidObject": "292258"
    }
}
```

List Record

The List Record REST API returns the list of lookup values or foreign key values. Lookups provide reference data and give a list of possible values for a given column.

The API uses the GET method.

Use the API to also get the lookup code values and lookup code descriptions. You can specify the sort order for the lookups. Use the POST method when you need more and complex parameters.

Request URL

The List Record REST URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<lookup table>?action=list
```

Make the following HTTP GET request to the URL:

```
GET http://<host>:<port>/<context>/<database ID>/<lookup table>?action=list
```

Use the following URL format to list the codes of the lookup values:

```
http://<host>:<port>/<context>/<database ID>/<lookup table>?action=list&idlabel=<lookup
code>%3A<lookup display name>
```

Note: Use the Get Metadata API to get the exact URL to list the lookup values.

Query Parameters

You can append the query parameters to the request URL to filter the results.

The following table lists the query parameters:

Parameter	Description
suppressLinks	Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all parent-child links in the response. Default is false. For example, the `Person/1242?depth=10&suppressLinks=true` query will display the record details up to 10 child levels, with no parent-child links visible in the response.
order	Used to list the lookup values in ascending or descending order. Use the character + as prefix to specify ascending order and the character – as prefix to specify descending order. By default, when you do not specify the prefix, the result set is ordered in ascending order. For example, the `LUNamePrefix?action=list&order=-namePrefixDisp` query lists the prefixes for name, which are sorted by the display names of the prefixes in descending order.

POST Request to Specify Sort Order

Use a POST request to specify the sort order for the lookup values. Include the parameters or fields in the POST body.

The following example shows how to use the POST request for a list operation:

```
http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/ListCO.json
{
 parameters:
 {
   coFilter: {
   object: {
     name:"LUCountry",
     order: "-countryNameDisp"
   }
  }
 }
}
```

Sample API Request

The following sample request list the lookup values:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/LUGender?action=list
```

The following sample request lists the codes for the gender lookup values:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/LUGender?
action=list&idlabel=genderCode%3AgenderDisp
```

The following sample request lists the prefixes for names, with the display names of the prefixes in descending order:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/LUNamePrefix?action=list&order=-
namePrefixDisp
```

Sample API Response

The following sample response shows the list of lookup values:

```
{
    "firstRecord": 1,
    "pageSize": 2147483647,
    "searchToken": "SVR1.AU5LC",
    "item": [
        {
            "rowidObject": "1             ",
            "genderDisp": "UNKNOWN",
            "genderCode": "N"
        },
        {
            "rowidObject": "2             ",
            "genderDisp": "MALE",
            "genderCode": "M"
        },
        {
            "rowidObject": "3             ",
            "genderDisp": "FEMALE",
            "genderCode": "F"
        }
    ]
}
```

The following sample response shows the code for the lookup values:

```
{
    "item": [
        {
            "id": "F",
            "label": "FEMALE"
        },
        {
            "id": "M",
            "label": "MALE"
        },
        {
            "id": "N",
            "label": "UNKNOWN"
        }
    ],
    "firstRecord": 1,
    "recordCount": 0,
    "pageSize": 2147483647,
    "searchToken": "SVR1.AU5LD"
}
```

Search Record

The Search Record REST API searches for indexed values in a searchable root record and in all the child records. You can use filters and facets to view a subset of the search results. Facets group the search results into categories, while filters narrow down the search results. The API searches all fields that are configured as searchable and returns the records that match the search criteria.

The API uses the GET method to search the indexes of the searchable fields.

Request URL

The Search Record URL has the following formats:

URL for a simple search

Use the following URL for a simple search:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?q=<string value>
```

Make the following HTTP GET request to the Search Record URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>?q=<string value>
```

URL for a fielded search

Use the following URL for a fielded search:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?fq=<business entity
field name>='<business entity field value>'
```

If you specify a negative numerical value for a business entity field, such as `-120`, the ranking of the records that are returned might be affected.

URL for a facets search

Use the following URL for a simple search with facets:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?q=<string
value>&facets=<field name>
```

Use the following URL for a fielded search with facets:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?fq=<business entity
field name>='<business entity field value>'&facets=<field name>
```

URL for a filter search

Use the following URL for a simple search with filters:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?q=<string
value>&filters=<field name1>=<field value1> AND <field name2>=<field value2> ...
```

Use the q or the fq parameter in the search.

URL encoding

Use URL encoding because the URL includes characters, such as spaces and single quotation marks.

The following example shows the URL encoded representation of the Search Record URL:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?q=<field name>%3D
%27<value of the field>
```

Query Parameters

Use the q or the fq query parameters to provide the string value for the search. The q and the fq query parameters are mutually exclusive. Use the fq parameter for a fielded search. Use the AND logical operator for multiple conditions.

The following table lists the parameters that you can use in the URL:

Parameter	Description

q	Specifies the string value or the search term. The query searches for occurrences of the search term anywhere in a record. Used in a simple search.
	For example, the `Person?q=STEVE` query searches for records with the term STEVE.
	To search for two or more terms together, include the terms in double quotation marks. Use the character + before each term if you want the search results to contain the term. If the field value contains a space, enclose the field value in single quotes.
	Use the following query to search for an exact match to WILLIAM JOHN LAWSON:
	`Person?q="WILLIAM JOHN LAWSON"`
	Use the following query to search for WILLIAM, JOHN or LAWSON:
	`Person?q=WILLIAM JOHN LAWSON`
	Use the following query to search for WILLIAM, JOHN, and LAWSON:
	`Person?q=WILLIAM JOHN LAWSON&queryOperator=AND`

fq	Specifies the string value or search term in a particular field. The query searches for the term only in that part of a record. Used in a targeted search based on indexed fields. For example, the `Person?fq=displayname=STEVE` query searches for records with the display name STEVE.
facets	Specifies the fields that should be treated as facets or categories by which the search results are grouped. Specify only searchable fields. Used with the q and the fq parameters. Syntax is `&facets=FieldName1,FieldName2,FieldNameN` For example, the `Person?q=STEVE&facets=department` query searches for persons with the display name STEVE and groups the search results by the departments. Th search displays the records of the persons with the display name STEVE and these records are grouped by the departments.
filters	Specifies the fields by which you can narrow down the search results. Specify only filterable fields. Used with the q and the fq parameters. For example, the `Person?fq=STEVE&filters=birthdate='1980-11-27T08:00:00Z'` query searches for persons with the display name STEVE and filters the search results by the birth date. The search displays the records of the persons who have the display name STEVE and whose date of birth is 27 November, 1980. **Note:** Specify a date within single quotation marks.
depth	Specifies the number of child levels to return. Specify 2 to return the root node and its direct children, and 3 to return the root node, direct children, and grandchildren. Specify 1 to return the root node alone. By default, no depth is specified. If no depth is specified, then the search results return the root node and children where a match for the search term is found. For example, the `Person?q=STEVE&depth=2` query searches for records with the term STEVE and returns information about the root record and its direct children.

Parameter	Description
queryOperator	Specifies whether the search finds any of the strings in the search term or all of the strings in the search term. The parameter takes one of the following values: - `OR`. Searches for any of the strings listed in the `f` or `fq` parameter. - `AND`. Searches for all of the strings listed in the `f` or `fq` parameter. If you do not specify this parameter, the default is `OR`. For example, the `Person?q=WILLIAM JOHN LAWSON&queryOperator=AND` query searches for records that contains WILLIAM, JOHN, and LAWSON.
suppressLinks	Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all parent-child links in the response. Default is false. For example, the `Person?q=STEVE&suppressLinks=true` query searches for records with the term STEVE and returns the response where no parent-child links are visible.
readSystemFields	Indicates whether to return the system fields in the result. Default is false.
order	Comma-separated list of field names with an optional prefix of + or –. The prefix + indicates to sort the results in ascending order, and the prefix – indicates to sort the results in descending order. Default is +. If you want to use a child field to sort the results, use the full name of the field. For example, `BillAddresses.Address.cityName`. When you specify more than one parameter, the result set is ordered by the parameter that is first in the list, followed by the next. For example, the `Person?order=displayName,-BillAddresses.Address.cityName` query sorts the results by display name in ascending order and then by city name in descending order.
maxRecordsToSort	Maximum number of search results that you want to sort. Default is 1000.

Specify a range with the `filters` parameter:
You can use the filters parameter to narrow down the search results within a specified range. You can specify the range for filterable fields of the numeric and the date data types.

Use the following format for the integer data type:

```
fieldName1=[fromValue,toValue]
```

The range is from the fromValue to the toValue. Ensure that the fromValue is lower than the toValue. For example, the `filters=age=[35,45]` query narrows the search results and searches for records in the age group of 35 to 45.

Use the following format for the date data type:

```
fieldName1=[fromDate,toDate]
```

The range is from the fromDate to the toDate. For example, the `filters=birthdate=[2000-06-12T12:30:00Z,2015-06-12T12:30:00Z]` query specifies the date of birth between 12 June 2000 and 12 June 2015.

Note: When you specify a exact match date filter, enclose it within single quotation marks. When you specify a date range, do not use quotation marks.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

Request with the q parameter

The following sample request searches the Person business entity for records with the name STEVE.

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?q=STEVE
```

Request with the fq parameter

The following sample request searches the Person business entity for records with the display name STEVE. The displayName field is an indexed field.

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?fq=displayName=STEVE
```

Request with the sort option

The following sample request searches the Person business entity for records with the display name STEVE and sorts the results by city in ascending order:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
fq=displayName=STEVE&order=BillAddresses.Address.cityName
```

Request with the fq parameter and the AND logical operator

The following sample request searches the Person business entity for records with the display name STEVE and the tax ID DM106:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?fq=displayName=STEVE AND
taxId=DM106
```

Request with a facet

The following sample request searches the Person business entity for records with the display name STEVE and narrows down the results by grouping them into departments:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
fq=displayName=STEVE&facets=department
```

Request with a filter (exact filter)

The following sample request searches the Person business entity for records with the display name STEVE and filters by the specified city and country:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
fq=displayName=STEVE&filters=cityName=Canberra AND country=Australia
```

Request with a filter range

The following sample request searches the Person business entity for records with display name STEVE and filters by the age group 35 to 45:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
fq=displayName=STEVE&filters=age=[35,45] AND cityName=Canberra
```

Sample API Response

The following sample response shows the result of a search by the name STEVE:

```
{
    "firstRecord": 1,
    "recordCount": 2,
    "pageSize": 10,
    "item": [
        {
```

```
            "Person": {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
Person/1443",
                        "rel": "self"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
Person/1443?depth=2",
                        "rel": "children"
                    }
                ],
                "rowidObject": "1443          ",
                "label": "CRAIG,STEVE",
                "partyType": "Person",
                "lastName": "CRAIG",
                "firstName": "STEVE",
                "taxID": "stevecraig                    ",
                "displayName": "STEVE CRAIG"
            }
        },
        {
            "Person": {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
Person/285",
                        "rel": "self"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
Person/285?depth=2",
                        "rel": "children"
                    }
                ],
                "rowidObject": "285           ",
                "label": "PEARSON,STEVE",
                "partyType": "Person",
                "lastName": "PEARSON",
                "firstName": "STEVE",
                "displayName": "STEVE PEARSON"
            }
        }
    ]
}
```


Suggester

The Suggester REST API returns a list of related terms for a search string, based on the data present in your database. Use the API to accept the characters that you type in a user interface field and return suggestions to autocomplete what you type. You can find and select the string from the list of suggestions. Use the Suggester API for searchable fields.

The API uses the GET method.

Note: To use the API to provide a list of autocomplete suggestions for a searchable field, set the `suggester` property of the field to true and reindex the data.

Request URL

The Suggester REST URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?suggest=<string>
```

Make the following HTTP GET request to the URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>?suggest=<string>
```

Note: If you use the Solr search engine, the suggester index must rebuild each time the Process Server restarts. To ensure that the index rebuilds each time a Process Server restarts, set the buildIndex parameter to `true`:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?
suggest=<string>&buildIndex=true
```

Query Parameters

The following table lists the query parameters:

Parameter	Description
suggest	Required. Specifies the string that you want suggestions for.
recordsToReturn	Specifies the number of rows to return.
buildIndex	Optional. Indicates whether to build the index. When you restart the system, set this parameter to true to explicitly build the index for the collection. This parameter might be deprecated in a future release.

Sample API Request

The following sample request returns a list of suggestions that you can use in a user interface:

```
GET http://localhost:8080/cmx/cs/localhost-infa-DS_UI1/Person.json?suggest=Abhinav
```

Sample API Response

The following sample response shows the list of suggestions:

```
{
term: [2]
"abhinav goel"
"abhinav gupta"

}
```

SearchQuery

The SearchQuery REST API searches for records that are an exact match of the field values specified in a query. You can use the SearchQuery API to retrieve all the records for a specific business entity or to retrieve records based on specific field values. Unlike the Search Records API that searches for the specified values across all the searchable fields, the SearchQuery API searches for the specified values within specific fields.

You can filter the query results to display specific values in a root business entity record and in the child records. You can use the AND, IN, and RANGE operators in the query.

To include specific fields in the query results, specify the fields or a business entity view that includes the fields. You can sort the query results to view the records in ascending or descending order.

The API uses the GET method.

Request URL

The context for the SearchQuery URL is `cmx/cs`. In a Hosted MDM environment, include the tenant name in the context, such as `<tenant name>/cmx/cs`.

The SearchQuery URL has the following formats:

URL to return all the records of a specific business entity type

Use the following URL to search for all the records of the specified business entity type:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?action=query
```

Make the following HTTP GET request to the SearchQuery URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>?action=query
```

URL to return all the details of records that match the specified field values

Use the following URL to search for records that match the field values that you specify:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?
action=query&filter=<business entity field name 1>='<business entity field value 1>'
AND <business entity field name 2>='<business entity field value 2>'...AND <business
entity field name n>='<business entity field value n>'
```

URL to return specific details of records that match the specified field values

Use the following URL to search for records and display specific record fields in the search results:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?
action=query&filter=<business entity field name 1>='<business entity field value 1>'
AND <business entity field name 2>='<business entity field value 2>'...AND <business
entity field name n>='<business entity field value n>'&outputView=<business entity
view>
```

Query Parameters

Define the query as a list of field-value pairs.

The following table describes the query parameters that you can use in the URL:

Parameter	Description
action	Required. Returns all the records of the specified business entity type in the query results. Set to `query`, and use the parameter with the `filter` parameter. When used without the `filter` parameter, the query searches for all the records of the specified business entity type. For example, use the following query to search for all the Person business entity records: `Person?action=query`
filter	Required. Specifies a list of field-value pairs separated by operators. Valid operators are AND, IN, and Range. For example, use the following query to search for the Person records with the first name STEVE and the last name SMITH: `Person?action=query&filter=firstName='STEVE' AND lastName='SMITH'`

Parameter	Description
depth	Specifies the number of child record levels to return. For example, you can specify the following levels: - 1. Return the root record. - 2. Returns the root record and its direct child records. - 3. Returns the root record, the direct child records, and grandchild records. For example, use the following query to search for records with the first name STEVE and return information about the root record and its direct child records: `Person?action=query&filter=firstName='STEVE' AND lastName='SMITH'&depth=2`
suppressLinks	Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all the parent-child links in the response. Default is false. For example, use the following query to search for records with the first name STEVE and return a response where no parent-child links are visible: `Person?action=query&filter=firstName='STEVE'&suppressLinks=true`
readSystemFields	Indicates whether to return the system fields in the result. Default is false.
fields	Specifies the fields to display in the query results.
outputView	Specifies the business entity view that you want to use to display the query results. When you configure the business entity view for the query results, include the fields that you want to display in the query results.
Order	Specifies the sort order of the query results. Use the plus (+) character as prefix to specify ascending order and the minus (-) character as prefix to specify descending order. By default, the query result is in ascending order. When you specify more than one parameter, the result set is ordered by the parameter that is first in the list, followed by the next parameter.

You can use the following operators within the filter parameter:

AND

Searches for records with all the field values listed in the filter parameter.

For example, use the following query to search for records with the first name STEVE and the last name SMITH:

```
Person?action=query&filter=firstName='STEVE' AND lastName='SMITH'
```

IN

Searches for records with any of the values listed in the filter parameter.

For example, use the following query to search for records with the first name STEVE or JOHN:

```
Person?action=query&filter=firstName IN [STEVE,JOHN]
```

Range

Searches for records within a specified range. You can specify a range for the numeric and the date data type fields.

Use the following format for the integer data type:

```
<business entity field name>=[fromValue,toValue]
```

The range is from the fromValue to the toValue. Ensure that the fromValue is lower than the toValue.

For example, use the following query to search for records in the age group 35 to 45:

```
Person?action=query&filter=firstName IN [STEVE,JOHN] AND age=[35,45]
```

Use the following format for the date data type:

```
<business entity field name>=[fromDate,toDate]
```

The range is from the fromDate to the toDate.

For example, use the following query to search for records with the date of birth between 12 June 2000 and 12 June 2015:

```
Person?action=query&filter=birthDate=[2000-06-12T12:30:00Z,2015-06-12T12:30:00Z]
```

Sample API Request

The following sample request queries the Person business entity for records with the first name STEVE and last name SMITH:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
action=query&filter=firstName='STEVE' AND lastName='SMITH'&outputView=PersonAddressView
```

Sample API Response

The following sample response shows the results of a query for the Person records with the first name STEVE and last name SMITH:


```
{
  "link": [],
  "firstRecord": 1,
  "pageSize": 10,
  "searchToken": "SVR1.1T8UU",
  "facet": [],
  "item": [
    {
      "Person": {
        "rowidObject": "268           ",
        "label": "Person: SMITH, STEVE,268           ",
        "partyType": "Person",
        "lastName": "SMITH",
        "displayName": "STEVE SMITH",
        "firstName": "STEVE"
      }
    },
    {
      "Person": {
        "rowidObject": "448           ",
        "label": "Person: SMITH, STEVE,448           ",
        "partyType": "Person",
        "lastName": "SMITH",
        "displayName": "STEVE SMITH",
        "firstName": "STEVE"
      }
    }
  ]
}
```

Exporting the SearchQuery Results to a CSV File

To export the results of a SearchQuery request to a CSV file, in the request URL path, specify the name of the business entity as a `.CSV` file. You can use all the query parameters in the request URL.

For example, use the following request URL to export the results of a search for records that match the field values that you specify:

```
http://<host>:<port>/<context>/<database ID>/<business entity>.CSV?
action=query&filter=<business entity field name 1>='<business entity field value 1>' AND
<business entity field name 2>='<business entity field value 2>'...AND <business entity
field name n>='<business entity field value n>'
```

Sample API Request

The following sample request queries for records with the first name STEVE and last name SMITH and returns the query results in the CSV format:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person.CSV?
action=query&filter=firstName='STEVE' AND
lastName='SMITH'&fields=recordID,firstName,lastName
```

Sample API Response

The following sample shows the query results in the CSV format for a query with the first name STEVE and last name SMITH:

```
recordID,firstName,lastName
00023,Steve,Smith
00048,Steve,Smith
```

SearchMatch

The SearchMatch REST API searches for records that are fuzzy matches. The API identifies the matching records based on specific business entity fields that are configured as match columns in the MDM Hub. Before you use the SearchMatch API, use the List Match Columns API to identify the match columns for a business entity.

Optionally, you can specify a match rule set instead of match columns. To use a match rule set for the query, ensure that the Enable Search by Rules option is enabled for the match rule set. For information on configuring match columns and match rule sets, see the *Multidomain MDM Configuration Guide*.

You can use the AND, IN, and RANGE operators in the query.

To include specific fields in the query results, specify the fields or a business entity view that includes the fields. You can specify the sort order to view the records in the query results in ascending or descending order.

The API uses the GET method to query the business entity fields and returns the records that are fuzzy matches along with their match scores and associated match rules.

Request URL

The context for the SearchMatch URL is `cmx/cs`. In a Hosted MDM environment, include the tenant name in the context, such as `<tenant name>/cmx/cs`.

The SearchMatch URL has the following formats:

URL to return matching records based on the values of specific fields that are configured as match columns

Use the following URL to search for records that match the field values that you specify:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?
action=match&fuzzyFilter=<business entity field name 1>='<business entity field
```

```
value 1>',<business entity field name 2>='<business entity field value
2>',...<business entity field name n>='<business entity field value n>'
```

Make the following HTTP GET request to the SearchMatch URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>?
action=match&fuzzyFilter=<business entity field name 1>='<business entity field
value 1>',<business entity field name 2>='<business entity field value
2>',...<business entity field name n>='<business entity field value n>'
```

URL to return matching records based on a match rule set

Use the following URL to search for matching records based on a match rule set that you specify:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?
action=match&fuzzyFilter=<business entity field name 1>='<business entity field
value 1>',<business entity field name 2>='<business entity field value
2>',...<business entity field name n>='<business entity field value
n>'&matchRuleSet=<match rule set name>
```

Query Parameters

Use the `fuzzyFilter` parameter to specify the field values that you want to query. Use the `fuzzyFilter` parameter with the `action` parameter.

The following table describes the query parameters that you can use in the URL:

Parameter	Description
action	Required. Returns the matching records for the specified business entity. Set to `match`, and use the parameter with the `fuzzyFilter` parameter. For example, use the following query to search for a person with the first name STEVE: `Person?action=match&fuzzyFilter=STEVE`
fuzzyFilter	Required. Specifies a comma-separated list of field name and field value pairs that you want to use to query for records of a specific business entity type. For example, use the following query to search for records with the first name STEVE, who have a Toronto address: `Person?action=match&fuzzyFilter=firstName=STEVE,Address.Address.City=TORONTO`
matchRuleSet	Specifies a match rule set based on which you want to identify the matching records. If you do not have a specific match rule set, specify NONE. The automatic and the manual merge match rules are used.

filter	Specifies the field values to use to filter the results of a fuzzy search.
	For example, use the following query to search for records with the first name STEVE, who live in Toronto:
	`Person?`
	`action=match&fuzzyFilter=firstName='STEVE',lastName="SMITH'&filter=city=Toronto`

depth	Specifies the number of child record levels to return. For example, you can specify the following levels: - 1. Return the root record. - 2. Returns the root record and its direct child records. - 3. Returns the root record, the direct child records, and grandchild records. For example, use the following query to search for records with the first name STEVE and return information about the root record and its direct child records: `Person?action=match&fuzzyFilter=firstName='STEVE'&filter=city=Toronto`

Parameter	Description
suppressLinks	Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all the parent-child links in the response. Default is false. For example, use the following query to search for records with the first name STEVE and return a response where no parent-child links are visible: `Person?action=match&fuzzyFilter=firstName='STEVE'&suppressLinks=true`
readSystemFields	Indicates whether to return the system fields in the result. Default is false.
fields	Specifies the fields to display in the query results.
outputView	Specifies the business entity view that you want to use to display the query results. When you configure the business entity view for the query results, include the fields that you want to display in the query results.

You can use the following operators within the filter parameter:

AND

Searches for records with all the field values listed in the filter parameter.

For example, use the following query to search for records with the first name STEVE and the last name SMITH:

```
Person?
action=match&fuzzyFilter=firstName='STEVE',lastName='SMITH'&filter=city=Toronto AND
gender=Male
```

IN

Searches for records with any of the values listed in the filter parameter.

For example, use the following query to search for records with the first name STEVE or last name JOHN, who live in the city Toronto or Ottawa:

```
Person?action=match&fuzzyFilter=firstName='STEVE',lastName='SMITH'&filter=city in
[Toronto,Ottawa]
```

Range

Searches for records within a specified range. You can specify a range for the numeric and the date data type fields.

Use the following format for the integer data type:

```
<business entity field name>=[fromValue,toValue]
```

The range is from the fromValue to the toValue. Ensure that the fromValue is lower than the toValue.

For example, use the following query to search for records in the age group 35 to 45:

```
Person?action=match&fuzzyFilter=firstName='STEVE',lastName='SMITH'&filter=age=[35,45]
```

Use the following format for the date data type:

```
<business entity field name>=[fromDate,toDate]
```

The range is from the fromDate to the toDate.

For example, use the following query to search for records with the date of birth between 12 June 2000 and 12 June 2015:

```
Person?
action=match&fuzzyFilter=firstName='STEVE',lastName='SMITH'&filter=birthDate=[2000-06
-12T12:30:00Z,2015-06-12T12:30:00Z]
```

Sample API Request

The following sample request queries the Person business entity for records with the first name STEVE by using the match rule set IDL2:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
action=match&fuzzyFilter=firstName=STEVE&matchRuleSet=IDL2
```

Sample API Response

The following sample response shows the results of a query for records with the first name STEVE based on the match rule set IDL2:


```
{
  "link": [],
  "firstRecord": 1,
  "recordCount": 3,
  "pageSize": 10,
  "searchToken": "SVR1.17LJ2",
  "matchedEntity": [
    {
      "businessEntity": {
        "Person": {
          "rowidObject": "145           ",
          "label": "SAMUEL,STEVE",
          "partyType": "Person",
          "lastName": "SAMUEL",
          "displayName": "MR STEVE SAMUEL                                        ",
          "statusCd": "A ",
          "firstName": "STEVE",
          "genderCd": {
            "genderCode": "M"
          },
          "namePrefixCd": {
            "namePrefixCode": "MR"
          }
        }
      },
      "matchRule": "IDL2|1",
     "matchScore": "93",
      "link": [
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI2/meta/matchRule/
Person/IDL2|1.json",
          "rel": "matchRule"
        }
      ]
    },

    {
      "businessEntity": {
        "Person": {
          "rowidObject": "268           ",
          "label": "SMITH,STEVE",
          "partyType": "Person",
          "lastName": "SMITH",
          "displayName": "STEVE SMITH",
          "firstName": "SAM"
        }
      },
      "matchRule": "IDL2|1",
```

```
      "matchScore": "98",
      "link": [
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI2/meta/matchRule/
Person/IDL2|1.json",
          "rel": "matchRule"
        }
      ]
    },
    {
      "businessEntity": {
        "Person": {
          "rowidObject": "448           ",
          "label": "SMITH,STEVEN",
          "partyType": "Person",
          "lastName": "SMITH",
          "displayName": "SAM STEVEN",
          "firstName": "STEVEN"
        }
      },
      "matchRule": "IDL2|1",
      "matchScore": "98",
      "link": [
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI2/meta/matchRule/
Person/IDL2|1.json",
          "rel": "matchRule"
        }
      ]
    }

 ],
  "facet": []
}
```

Exporting the SearchMatch Results to a CSV File

To export the results of a SearchMatch request as a CSV file, in the request URL path, specify the name of the business entity as a `.CSV` file. You can use all the query parameters in the request URL.

For example, use the following request URL to export the results of a search for matching records based on a match rule set:

```
http://<host>:<port>/<context>/<database ID>/<business entity>.CSV?
action=match&fuzzyFilter=<business entity field name 1>='<business entity field value
1>',<business entity field name 2>='<business entity field value 2>',...<business entity
field name n>='<business entity field value n>'&matchRuleSet=<match rule set name>
```

Sample API Request

The following sample request searches for records that match the first name STEVE and last name SMITH and returns the query results in the CSV format:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person.CSV?
action=match&fuzzyFilter=firstName=STEVE,lastName=SMITH&fields=recordID,firstName,lastNam
e
```

Sample API Response

The following sample shows the query results for records that match the first name STEVE and last name SMITH in the CSV format:

```
recordID,firstName,lastName
00023,Steve,Smith
00035,Steven,Smith
00048,Steve,Smith
00079,Steve,Smithson
```

Get BPM Metadata

The Get BPM Metadata REST API returns the task types and two indicators that specify whether the BPM workflow tool can perform the Get Task Lineage service and the administration services.

The API uses the GET method.

Request URL

The Get BPM Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/BPMMetadata
```

Make the following HTTP GET request to the Get BPM Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/BPMMetadata
```

Sample API Request

The following sample request returns information about the task types and the BPM workflow tool:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/BPMMetadata
```

Sample API Response

The following sample response shows the task types and the value of the two indicators for the BPM workflow tool:

```
{
    "parameters": {
        "doesSupportAdministration": true,
        "doesSupportLineage": true,
        "doesSupportAttachments": true,
        "maximumAttachmentFileSizeInMb": 20,
        "taskTypes": {
            "taskTypes": [
                {
                    "name": "Merge",
                    "label": "Merge"
                },
                {
                    "name": "FinalReview",
                    "label": "FinalReview"
                },
                {
                    "name": "Update",
                    "label": "Update"
                },
                {
                    "name": "Notification",
                    "label": "Notification"
                },
                {
                    "name": "ReviewNoApprove",
                    "label": "ReviewNoApprove"
                },
                {
                    "name": "Unmerge",
                    "label": "Unmerge"
                }
            ]
        }
    }
}
```

List Tasks

The List Tasks API returns a list of workflow tasks. A workflow defines the activities in a business process and the paths of execution through the activities. Each activity is called a task.

The API uses the GET method to return a sorted and paged list of tasks.

Request URL

The List Tasks URL has the following format.

```
http://<host>:<port>/<context>/<database ID>/task
```

Make the following HTTP GET request to the List Tasks URL:

```
GET http://<host>:<port>/<context>/<database ID>/task
```

You can add HTTP headers to the request.

Query Parameters

Use the task data fields as query parameters to filter the list of tasks.

You can use the following query parameters:

Parameter	Description
taskType	A set of actions that you can perform on a record. Use the name attribute to specify the task type. For more information about task types, see the *Multidomain MDM Data Director Implementation Guide*.
taskId	ID of the task.
processId	ID of the workflow process that contains the task.
owner	User who performs the task.
title	Short description for the task.
status	Status of the task in the workflow. Use one of the following two values: - Open: Task has not started or is in progress. - Closed: Task is completed or is cancelled.
priority	Level of importance of the task. Use one of the following values: high, normal, and low.
creator	User who creates the task.
createDateBefore and createDateAfter	Date range. You can filter the tasks by the createDate field.
dueDateBefore and dueDateAfter	Date range. You can filter the tasks by the dueDate field.

Use the query parameters as name-value pairs in the request URL.

The following example shows how to use the query parameters to filter tasks:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task?
recordsToReturn=100&owner=sergey&status=OPEN
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sort Parameters

To determine the sort order in the REST API response, use the generic sort parameter and provide a comma-separated list of task fields. You can specify the sort order for each field. Use the dash sign (-) to specify descending order. The default sort order is ascending.

The following example shows how to sort the results:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task?
recordsToReturn=100&owner=sergey&status=OPEN&sort=-priority
```

Sample API Request

The following sample request retrieves the list of tasks:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task
```

The request does not contain a body.

Sample API Response

The following sample response shows the list of tasks in the JSON format:

```
{
    "firstRecord": 1,
    "recordCount": 10,
    "pageSize": 10,
    "task": [
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:15443",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "taskAction": [
                    {
                        "name": "Escalate",
                        "label": "Escalate",
                        "nextTaskType": "AVOSBeFinalReview",
                        "comment": "AS_REQUIRED",
                        "attachment": "NEVER",
                        "manualReassign": false,
                        "closeTaskView": true,
                        "cancelTask": false
                    },
                    {
                        "name": "Reject",
                        "label": "Reject",
                        "nextTaskType": "AVOSBeUpdate",
                        "comment": "MANDATORY",
                        "attachment": "MANDATORY",
                        "manualReassign": false,
                        "closeTaskView": true,
                        "cancelTask": false
                    },
                    {
                        "name": "Disclaim",
```

```
                            "label": "Disclaim",
                            "nextTaskType": "AVOSBeReviewNoApprove",
                            "comment": "AS_REQUIRED",
                            "attachment": "NEVER",
                            "manualReassign": false,
                            "closeTaskView": true,
                            "cancelTask": false
                        }
                    ],
                    "pendingBVT": true,
                    "updateType": "PENDING"
                },
                "taskId": "urn:b4p2:15443",
                "title": "Review changes in SMITH,SMITH",
                "dueDate": "2015-07-15T21:45:59-07:00",
                "status": "OPEN",
                "priority": "NORMAL",
                "businessEntity": "Person"
            },
            {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:15440",
                        "rel": "self"
                    }
                ],
                "taskType": {
                    "name": "ReviewNoApprove",
                    "label": "Review No approve",
                    "taskKind": "REVIEW",
                    "pendingBVT": true,
                    "updateType": "PENDING"
                },
                "taskId": "urn:b4p2:15440",
                "title": "Review changes in SMITH,JOHN",
                "dueDate": "2015-07-15T21:37:50-07:00",
                "status": "OPEN",
                "priority": "NORMAL",
                "businessEntity": "Person"
            },
            {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:15437",
                        "rel": "self"
                    }
                ],
                "taskType": {
                    "name": "ReviewNoApprove",
                    "label": "Review No approve",
                    "taskKind": "REVIEW",
                    "taskAction": [
                        {
                            "name": Reject",
                            "label": "Reject",
                            "nextTaskType:AVOSBeUpdate",
                            "comment": "AS_REQUIRED",
                            "attachment": "MANDATORY",
                            "manualReassign": false,
                            "closeTaskView": true,
                            "cancelTask": false
                        }
                    ],
                    "pendingBVT": true,
                    "updateType": "PENDING"
                },
                "taskId": "urn:b4p2:15437",
                "title": "Review changes in SMITH,JOHN",
                "dueDate": "2015-07-15T21:34:32-07:00",
```

```
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14820",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14820",
            "title": "Review changes in STAS,STAS",
            "dueDate": "2015-07-14T10:40:51-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14809",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14809",
            "title": "Review changes in ,93C8ORSCOFSA687",
            "dueDate": "2015-07-14T08:28:15-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14609",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14609",
            "title": "Review changes in A8,A8",
            "dueDate": "2015-07-13T08:40:11-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
```

```
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14425",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14425",
            "title": "Review changes in A7,A7",
            "dueDate": "2015-07-10T14:11:02-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14422",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14422",
            "title": "Review changes in A6,A6",
            "dueDate": "2015-07-10T13:54:09-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14415",
                    "rel": "self"
                }
            ],
            "taskType": {
                "name": "ReviewNoApprove",
                "label": "Review No approve",
                "taskKind": "REVIEW",
                "pendingBVT": true,
                "updateType": "PENDING"
            },
            "taskId": "urn:b4p2:14415",
            "title": "Review changes in A5,A5",
            "dueDate": "2015-07-10T13:51:12-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        },
        {
            "link": [
                {
                    "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/
urn:b4p2:14355",
                    "rel": "self"
```

```
                }
            ],
            "taskType": {
                "name": "Notification",
                "label": "Notification",
                "taskKind": "REVIEW",
                "pendingBVT": false,
                "updateType": "ACTIVE"
            },
            "taskId": "urn:b4p2:14355",
            "title": "Review changes in A4,A4",
            "dueDate": "2015-07-10T10:31:57-07:00",
            "status": "OPEN",
            "priority": "NORMAL",
            "businessEntity": "Person"
        }
    ]
}
```

Read Task

The Read Task REST API returns the details of a task, such as task type, priority, and status.

The API uses the GET method.

Request URL

The Read Task URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/task/<taskId>
```

Note: Use the List Tasks API to get the ID of the task.

Make the following HTTP GET request to the Read Task URL:

```
GET http://<host>:<port>/<context>/<database ID>/task/<taskId>
```

Sample API Request

The following sample request returns the details of a task:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/urn:b4p2:16605
```

Sample API Response

The following sample response shows the details of the task:

```
{
    "taskType": {
        "name": "ReviewNoApprove",
        "label": "Review No Approve",
        "taskKind": "REVIEW",
        "taskAction": [
            {
                "name": "Escalate",
                "label": "Escalate",
                "nextTaskType": "AVOSBeFinalReview",
                "comment": "AS_REQUIRED",
                "attachment": "NEVER",
                "manualReassign": false,
                "closeTaskView": true,
                "cancelTask": false
            },
            {
                "name": "Reject",
                "label": "Reject",
```

```
                "nextTaskType": "AVOSBeUpdate",
                "comment": "MANDATORY",
                "attachment": "MANDATORY",
                "manualReassign": false,
                "closeTaskView": true,
                "cancelTask": false
            },
            {
                "name": "Disclaim",
                "label": "Disclaim",
                "nextTaskType": "AVOSBeReviewNoApprove",
                "comment": "AS_REQUIRED",
                "attachment": "NEVER",
                "manualReassign": false,
                "closeTaskView": true,
                "cancelTask": false
            }
        ],
        "pendingBVT": true,
        "updateType": "PENDING"
    },
    "taskId": "urn:b4p2:16605",
    "processId": "16603",
    "title": "Review changes in HERNANDEZ,ALEJANDRO",
    "dueDate": "2015-07-23T01:18:39.125-07:00",
    "status": "OPEN",
    "priority": "NORMAL",
    "taskRecord": [
        {
            "businessEntity": {
                "key": {
                    "rowid": "114",
                    "sourceKey": "SYS:114"
                },
                "name": "Person"
            }
        },
        {
            "businessEntity": {
                "key": {
                    "rowid": "114           ",
                    "sourceKey": "SYS:114",
                    "rowidXref": "4680363"
                },
                "name": "Person.XREF"
            }
        }
    ],
    "creator": "avos",
    "createDate": "2015-07-16T01:18:46.148-07:00",
    "attachments": [
        {
            "id": urn.b4p2:22203::file1.txt",
            "name": "file1.txt",
            "contentType": "text/plain",
            "creator": "admin",
            "createDate": "2018-02-26T14:31:05.590-05:00"
        }
    ],
    "businessEntity": "Person",
    "interactionId": "72340000003000",
    "orsId": "localhost-orcl-DS_UI1"
}
```

Create Task

The Create Task REST API creates a task and starts a workflow.

The API uses the POST method to create a task and returns the ID of the workflow process that contains the task.

Request URL

The Create Task URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/task
```

Make the following HTTP POST request to the Create Task URL:

```
POST http://<host>:<port>/<context>/<database ID>/task
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
POST http://<host>:<port>/<context>/<database ID>/task
Content-Type: application/<json/xml>
```

Request Body

Specify the task attributes when you create the task. Use the JSON format or the XML format to send the task data in the request.

The following table describes the task parameters in the request body:

Parameter	Description
taskType	A set of actions that you can perform on a record. Use the name attribute to specify the task type. For more information about task types, see the *Multidomain MDM Data Director Implementation Guide*.
owner	User to whom the creator assigns the task.
title	Short description for the task.
comments	Comments for the task.
attachments	Attachments for the task.
dueDate	Date when the owner must complete the task.
status	Status of the task in the workflow. Use one of the following two values: - Open: Task has not started or is in progress. - Closed: Task is completed or is cancelled.
priority	Level of importance of the task. Use one of the following values: high, normal, and low. Default is normal.
creator	User who creates the task.
createDate	Date when you create the task.
orsId	ID of the Operational Reference Store (ORS) as registered in the Databases tool in the Hub Console.

Parameter	Description
processId	ActiveVOS® task type ID. For more information, see the *Multidomain MDM Data Director Implementation Guide*.
taskRecord	The business object root record or the cross-reference record associated with the task. Use the row ID or the source system and source key to specify the record.
businessEntity	Name of the business entity to which the taskRecord belongs.
interactionId	ID of the interaction. Use the interaction ID to keep a task context relationship between a task and records.
groups	Assign a task to all users in the specified user groups. You define user groups in the MDM Hub Console. Specify the groups as an array.

The following sample code uses the rowId to specify the taskRecord:

```
taskRecord: [{
       businessEntity:{
         name: "Person",
         key:{
            rowid: "233",
         }
       }
     }]
```

The request body has the following format:

```
{
     taskType: {name:"name of the task"},
     owner: "user who performs the task",
     title: "title of the task",
     comments: "description of the task",
     attachments: [
          {
               id: "TEMP_SVR1.1VDVS"
          }
     ],
     dueDate: "date to complete the task",
     status: "status of the task",
     priority: "priority of the task",
     creator: "use who creates the task",
     createDate: "date on which the task is created",
     updatedBy: "user who last updated the task",
     lastUpdateDate: "date on which the task was last updated",
     businessEntity: "name of the business entity",
     interactionID: "ID of an interaction",
     groups: ["group name A", "group name B", ...],
     orsId: "database ID",
     processId: "ActiveVOS task type ID",
     taskRecord: [{
       businessEntity:{
         name: "name of the business entity",
         key:{
            rowid: "rowId of the record", //Use the rowId or the source system and
source key to identify the record.
         }
       }
     }]
}
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample request creates a task for a root record:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task

{
     taskType: {name:"UpdateWithApprovalWorkflow"},
     taskId: "",
     owner: "manager",
     title: "Smoke test task",
     comments: "Smoke testing",
     dueDate: "2015-06-15T00:00:00",
     status: "OPEN",
     priority: "NORMAL",
     creator: "admin",
     createDate: "2015-06-15T00:00:00",
     updatedBy: "admin",
     lastUpdateDate: "2015-06-15T00:00:00",
     businessEntity: "Person",
     orsId: "localhost-orcl-DS_UI1",
     processId: "IDDUpdateWithApprovalTask",
     taskRecord: [{
       businessEntity:{
         name: "Person",
         key:{
            rowid: "123"
         }
       }
     }]
}
```

Sample API Response

The following example shows the response on successfully creating a task. The API returns the ID of the workflow process that contains the task.

```
{
    "parameters": {
        "processId": "15827"
    }
}
```

Update Task

The Update Task REST API updates a single task.

The API uses the PATCH method to update some task fields and the PUT method to update the complete task. Provide the ID of the task as the URL parameter.

Request URL

The Update Task URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/task/<taskId>
```

Note: Use the List Tasks API to get the ID of the task.

Make the following HTTP PUT request to the Update Task URL to update the complete task:

```
PUT http://<host>:<port>/<context>/<database ID>/task/<taskId>
```

Make the following HTTP PATCH request to the Update Task URL to update some task fields:

```
PATCH http://<host>:<port>/<context>/<database ID>/task/<taskId>
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
PUT http://<host>:<port>/<context>/<database ID>/task/<taskId>
Content-Type: application/<json/xml>
```

Request Body

Use the Read Task API to get the details of the task. Specify the task attributes when you update the task. Use the JSON format or the XML format to send the data to update in the request.

The following table describes the task parameters in the request body:

Parameter	Description
taskType	A set of actions that you can perform on a record. Use the name attribute to specify the task type. For more information about task types, see the *Multidomain MDM Data Director Implementation Guide*.
taskId	ID of the task.
owner	User who performs the task.
title	Short description for the task.
comments	Comments for the task.
attachments	Attachments for the task.
dueDate	Date when the owner must complete the task.
status	Status of the task in the workflow. Use one of the following two values: - Open: Task has not started or is in progress. - Closed: Task is completed or is cancelled.
priority	Level of importance of the task. Use one of the following values: high, normal, and low. Default is normal.
creator	User who creates the task.
createDate	Date when the task was created.
updatedBy	User who updates the task.
lastUpdateDate	Date when the task was last updated.
orsId	ID of the ORS as registered in the Databases tool in the Hub Console.
processId	ID of the workflow process that contains the task.
taskRecord	The root record or the cross-reference record associated with the task. Use the row ID or the source system and source key to specify the record.
businessEntity name	Name of the business entity to which the taskRecord belongs.

The following sample code uses the rowId to specify the taskRecord:

```
taskRecord: [{
       businessEntity:{
         name: 'Person',
         key:{
            rowid: '233',
            systemName: '',
            sourceKey: ''
         }
       }
     }]
```

For a PATCH request, the request body contains those task fields that you want to change. You can change the task title, the priority, the due date, and the owner.

For a PUT request, the request body contains all task fields. Use the following request body for a PUT request:

```
{
     taskType: {name:"name of the task"},
     taskId: "ID of the task",
     owner: "user who performs the task",
     title: "title of the task",
     comments: "description of the task",
     attachments: [
          {
               id: "TEMP_SVR1.1VDVS"
          }
     ],
     dueDate: "date to complete the task",
     status: "status of the task",
     priority: "priority of the task",
     creator: "use who creates the task",
     createDate: "date on which the task is created",
     updatedBy: "user who last updated the task",
     lastUpdateDate: "date on which the task was last updated",
     businessEntity: "name of the business entity",
     orsId: "database ID",
     processId: 'ActiveVOS task type ID',
     taskRecord: [{
       businessEntity:{
         name: 'name of the business entity',
         key:{
            rowid: 'rowId of the record',//Use the rowId or the source system and source
key to identify the record.
            systemName: '',
            sourceKey: ''
         }
       }
     }]
}
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample PUT request updates a complete task:

```
PUT http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/urn:b4p2:15934
```

```
{
     taskType: {name:"UpdateWithApprovalWorkflow"},
     taskId: "urn:b4p2:15934",
     owner: "John",
     title: "Smoke test task - updated",
```

```
        comments: "Smoke testing - updated",
        attachments: [
             {
                  id: "TEMP_SVR1.1VDVS"
             }
        ],
        dueDate: "2015-08-15T00:00:00",
        status: "OPEN",
        priority: "HIGH",
        creator: "admin",
        createDate: "2015-06-15T00:00:00",
        updatedBy: "admin",
        lastUpdateDate: "2015-06-15T00:00:00",
        businessEntity: "Person",
        orsId: "localhost-orcl-DS_UI1",
        processId: '3719',
        taskRecord: [{
          businessEntity:{
            name: 'Person',
            key:{
               rowid: '123',
               systemName: '',
               sourceKey: ''
            }
          }
        }]
}
```

The following sample PATCH request updates some task fields:

```
PATCH http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/urn:b4p2:15934

{
     processId: "3719",
     priority: "HIGH",
     owner: "John"
}
```

Sample API Response

The API returns a 200 OK response code on successfully updating a task. The response body is empty.

Task Complete

The Task Complete REST API closes a task workflow after you complete all the tasks in the workflow. Use the API to close a workflow after you process all the task related records. For example, when you select merge candidates, you can create a task that initiates the merge workflow. The merge task is complete after you preview each candidate and either merge it or mark it as not a match. Use the API to close the merge workflow.

The API uses the PUT method.

Request URL

The Task Complete URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=complete
```

Make the following HTTP PUT request to the Task Complete URL:

```
PUT http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=complete
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
PUT http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=complete
Content-Type: application/<json/xml>
```

Request Body

Send the task details in the request body. Use the Read Task API to get the details of the task.

The following table describes the task parameters in the request body:

Parameter	Description
taskType	A set of actions that you can perform on a record. Use the name attribute to specify the task type. For more information about task types, see the *Multidomain MDM Data Director Implementation Guide*.
taskId	ID of the task.
owner	User who performs the task.
title	Short description for the task.
comments	Comments for the task.
dueDate	Date when the owner must complete the task.
status	Status of the task in the workflow. Use one of the following two values: - Open: Task has not started or is in progress. - Closed: Task is completed or is cancelled.
priority	Level of importance of the task.
creator	User who creates the task.
createDate	Date when the task was created.
updatedBy	User who updates the task.
lastUpdateDate	Date when the task was last updated.
orsId	ID of the ORS as registered in the Databases tool in the Hub Console.
processId	ID of the workflow process that contains the task.
taskRecord	The root record or the cross-reference record associated with the task. Use the row ID or the source system and source key to specify the record.
businessEntity name	Name of the business entity to which the taskRecord belongs.

The following sample code uses the rowId to specify the taskRecord:

```
taskRecord: [{
       businessEntity:{
         name: 'Person',
         key:{
            rowid: '233',
            systemName: '',
            sourceKey: ''
         }
       }
     }]
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample request completes the merge workflow:

```
PUT http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/urn:b4p2:20210?
action=complete

{
     "taskType": {"name":"Merge"},
     "taskId": "urn:b4p2:20210",
     "owner": "admin",
     "dueDate": "2015-08-14T17:00:00-07:00",
     "status": "OPEN",
     "priority": "NORMAL",
     "creator": "admin",
     "createDate": "2015-06-15T00:00:00",
     "updatedBy": "admin",
     "lastUpdateDate": "2015-06-15T00:00:00",
     "businessEntity": "Person",
     "orsId": "localhost-orcl-DS_UI1",
     "processId": "20208",
     "taskRecord": [{
       "businessEntity":{
         "name": "Person",
         "key":{
            "rowid": "233",
            "systemName": "",
            "sourceKey": ""
         }
       }
     }]
}
```

Sample API Response

The API returns a 200 OK response on successfully completing a task workflow. The response body is empty.

Execute Task Action

The Execute Task Action REST API sets a task back to the workflow for further processing. Each task type has a set of task actions and a workflow that specifies the sequence of tasks. When you execute a task action, the task moves to the next step in the workflow. If a task action does not have a subsequent task, the workflow terminates when you execute the task action.

The API uses the POST method to perform actions, such as approve, escalate, or cancel a task.

Request URL

The following URL specifies the format of the Execute Task Action URL:

```
http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=<taskAction>
```

Note: Use the List Tasks API to get the ID of the task.

Make the following HTTP POST request to the Execute Task Action URL:

```
POST http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=<taskAction>
```

If you want to edit the task before executing the task action, add the Content-Type header to specify the media type of the request data:

```
POST http://<host>:<port>/<context>/<database ID>/task/<taskId>?action=<taskAction>
Content-Type: application/<json/xml>
```

Request Body

Provide the task data in the request body if you want to change the task details before you execute the task action.

The following table describes the parameters in the request body:

Parameter	Description
taskType	A set of actions that you can perform on a record. Use the name attribute to specify the task type. For more information about task types, see the *Multidomain MDM Data Director Implementation Guide*.
taskId	ID of the task.
owner	User who performs the task.
title	Short description for the task.
comments	Comments for the task.
attachments	Attachments for the task.
dueDate	Date when the owner must complete the task.
status	Status of the task in the workflow. Use one of the following two values: - Open: Task has not started or is in progress. - Closed: Task is completed or is cancelled.
priority	Level of importance of the task. Use one of the following values: high, normal, and low.
creator	User who creates the task.
createDate	Date when the task was created.
updatedBy	User who updates the task.
lastUpdateDate	Date when the task was last updated.
businessEntity	Name of the business entity.
orsId	ID of the ORS as registered in the Databases tool in the Hub Console.
processId	ID of the workflow process that contains the task.
taskRecord	The root record or the cross-reference record associated with the task. Use the row ID or the source system and source key to specify the record.
businessEntity name	Name of the business entity to which the taskRecord belongs.

The following sample code uses the rowId to specify the taskRecord:

```
taskRecord: [{
       businessEntity:{
         name: 'Person',
         key:{
            rowid: '233',
            systemName: '',
            sourceKey: ''
         }
       }
     }]
```

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample request cancels a task and ends the workflow:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/task/urn:b4p2:15934?
taskAction=Cancel
```

```
{
     taskType: {
      name:"UpdateWithApprovalWorkflow",
      taskAction: [{name: "Cancel"}]
     },
     taskId: "urn:b4p2:15934",
     owner: "manager",
     title: "Smoke test task 222",
     comments: "Smoke testing",
     attachments: [
          {
               id: "TEMP_SVR1.1VDVS"
          }
     ],
     dueDate: "2015-06-15T00:00:00",
     status: "OPEN",
     priority: "NORMAL",
     creator: "admin",
     createDate: "2015-06-15T00:00:00",
     updatedBy: "admin",
     lastUpdateDate: "2015-06-15T00:00:00",
     businessEntity: "Person",
     orsId: "localhost-orcl-DS_UI1",
     processId: '3685',
     taskRecord: [{
       businessEntity:{
         name: 'Person',
         key:{
            rowid: '123',
            systemName: '',
            sourceKey: ''
         }
       }
     }]
}
```

Sample API Response

The API returns a 200 OK response code on successfully executing a task action. The response body is empty.

List Assignable Users

The List Assignable Users REST API returns a list of users to whom you can assign the tasks that belong to a task type. Use the API to get the target users for a task.

The API uses the GET method.

Request URL

The List Assignable Users URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/user?taskType=<task
type>&businessEntity=<business entity name>
```

Make the following HTTP GET request to the List Assignable Users URL:

```
GET http://<host>:<port>/<context>/<database ID>/user?taskType=<task
type>&businessEntity=<business entity name>
```

Query Parameters

The following table lists the required parameters in the URL:

Parameter	Description
taskType	A set of actions that you can perform on a record. The task types include update with approval, update with optional approval, merge, unmerge, review no approval, final review, and update rejected record.
businessEntity	Name of the business entity.

Sample API Request

The following sample request retrieves a list of assignable users:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/user.json?
taskType=ReviewNoApprove&businessEntity=Person
```

Sample API Response

The following sample response shows the list of assignable users for the task type ReviewNoApprove:

```
{
    "users": {
        "user": [
            {
                "userName": "admin"
            }
        ]
    },
    "roles": {}
}
```

List File Metadata

The List File Metadata REST API returns a list of file metadata in a storage.

Use the List File Metadata REST API with a BPM or TEMP storage.

The API uses the GET method.

Request URL

The List File Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>
```

Make the following HTTP GET request to the List File Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/<storage>
```

Sample API Request

The following sample request retrieves a list of file metadata in a TEMP storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP
```

Sample API Response

The following sample response shows a list of file metadata:

```
{
    files: [
        {
            “fileId”: “TEMP_SVR1.1VDVS”,
            “fileName”: “file1.txt”,
            “fileType”: “text”,
            “fileContentType”: “text/plain”,
        },
        {
            “fileId”: “TEMP_SVR1.2ESDS”,
            “fileName”: “image1.png”,
            “fileType”: “image”,
            “fileContentType”: “image/png”,
        },
        ...
    ]
}
```

Create File Metadata

The Create File Metadata REST API creates metadata for a file and returns a file ID for the file.

You can use the file ID to upload, attach, update, download, and delete the file.

Use the Create File Metadata REST API with a DB or TEMP storage.

The API uses the POST method.

Request URL

The Create File Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>
```

Make the following HTTP POST request to the Create File Metadata URL:

```
POST http://<host>:<port>/<context>/<database ID>/<storage>
```

Request Body

Specify the metadata for the file.

The following table describes the parameters for the file metadata in the request body:

Parameter	Description
fileName	Name of the file. For example, `file.txt`.
fileType	Category of the file type. For example, `text` or `image`.
fileContentType	Content type of the file. The content type consists of a type and a subtype that are separated by a `/`. For example, `image/png`.

Note: The parameters required for the Create File Metadata REST API request are storage-specific.

Sample API Request

The following sample request creates metadata for a file in a TEMP storage:

```
POST http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP

{
    "fileName": "file1.txt",
    "fileType": "text",
    "fileContentType": "text/plain"
}
```

Sample API Response

The following example shows the response on successfully creating metadata for a file in a TEMP storage. The API returns the file ID for the file.

```
TEMP_SVR1.1VDVS
```

Note: The format of the file ID is <storage type>_<uniqueID>.

Get File Metadata

The Get File Metadata REST API returns the metadata for a file associated with a file ID.

Use the Get File Metadata REST API with a BPM, BUNDLE, DB, or TEMP storage.

The API uses the GET method.

Request URL

The Get File Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Make the following HTTP GET request to the Get File Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Sample API Request

The following sample request returns the metadata for a file with the file ID, `TEMP_SVR1.1VDVS`, in the TEMP storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS
```

The following sample request returns the metadata for the resource bundle file with the file ID, `besMetadata`, in the BUNDLE storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata
```

Sample API Response

The following sample response shows the metadata for a file with the file ID, `TEMP_SVR1.1VDVS`, in a TEMP storage:

```
{
    "fileName”: "file1.txt",
    "fileType”: "text",
    "fileContentType”: "text/plain"
}
```

The following sample response shows the metadata for the resource bundle file, `besMetadata`, in a BUNDLE storage:

```
{
    "fileName": "besMetadata.zip",
    "fileType": "BES Metadata Bundle",
    "fileContentType": "application/zip",
    "digest": "a08c5d97da7e6a780ed7c427ff14a8d2d396438cd65b654ad67424e226f64a41"
}
```

Update File Metadata

The Update File Metadata REST API updates the metadata for a file associated with a file ID.

Use the Update File Metadata REST API with a DB or TEMP storage.

The API uses the PUT method.

Request URL

The Update File Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Make the following HTTP PUT request to the Update File Metadata URL:

```
PUT
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Sample API Request

The following sample request updates the metadata for a file with the file ID, `TEMP_SVR1.1VDVS`, in a TEMP storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS

{
    “fileName”: “file2.txt”,
    “fileType”: “text”,
    “fileContentType”: “text/plain”
}
```

The following sample request updates the metadata for a file with the file ID, `DB_SVR1.OJU1`, in a DB storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU1
{
    "fileName": "Document_2.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

Sample API Response

The API returns a 200 OK response code on successfully updating the file metadata. The response body is empty.

Upload File Content

The Upload File Content REST API uploads the content for a file associated to a file ID.

Use the Upload File Content REST API with a BUNDLE, DB, or TEMP storage.

The API uses the PUT method.

Request URL

The Upload File Content URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>/content
```

Make the following HTTP PUT request to the Upload File Content URL:

```
PUT http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>/content
```

Sample API Request

The following sample request uploads the content for a file with the file ID, `TEMP_SVR1.1VDVS`, in a TEMP storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS/content

Test attachment content: file 1
```

The following sample request uploads the content for a file with the file ID, `DB_SVR1.OJU1`, in a DB storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU1/content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

The following sample request uploads the content for a resource bundle file with the file ID, `besMetadata`, in a BUNDLE storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/content
Content-Type: application/octet-stream
Body: binary stream – zip file with besMetadata bundle
```

Sample API Response

The API returns a 200 OK response code on successfully uploading the content for a file. The response body is empty.

Get File Content

The Get File Content REST API returns the content for a file associated with a file ID.

Use the Get File Content REST API with a BPM, BUNDLE, DB, or TEMP storage.

The API uses the GET method.

Request URL

The Get File Content URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>/content
```

Make the following HTTP GET request to the Get File Content URL:

```
GET http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>/content
```

Sample API Request

The following sample request returns the content for a file with the file ID, `urn:b4p2:22203::file1.txt`, in a BPM storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BPM/
urn:b4p2:22203::file1.txt/content
```

Note: Use the Read Task REST API to retrieve the file ID of a task attachment in a BPM storage.

The following sample request returns the content for a file with the file ID, `DB_SVR1.OJU1`, in a DB storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU1/content
```

Note: Use the Read Record REST API to retrieve the file ID of the file that you attach to a record.

The following sample request returns the content for the resource bundle file with the file ID, `besMetadata`, in a BUNDLE storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/content
```

Sample API Response

The following sample response shows the content for a TXT file in a BPM storage:

```
Test attachment content: file 1
```

The following sample response shows the content for the resource bundle file in a BUNDLE storage:

```
Content-Disposition → attachment; filename=besMetadata.zip
Content-Type → application/octet-stream
Transfer-Encoding → chunked
```

Delete File

The Delete File REST API deletes the file associated with a file ID, which includes the file metadata and content.

Use the Delete File REST API with a BUNDLE, DB, or TEMP storage.

The API uses the DELETE method.

Request URL

The Delete File URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Make the following HTTP DELETE request to the Delete File URL:

```
DELETE http://<host>:<port>/<context>/<database ID>/<storage>/<fileId>
```

Sample API Request

The following sample request deletes the file associated with the file ID, `TEMP_SVR1.1VDVS`, in a TEMP storage, including the file metadata and content:

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS
```

The following sample request deletes the file associated with the file ID, `DB_SVR1.OJU1`, in a DB storage, including the file metadata and content:

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU1
```

The following sample request deletes the resource bundle file with the file ID, `besMetadata`, in a BUNDLE storage, including the file metadata and content:

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata
```

Sample API Response

The API returns a 200 OK response code on successfully deleting a file. The response body is empty.

Preview Promote

The Preview Promote REST API returns a preview of a resulting record if you promote the pending changes.

The API uses the GET method. You can see how a record would look if you apply the pending changes to the record. The API response contains the record with the new values and a change summary with the old values. The API does not return information about data that you delete. Provide the interaction ID of the pending changes in the URL.

Request URL

The Preview Promote URL has the following format:

```
http://<host>:<port>/<context>/<database ID><business entity>/<rowId>?
action=previewPromote&interactionID=<interaction ID>
```

Make the following HTTP GET request to the Preview Promote URL:

```
GET http://<host>:<port>/<context>/<database ID><business entity>/<rowId>?
action=previewPromote&interactionID=<interaction ID>
```

Query Parameters

The interaction ID of the pending changes is a required parameter in the URL.

The following table lists the query parameters:

Parameter	Description
contentMetadata	Metadata for the merge preview. Provide a comma-separated list. You can use the following values: - BVT. Specify the rowid of the record that contains the most trustworthy value to use in the merge preview. Returns information about the cross-reference record and the original record ID. - MERGE. Specify the rowids of the records to merge. Returns information about how the descendant records were merged.
interactionId	Interaction ID of the pending changes.
effectiveDate	Optional. Date for which you want to preview the changes. Use the parameter for timeline-enabled base objects.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample request creates a preview of a root record in the Person business entity:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?
action=previewPromote&interactionId=72300000001000
```

Sample API Response

The following sample response returns a preview of the record with the new values and a change summary of old values:

```
{
    "rowidObject": "233           ",
    "creator": "admin",
    "createDate": "2008-08-12T02:15:02-07:00",
    "updatedBy": "admin",
    "lastUpdateDate": "2015-07-14T03:42:38.778-07:00",
    "consolidationInd": "1",
    "lastRowidSystem": "SYS0          ",
    "dirtyIndicator": "0",
    "interactionId": "72300000001000",
    "hubStateInd": "1",
    "label": "LLOYD,BOB",
    "partyType": "Person",
    "lastName": "LLOYD",
    "firstName": "BOB",
    "displayName": "BOB LLOYD",
    "preferredPhone": {
        "rowidObject": "164           ",
        "$original": {
            "rowidObject": "164           "
        }
    },
    "$original": {
        "label": "DUNN,LLOYD",
        "lastName": "DUNN",
        "firstName": "LLOYD",
        "displayName": "LLOYD DUNN"
```

```
    }
}
```

Promote

The Promote REST API promotes all pending changes made to a record based on the interaction ID of the change request.

The API uses the POST method. Provide the interaction ID as a query parameter.

Request URL

The Promote URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root
record>?action=promote&interactionId=<interaction ID>
```

Make the following HTTP POST request to the Promote URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root
record>?action=promote&interactionId=<interaction ID>
```

Query Parameter

The interaction ID of the pending changes is a required parameter. The API uses the interaction ID to find all the records related to the pending changes.

Sample API Request

The following sample request promotes all the pending changes based on the interaction ID of the change request:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038246?
action=promote&interactionId=69120000294000
```

Sample API Response

The following sample response contains the row ID of the record after promoting the pending changes:

```
{
            Person: {
                        rowidObject: "1038246"
                    }
}
```

Delete Pending

The Delete Pending REST API deletes all pending changes you make to a record based on the interaction ID of the change request.

The API uses the DELETE method and returns the row ID of the record.

Request URL

The Delete Pending URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=deletePending&interactionId=<interaction ID>
```

Make the following DELETE request to the Delete Pending URL:

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=deletePending&interactionId=<interaction ID>
```

Query Parameter

Provide the interaction ID of the pending changes that you want to delete.

Sample API Request

The following sample request deletes all the pending changes based on the interaction ID of the change request:

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?
action=deletePending&interactionId=72300000001000
```

Sample API Response

The following sample response contains the row ID of the record after deleting the pending changes:

```
{
    Person: {
        rowidObject: "233"
    }
}
```

Preview Merge

The Preview Merge REST API returns a preview of a consolidated root record if you merge two or more root records.

The API uses the POST method and accepts a list of root records and field-level overrides to return a preview of the merged record. The row ID of the target record is a required parameter.

Request URL

The Preview Merge URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge
```

The following Preview Merge URL format specifies the number of child levels to return:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge&depth=2
```

Make the following HTTP POST request to the Preview Merge URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=previewMerge
```

Note: In the request body, add the keys property and specify the root records that you want to merge with the target record.

To override matches for the child records, add the contentMetadata parameter:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=previewMerge&contentMetadata=MERGE/<json/xml>
```

Note: In the request body, add the overrides property and specify the merge overrides.

To specify the media type of the data you want to send with the request, add the Content-Type header:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=previewMerge
Content-Type: application/<json/xml>
```

Query Parameters

The row ID of the target record is a required parameter.

You can use the following query parameters:

Parameter	Description
contentMetadata	Metadata for the merge preview. Provide a comma-separated list. You can use the following values: - BVT. Returns information about the winning cross-reference record and the original record ID. - MERGE. Returns information about how the descendant records were merged.
depth	Number of child levels to return.
effectiveDate	Date for which you want to generate the preview.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Request Body

Before you begin, use the Read Matched Records API to determine which matched records you can merge with the original root record. Send the list of records in the request body for the Preview Merge API.

You can override field values in the root record. For example, if none of the matched root records contain the correct spelling of the first name, you can specify the correct first name in the request body. Also, you can remove matched records or specify other matching records.

Use the following properties in the request body:

Properties / Elements	Type	Description
keys	array	Required. An ordered list of the matched root records that you want to merge. You can identify the records either by row ID or by a combination of the source system and the source key.
overrides	object	Overrides the field values in a root record and the matches for child records.
MERGE	object	Overrides the field values in child records that you want to merge. Add the type of child record within the `overrides` object and then add the `MERGE` object.

The following JSON code sample identifies a root record to merge with the target root record:

```
{
  keys: [
    {
       rowid: "P2"
    }
  ]
}
```

The following code shows how to override a field in the Party root record and how to override the merge candidates for Telephone child records:

```
{
  keys: [
    {
      rowid: "P2"
    }
  ]
  overrides: {
    Party: {
      rowidObject: "P1",
      firstName: "Serge",   //override the value for the first name
      Telephone: {   // override which Telephone child records to merge
        item:[
          {
            rowidObject: "T1",
            MERGE: {
              item: [   // to remove the original merge candidates, specify null
                null,
                null
              ],
              $original: {
                item: [
                  {key:{rowid: "T2"}},
                  {key:{rowid: "T3"}}
                ]
              }
            }
          },
          {
            rowidObject: "T4",
            MERGE: {
              item: [   // to add or change merge candidates, specify matched records
                {key:{rowid: "T2"}}
              ],
              $original: {
                item: [
                  null
                ]
              }
            }
          }
        ]
      }
    }
  }
}
```

Sample API Request

The following sample request returns the preview of a consolidated record:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245?
action=previewMerge
```

```
{
  keys: [
    {
        rowid: "2478246"
    },
   {
        rowid: "2478230"
    }
  ],
  overrides: {
    Person: {
       firstName: "Charlie"
    }
  }
}
```

Sample API Response

The following sample response shows the preview of the consolidated record:

```
{
    "Person": {
        "rowidObject": "2478245       ",
        "partyType": "Person",
        "lastName": "Smith",
        "firstName": "Charlie",
        "displayName": "ALICE SMITH"
    }
}
```

Update Pending Merge

To save changes to records that are part of a pending merge task, such as record value overrides, use the Update Pending Merge REST API. The API saves the changes based on the interaction ID of the records.

The API uses the POST method.

Request URL

The path component of the request URL must include the row ID of the target record.

The Update Pending Merge URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>
/<rowid of the root record>?
action=updatePendingMerge&interactionId=<interaction ID>
```

Make the following HTTP POST request to the Update Pending Merge URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>
/<rowid of the root record>?
action=updatePendingMerge&interactionId=<interaction ID>
```

In the request body, add the keys and specify the root records that you want to merge with the target root record. You can also specify the child records for which you want to override matches.

Query Parameter

The following table describes the query parameters that you can use in the URL:

Parameter	Description
action	Required. Saves changes, such as field-level overrides, to a pending merge task. Set to `updatePendingMerge`, and use the parameter with the `interactionId` parameter. For example, use the following query to save changes to the Person business entity records that are pending the merge action: `Person?action=updatePendingMerge&interactionId`
interactionId	Required. Interaction ID of the pending merge task.

Request Body

Before you use the Update Pending Merge API, use the Read Matched Records REST API to determine the matched records that you can merge with the target root record. Send the list of records in the request body for the Update Pending Merge API.

You can override field values in the root record. For example, if none of the matched root records contain the correct spelling of the first name, you can specify the correct first name in the request body. Also, you can remove matched records or specify other matching records.

Use the following properties in the request body:

Properties / Elements	Type	Description
keys	array	Required. An ordered list of the matched root records that you want to merge. You can identify the records either by row ID or by a combination of the source system and the source key.
overrides	object	Overrides the field values in a root record and the matches for child records.
MERGE	object	Overrides the field values in child records that you want to merge. Add the type of child record within the `overrides` object and then add the `MERGE` object.

The following JSON code sample identifies two root records to merge with the target root record:

```
{
  keys: [
    {rowid: "2478246"},
    {rowid: "2478230"}
  ]
}
```

The following sample request body shows how to override a field in the Party root record and how to override the matched records for Telephone child records:

```
{
  keys: [
    {
      rowid: "2478246"
    }
  ]
  overrides: {
    Party: {
      rowidObject: "2478230",
      firstName: "Charlie",   //Override the value for the first name
      Telephone: {   // Specifies the Telephone child records to merge
        item:[
          {
            rowidObject: "2511",
            MERGE: {
              item: [   // To remove the original merge candidates, specify null
                null,
                null
              ],
              $original: {
                item: [
                  {key:{rowid: "2822"}},
                  {key:{rowid: "2733"}}
                ]
              }
            }
          },
          {
            rowidObject: "2644",
            MERGE: {
```

```
            item: [   // To add or change merge candidates, specify matched records
              {key:{rowid: "2822"}}
            ],
          $original: {
            item: [
              null
            ]
          }
        }
      ]
    }
  }
}
```

Sample API Request

The following sample request overrides the first name field of the target record with the value Charlie:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245?
action=updatePendingMerge&interactionId=3982462873645
```

```
{
  keys: [
    {
        rowid: "2478246"
    },
   {
        rowid: "2478230"
    }
  ],
  overrides: {
    Person: {
       firstName: "Charlie"
    }
  }
}
```

Sample API Response

The following sample response contains the row ID of the target record for which the merge action is pending:

```
{
    "Person": {
        "key": {
            "rowid": "2478245"
        },
        "rowidObject": "2478245"
    }
}
```

Pending Merge

The Pending Merge REST API updates all pending merge tasks you make to a record based on the interaction ID of the change request. Pending Merge allows you to defer merge operations until the workflow process grants approval for all merge tasks.

The API uses the POST method and returns the row ID of the record.

Request URL

The Pending Merge URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=PendingMerge&interactionId=<interaction ID>
```

Make the following HTTP POST request to the Pending Merge URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=Merge
```

Query Parameter

The interaction ID of the pending merge is a required parameter.

Sample API Request

The following sample request updates all pending merge tasks associated with the interaction ID:

```
POST /Person/123?action=pendingMerge&interactionId=123
```

Sample API Response

The following sample response contains the row IDs of the affected root base objects:

```
{
    keys: [{rowid: "456"}, {rowid: "789"}],
    overrides: {...}
}
```

PromoteMerge

The Promote Merge REST API runs all pending merge tasks associated with the interaction ID of the change request.

The API uses the POST method and returns the row ID of the winning record.

Request URL

The Promote Merge URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=PromoteMerge&interactionId=<interaction ID>
```

Make the following HTTP POST request to the Promote Merge URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=Merge
```

Query Parameter

The interaction ID of the pending merge task is a required parameter. The API uses the interaction ID to find all pending merge tasks and runs the merges.

Sample API Request

The following sample request promotes all pending merge tasks associated with the interaction ID:

```
POST /Person/123?action=promoteMerge&interactionId=123
```

Sample API Response

The following sample response contains the row IDs of the records after promoting the pending merge tasks:

```
POST /Person/123?action=promoteMerge&interactionId=123
```

Merge Records

The Merge Records REST API merges two or more root records to create a single consolidated record. The row ID of the consolidated record is the row ID of the record to which you merge the other records.

The API uses the POST method. You can specify field-level overrides for the merged record in the request body.

Request URL

The Merge Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=Merge
```

Make the following HTTP POST request to the Merge Records URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=Merge
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the
record>?action=Merge
Content-Type: application/<json/xml>
```

Query Parameters

The following table lists the parameters that you can use in the URL:

Parameter	Description
taskComment	Add a comment to the workflow task triggered by the API.
taskAttachments	If task attachments are enabled, attach a file to the workflow task triggered by the API.

Request Body

Before you begin, use the Preview Merge API to preview the results of merging the selected root records. When you are happy with the preview, use the same properties in the request body for the Merge Records API.

You can override field values in the root record. For example, if none of the matched root records contain the correct spelling of the first name, you can specify the correct first name in the request body. Also, you can remove matched records or specify other matching records.

Use the following properties in the request body:

Properties / Elements	Type	Description
keys	array	Required. An ordered list of the matched root records that you want to merge. You can identify the records either by row ID or by a combination of the source system and the source key.
overrides	object	Overrides the field values in a root record and the matches for child records.
MERGE	object	Overrides the field values in child records that you want to merge. Add the type of child record within the `overrides` object and then add the `MERGE` object.

The following JSON code sample identifies two root records to merge with the target root record:

```
{
  keys: [
    {rowid: "2478246"},
    {rowid: "2478269"}
  ]
}
```

For an example of how to use the `overrides` and `MERGE` properties with the Merge Records API, see the request body for the Merge Preview API.

Sample API Request

The following sample request merges records to form a consolidated record. The request adds a comment and attachment to the workflow task triggered by the API.

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245?
action=merge&taskComment=Read my comment&taskAttachments=TEMP_SVR1.29OT8,TEMP_SVR1.29OT9
Content-Type: application/<json/xml>

{
  keys: [
    {
        rowid: "2478246"
    }
  ],
  overrides: {
    Person: {
       firstName: "Charlie"
    }
  }
}
```

Sample API Response

The following sample response shows the consolidated record:

```
{
    "Person": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/
2478245",
                "rel": "self"
            }
        ],
        "key": {
            "rowid": "2478245"
        },
        "rowidObject": "2478245"
```

```
    }
}
```

Unmerge Records

The Unmerge Records REST API unmerges a root record into the individual root records that existed before you merged the records.

The API uses the POST method.

Request URL

The Unmerge Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=unmerge
```

Make the following HTTP POST request to the Unmerge Records URL:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=unmerge
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=unmerge
Content-Type: application/<json/xml>
```

Query Parameters

The following table lists the parameters that you can use in the URL:

Parameter	Description
taskComment	Add a comment to the workflow task triggered by the API.
taskAttachments	If task attachments are enabled, attach a file to the workflow task triggered by the API.

Request Body

Send the list of the records that you want to unmerge from the consolidated record in the request body. Use the xref row ID or the source system and the source key to specify the records.

Use the Read Record API to get the xref rowId of the record to unmerge. The following sample request retrieves the XREF metadata of a record:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2638243?
contentMetadata=XREF
```

Sample API Request

The following sample request unmerges a record from the consolidated record. The request adds a comment and attachment to the workflow task triggered by the API.

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478248?
action=unmerge&taskComment=Read my
comment&taskAttachments=TEMP_SVR1.29OT8,TEMP_SVR1.29OT9

{
        rowid: "4880369"
}
```

Sample API Response

The following sample response shows the record that you unmerge from the consolidated record:

```
{
    "Person": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/
2478249",
                "rel": "self"
            }
        ],
        "key": {
            "rowid": "2478249       "
        },
        "rowidObject": "2478249       "
    }
}
```

Read a Relationship

The Read Relationship REST API returns the details of a relationship record, such as the party types, rowIDs, and display names of the two records.

The API uses the GET method.

Request URL

The Read Relationship URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID of the relationship
record>
```

Make the following HTTP GET request to the URL:

```
GET http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID of the
relationship record>
```

Query Parameters

The following table lists the query parameters:

Parameter	Description
suppressLinks	Optional. Indicates whether the parent-child links are visible in the API response. Set the parameter to true to suppress all the parent-child links in the response. Set the parameter to false to not display the links in the API response. Default is false.
depth	Optional. Number of child levels to return.

Sample API Request

The following sample request returns the details of the relationship record with the row ID 85, which is of the relationship type `ProductGroupProductGroupIsParentOfProductProducts`:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85
```

The following sample request returns the details with depth equal to 2:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85?depth=2
```

Sample API Response

The following example shows the details of the relationship record with the row ID 85, which is of the relationship type `ProductGroupProductGroupIsParentOfProductProducts`:


```
{
  "link": [
    {
      "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85.json",
      "rel": "self"
    },
    {
      "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85.json?depth=2",
      "rel": "children"
    }
  ],
  "rowidObject": "85            ",
  "label": "ProductGroup Product Group is parent of Product Products",
  "rowidRelType": "9             ",
  "rowidHierarchy": "3              ",
  "from": {
    "link": [
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85.json",
        "rel": "parent"
      },
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/from/86.json",
        "rel": "self"
      },
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/from/86.json?depth=2",
        "rel": "children"
      }
    ],
    "rowidObject": "86            ",
    "label": "ProductGroup",
    "productType": "Product Group",
    "productNumber": "Presenter2",
    "productName": "Presenter",
    "productDesc": "Presenter Family",
    "productTypeCd": {
      "link": [
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/from/86.json",
          "rel": "parent"
        },
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/from/86/productTypeCd.json?depth=2",
          "rel": "children"
        },
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/from/86/productTypeCd.json",
          "rel": "self"
        }
      ],
      "productType": "Product Group"
    }
  },
  "to": {
    "link": [
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
```

```
ProductGroupProductGroupIsParentOfProductProducts/85/to/66.json",
        "rel": "self"
      },
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/to/66.json?depth=2",
        "rel": "children"
      },
      {
        "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85.json",
        "rel": "parent"
      }
    ],
    "rowidObject": "66            ",
    "label": "Products",
    "productType": "Product",
    "productNumber": "931307-0403",
    "productName": "2.4 GHz Cordless Presenter",
    "productDesc": "A cordless presenter to streamline your delivery.",
    "productTypeCd": {
      "link": [
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/to/66/productTypeCd.json",
          "rel": "self"
        },
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/to/66.json",
          "rel": "parent"
        },
        {
          "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85/to/66/productTypeCd.json?depth=2",
          "rel": "children"
        }
      ],
      "productType": "Presenter"
    }
  }
}
```


Create a Relationship

The Create Relationship REST API creates a relationship between the specified records. To create a relationship between records, relationships must exist between the business entities to which the records belong. For example, if you want to specify a relationship between Informatica and John Smith, a relationship must exist between the Organization and the Person business entities. You must send the relationship data in the request body.

The API uses the PUT and the POST methods.

Request URL

The Create Relationship REST URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<relationship>?systemName=<name of the
source system>
```

Note: The name of the source system is a required parameter in the URL.

Make the following HTTP POST or PUT request to the URL:

```
POST http://<host>:<port>/<context>/<database ID>/<relationship>?systemName=<name of the
source system>
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
Content-Type: application/<json/xml>
```

URL Parameters

The name of the source system is a required parameter in the request URL.

Request Body

Send data for the relationship record in the REST request body. Use the JSON format or the XML format to send data. Provide the required parameter values in the request body.

Sample API Request

The following sample request creates the `OrganizationEmploysPerson` relationship between an Organization business entity with row ID 101 and a Person business entity with row ID 1101:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/OrganizationEmploysPerson?
systemName=SFA
Content-Type: application/json

{
  "from": {
    "rowidObject": "101"
    },

  "to": {
    "rowidObject": "1101          "
    },

    "relName": "Documentation",
    "relDesc": "Writer"
}
```

The `OrganizationEmploysPerson` relationship defines a relationship from an Organization business entity to a Person business entity. The `from` element specifies the record from which the relationship originates and the `to` element specifies the record at which the relationship ends.

Sample API Response

The following sample response shows the response header and body after successfully creating a relationship between an Organization business entity with row ID 101 and a Person business entity with row ID 1101:

```
{
        "OrganizationEmployesPerson": {
        "key": {
        "rowid": "414721"
        "sourceKey": "SVR1.1E7UW"
 }-

"rowidObject": "414721"

    "from": {
                    "key": {
                                "rowid": "101 "
                                                }-
        "rowidObject": "101 "
        }-

 "to": {
                "key": {
                            "rowid": "1101 "
```

```
        }-

"rowidObject": "1101 "
               }-

    }-

}
```

Update a Relationship

The Update Relationship REST API updates the relationship between two records. The API updates the additional attributes defined for the relationship.

The API uses the POST and the PUT methods.

Request URL

Update Relationship URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID>?systemName=<name of the source system>
```

Note: The name of the source system is a required parameter in the URL.

Make the following HTTP POST or PUT call to the URL:

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID>?systemName=<name of the source system>
```

Add the Content-Type header to specify the media type of the data you want to send with the request.

Request Body

Send the updates to the relationship record in the request body. Use the JSON format or the XML format to send data. Provide the required parameter values in the request body.

Sample API Request

The relationship with the row ID 414721 is a OrganizationEmploysPerson relationship between an Organization entity with row ID 101 and a Person entity with row ID 1101.

The following sample request updates the relationship record with the row ID 414721:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/OrganizationEmploysPerson/414721?systemName=SFA
Content-Type: application/json
{
  "from": {
    "rowidObject": "101"
    },

  "to": {
    "rowidObject": "1101          "
    },

    "relName": "Development",
    "relDesc": "Software Engineer",

                "$original":
                {"relName": "Documentation",
                    "relDesc": "Writer"}


}
```

Sample API Response

The following sample response is received after successfully updating the relationship with the row ID 414721:

```
{
        "OrganizationEmployesPerson": {
                "key": {
                                "rowid": "414721"
                                "sourceKey": "SVR1.1E7UW"
                                        }-
            "rowidObject": "414721"
        "from": {
            "key": {
                            "rowid": "101"
                                    }-
                "rowidObject": "101"
                }-
        "to": {
                "key": {
                            "rowid": "1101 "
                                }-
        "rowidObject": "1101 "
                       }-
            }-
}
```

Delete a Relationship

The Delete Relationship REST API deletes the relationship between two records.

The API uses the DELETE method.

Request URL

The Delete Relationship URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<rowID of the relationship record>?systemName=<name of the source system>
```

Note: The name of the source system as a required parameter in the URL.

Make the following HTTP DELETE request to the Delete URL:

```
DELETE http://<host>:<port>/<context>/<database ID>/<relationship>/<rowID of the relationship record>?systemName=<name of the source system>
```

Query Parameter

The name of the source system is a required URL parameter. Use the `systemName` parameter to specify the source system.

Sample API Request

The following sample request deletes a relationship record with the row ID 414721:

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/OrganizationEmploysPerson/414721?systemName=SFA
```

Sample API Response

The following sample response shows the response after successfully deleting the relationship record with the row ID 414721:

```
{
"OrganizationEmployesPerson": {
        "key": {
                "rowid": "414721"
                }-
"rowidObject": "414721"
            }-
}
```

Get Related Records

The Get Related Records REST API returns a list of records related to a specified root record based on the relationships that you have configured. The API also returns the details of the relationships.

The API uses the GET method.

Request URL

The Get Related Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=related
```

Make the following HTTP GET request to the Get Related Records URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=related
```

Query Parameters

You can append the query parameters to the request URL.

The following table lists the query parameters that you can use:

Parameter	Description
recordsToReturn	Number of records for the `many` child that you want to read.
searchToken	Search token to fetch subsequent pages of the result set.
returnTotal	Returns the number of records in the result set. Set to true to get the number of records in the result set. Default is false.

Filter Parameters

You can append parameters to the URL to filter the related records.

The following table lists the filter parameters that you can use:

Parameter	Description
recordStates	A comma-separated list of states of records that you want to retrieve. The supported record states are ACTIVE, PENDING, and DELETED. Default is ACTIVE. For example, the `/Party/123?action=related&recordStates=ACTIVE,PENDING` query returns records that are active or pending.
entityLabel	Label of the entity.
relationshipLabel	Label of the relationship.
entityType	Comma-separated list of entity types. For example, the `entityType=Person,Organization` list returns related records of the `Person` and `Organization` entity type.
relationshipType	Comma-separated list of relationship types. For example, the `relationshipType=Employee,Employer` list returns related records of the `Employee` and `Employer` relationship types.

Note: If you specify multiple filter conditions, the result contains all the records that satisfy the AND condition.

Response Body

The response body contains the list of related records, details of the related records and the relationships, and a search token. Use the search token to fetch the subsequent pages of the results.

Sample API Request

The following sample request fetches the related records and relationships configured for the Organization business entity with the row ID 101:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Organization/101?action=related
```

Sample API Response

The following sample response shows the related records and relationships for the Organization business entity type with the row ID 101:

```
{
  "link": [],
  "firstRecord": 1,
  "pageSize": 10,
  "searchToken": "SVR1.1H7YB",
  "relatedEntity": [
    {
      "businessEntity": {
        "SecurePerson": {
          "link": [
            {
              "href": "http://10.21.43.42:8080/cmx/cs/localhost-orcl-ds_ui1/
SecurePerson/1101",
              "rel": "self"
            }
```

```
          ],
          "rowidObject": "1101          ",
          "creator": "admin",
          "createDate": "2008-11-11T21:22:20-08:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2012-03-29T19:03:19-07:00",
          "consolidationInd": "1",
          "lastRowidSystem": "SYS0          ",
          "dirtyIndicator": "0",
          "interactionId": "20003000",
          "hubStateInd": "1",
          "partyType": "Person",
          "lastName": "Obama",
          "firstName": "Barack"
        }
      },
      "entityLabel": "Obama,Barack",
      "relationshipLabel": "Organization employes SecurePerson",
      "relationship": {
        "rowidObject": "414721        ",
        "creator": "admin",
        "createDate": "2016-10-17T01:58:12.436-07:00",
        "updatedBy": "admin",
        "lastUpdateDate": "2016-10-19T01:40:28.830-07:00",
        "consolidationInd": "4",
        "lastRowidSystem": "SFA           ",
        "interactionId": "1476866426786",
        "hubStateInd": "1",
        "rowidRelType": "101           ",
        "rowidHierarchy": "1             ",
        "relName": "Documentation",
        "relDesc": "Writer"
      },
      "entityType": "SecurePerson",
      "relationshipType": "OrganizationEmployesSecurePerson"
    },
    {
      "businessEntity": {
        "SecurePerson": {
          "link": [
            {
              "href": "http://10.21.43.42:8080/cmx/cs/localhost-orcl-ds_ui1/
SecurePerson/114",
              "rel": "self"
            }
          ],
          "rowidObject": "114           ",
          "creator": "admin",
          "createDate": "2008-08-11T23:00:55-07:00",
          "updatedBy": "Admin",
          "lastUpdateDate": "2008-08-12T02:59:17-07:00",
          "consolidationInd": "1",
          "lastRowidSystem": "Legacy        ",
          "dirtyIndicator": "0",
          "hubStateInd": "1",
          "partyType": "Person",
          "lastName": "HERNANDEZ",
          "displayName": "ALEJANDRO  HERNANDEZ                                ",
          "firstName": "ALEJANDRO"
        }
      },
      "entityLabel": "HERNANDEZ,ALEJANDRO",
      "relationshipLabel": "Organization employes SecurePerson",
      "relationship": {
        "rowidObject": "434721        ",
        "creator": "admin",
        "createDate": "2016-10-19T01:49:03.415-07:00",
        "updatedBy": "admin",
        "lastUpdateDate": "2016-10-19T01:49:03.415-07:00",
        "consolidationInd": "4",
        "lastRowidSystem": "SFA           ",
```

```
    "hubStateInd": "1",
    "rowidRelType": "101           ",
    "rowidHierarchy": "1             ",
    "relName": "Documentation",
    "relDesc": "Writer"
  },
  "entityType": "SecurePerson",
  "relationshipType": "OrganizationEmployesSecurePerson"
},
{
  "businessEntity": {
    "Person": {
      "link": [
        {
          "href": "http://10.21.43.42:8080/cmx/cs/localhost-orcl-ds_ui1/Person/1101",
          "rel": "self"
        }
      ],
      "rowidObject": "1101          ",
      "creator": "admin",
      "createDate": "2008-11-11T21:22:20-08:00",
      "updatedBy": "admin",
      "lastUpdateDate": "2012-03-29T19:03:19-07:00",
      "consolidationInd": "1",
      "lastRowidSystem": "SYS0          ",
      "dirtyIndicator": "0",
      "interactionId": "20003000",
      "hubStateInd": "1",
      "partyType": "Person",
      "lastName": "Obama",
      "firstName": "Barack"
    }
  },
  "entityLabel": "Obama,Barack",
  "relationshipLabel": "Organization employes Person",
  "relationship": {
    "rowidObject": "414721        ",
    "creator": "admin",
    "createDate": "2016-10-17T01:58:12.436-07:00",
    "updatedBy": "admin",
    "lastUpdateDate": "2016-10-19T01:40:28.830-07:00",
    "consolidationInd": "4",
    "lastRowidSystem": "SFA           ",
    "interactionId": "1476866426786",
    "hubStateInd": "1",
    "rowidRelType": "101           ",
    "rowidHierarchy": "1             ",
    "relName": "Documentation",
    "relDesc": "Writer"
  },
  "entityType": "Person",
  "relationshipType": "OrganizationEmployesPerson"
},
{
  "businessEntity": {
    "Person": {
      "link": [
        {
          "href": "http://10.21.43.42:8080/cmx/cs/localhost-orcl-ds_ui1/Person/114",
          "rel": "self"
        }
      ],
      "rowidObject": "114           ",
      "creator": "admin",
      "createDate": "2008-08-11T23:00:55-07:00",
      "updatedBy": "Admin",
      "lastUpdateDate": "2008-08-12T02:59:17-07:00",
      "consolidationInd": "1",
      "lastRowidSystem": "Legacy        ",
      "dirtyIndicator": "0",
      "hubStateInd": "1",
```

```
            "partyType": "Person",
            "lastName": "HERNANDEZ",
            "displayName": "ALEJANDRO  HERNANDEZ                                ",
            "statusCd": "A ",
            "firstName": "ALEJANDRO"
          }
        },
        "entityLabel": "HERNANDEZ,ALEJANDRO",
        "relationshipLabel": "Organization employes Person",
        "relationship": {
          "rowidObject": "434721        ",
          "creator": "admin",
          "createDate": "2016-10-19T01:49:03.415-07:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2016-10-19T01:49:03.415-07:00",
          "consolidationInd": "4",
          "lastRowidSystem": "SFA           ",
          "hubStateInd": "1",
          "rowidRelType": "101           ",
          "rowidHierarchy": "1             ",
          "relName": "Documentation",
          "relDesc": "Writer"
        },
        "entityType": "Person",
        "relationshipType": "OrganizationEmployesPerson"
      }
    ]
}
```


Read Matched Records

The Read Matched Records REST API returns records that match a specified root record. You can review the list of records to determine which records you can merge with the original root record. You can use the Merge Records API to merge the records.

The API uses the GET method.

Request URL

The Read Matched Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
```

Make the following HTTP GET request to the Read Matched Records URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
```

Response Body

The response body contains the number of records that match the specified record, the details of the matching records, and a search token. Use the search token to fetch the subsequent pages of the match result.

Sample API Request

The following sample request searches the business entity for records that match a record:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038245?action=matched
```

Sample API Response

The following sample response shows the details of the record that matches the specified record:


```
{
    "firstRecord": 1,
    "recordCount": 1,
    "pageSize": 10,
    "searchToken": "SVR1.AU5HE",
    "matchedEntity": [
        {
            "businessEntity": {
                "Person": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
Person/1038246",
                            "rel": "self"
                        }
                    ],
                    "rowidObject": "1038246           ",
                    "creator": "admin",
                    "createDate": "2008-08-12T02:15:02-07:00",
                    "updatedBy": "Admin",
                    "lastUpdateDate": "2008-08-12T02:59:17-07:00",
                    "consolidationInd": "1",
                    "lastRowidSystem": "SFA           ",
                    "dirtyIndicator": "0",
                    "hubStateInd": "1",
                    "partyType": "Person",
                    "lastName": "BATES",
                    "firstName": "DAISY",
                    "displayName": "DAISY BATES"
                }
            },
            "matchRule": "PUT"
        }
    ]
}
```


Update Matched Records

The Update Matched Records REST API creates or updates a record in the match table. The match table contains the pairs of matched records in a business entity after you run a match process on the business entity. Use the API to add records that qualify for a merge with the specified record.

The API uses the PUT method.

Request URL

The Update Matched Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
```

Make the following HTTP PUT request to the Update Matched Records URL:

```
PUT http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
PUT http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
Content-Type: application/<json/xml>
```

Request Body

Send the list of records that match the specified record in the request body. Use the row ID or the source system and source key to specify the records.

Sample API Request

The following sample adds a record in the match table:

```
PUT http://localhost:8080/cmx/cs/localhost-ORCL-DS_UI1/Person/1038245?action=matched

{
    keys: [
       {
          rowid: "1038246"
       }
   ]
}
```

Sample API Response

The API returns a 200 OK response on successfully creating a record in the match table. The response body is empty.

Delete Matched Records

The Delete Matched Records REST API deletes matched records associated with a root record from the match table.

The API uses the DELETE method.

Request URL

The Delete Matched Records URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
```

Make the following HTTP DELETE request to the Delete Matched Records URL:

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=matched
```

Add the Content-Type header to specify the media type of the data you want to send with the request:

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=matched
Content-Type: application/<json/xml>
```

Request Body

Send the list of records that you want to delete from the match table in the request body.

Sample API Request

The following sample request deletes a record that matches the specified root record from the match table:

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038245?action=matched

{
    keys: [
       {
          rowid: "1038246"
       }
```

```
    ]
}
```

Sample API Response

The API returns a 200 OK response on successfully deleting a record from the match table. The response body is empty.

Get Record History Events

The Get Record History Events REST API returns a list of history events, or groups of history events, associated with a record. Send the record ID in the request body.

The API uses the GET method to return the following data for each group of history events:

- Start and end date for the group
- Number of events in the group

The API returns the following data for each history event:

- History event ID
- Date of the change
- User who made the change
- List of history tables that are affected by the change
- List of record nodes that are affected by the change

Request URL

The Get Record History Events URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=listHistoryEvents
```

Make the following HTTP GET request to the Get Record History Events URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=listHistoryEvents
```

Query Parameters

The ID of the record is a required parameter. The API uses the record ID to find all related history events.

The following table lists the query parameters:

Parameter	Description
startDate and endDate	Optional. Date range for which you want to retrieve the data. If you specify a date range, the response contains only events within this range.
granularity	Optional. The level of detail to group history events. If specified, the response groups history events. Otherwise, the response does not group history events. Use one of the following values: - YEAR - QUARTER - MONTH - WEEK - DAY - HOUR - MINUTE - AUTO
recordStates	Optional. Record states returned in the list of history events. Provide a comma-separated list. You can use the following values: - ACTIVE - PENDING - DELETED
contentMetadata	Optional. Metadata for the list of history events. Provide a comma-separated list. You can use the following values: - XREF - PENDING_XREF - DELETED_XREF - HISTORY - MATCH - BVT - TRUST
children	Optional. Comma-separated list of the child node names. If specified, the response contains the child node names.
order	Optional. Comma-separated list of field names with an optional prefix of + or –. The prefix + indicates to sort the results in ascending order, and the prefix – indicates to sort the results in descending order. Default is +. When you specify more than one parameter, the result set is ordered by the parameter that is first in the list, followed by the next.
fields	Optional. Comma-separated list of business entity fields. If specified, the response only contains listed fields.
filter	Optional. Filter expression.
depth	Optional. Number of child levels to return.
recordsToReturn	Optional. Specifies the number of rows to return.
searchToken	Optional. Specifies the search token returned with previous request.

Parameter	Description
returnTotal	Optional. If set to `true`, returns the number of records in the result. Default is `false`.
firstRecord	Optional. Specifies the first row in the result.
changeType	Optional. Specifies the types of change returned in the result. Provide a comma-separated list. You can use the following values: - BO - XREF - BVT - MERGE - MERGE_AS_SOURCE - MERGE_AS_TARGET - UNMERGE_AS_SOURCE - UNMERGE_AS_TARGET

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Sample API Request

The following sample request returns all merges, grouped by year, for a record since January 1, 2000:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/123?
action=listHistoryEvents&startDate=2010-01-01&granularity=YEAR&depth=2&changeType=MERGE
```

Sample API Response

The following sample response lists the merges for the specified record since January 1, 2000:

```
{
    firstRecord: 1,
    recordCount: 2
    item: [
        {
            link: [ // you can use links directly to get event list
                {rel: "events", href: "/Person/123?
action=listHistoryEvents&startDate=2000-01-01&endDate=2001-01-01&depth=2&changeType=MERGE
"}
            ],
            startDate: "2000-01-01",
            endDate: "2001-01-01",
            eventCount: 123
        },
        // no events in 2001, 2002, ... 2009
        {
            link: [
                {rel: "events", href: "/Person/123?
action=listHistoryEvents&startDate=2010-01-01&endDate=2011-01-01&depth=2&changeType=MERGE
"}
            ],
            startDate: "2010-01-01",
            endDate: "2011-01-01",
            eventCount: 23
        }
        // no events in 2011, ..., 2016
    ]
}
```

Get Event Details

The Get Event Details REST API returns details of a specific history event associated with a record. The API returns details such as the type of change made, and the values before and after the change. Send the record ID and history event ID in the request body.

The API uses the GET method.

Request URL

The Get Event Details URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=getHistoryEventDetails
```

Make the following HTTP GET request to the Get Event Details URL:

```
GET http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?
action=getHistoryEventDetails
```

Query Parameters

Before you begin, use the Get Record History Events API to list the history events associated with a record. From the results returned, use the history event ID and the record ID as query parameters.

The following table lists the query parameters:

Parameter	Description
eventID	Required. The Get Record History Events API returns the event IDs for history events.
recordStates	Optional. Record states returned in the list of history events. Provide a comma-separated list. You can use the following values: - ACTIVE - PENDING - DELETED
contentMetadata	Optional. Metadata for the list of history events. Provide a comma-separated list. You can use the following values: - XREF - PENDING_XREF - DELETED_XREF - HISTORY - MATCH - BVT - TRUST
children	Optional. Comma-separated list of child node names. If specified, the response contains the child node names.
order	Optional. Comma-separated list of field names with an optional prefix of + or –. The prefix + indicates to sort the results in ascending order, and the prefix – indicates to sort the results in descending order. Default is +. When you specify more than one parameter, the result set is ordered by the parameter that is first in the list, followed by the next.

Parameter	Description
fields	Optional. Comma-separated list of business entity fields. If specified, the response only contains listed fields.
filter	Optional. Filter expression.
depth	Optional. Number of child levels to return.
recordsToReturn	Optional. Specifies the number of rows to return.
searchToken	Optional. Specifies the search token returned with previous request.
returnTotal	Optional. If set to `true`, returns the number of records in the result. Default is `false`.
firstRecord	Optional. Specifies the first row in the result.

Sample API Request

The following sample request returns information for a history event:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/123?
action=getHistoryEventDetails&eventId=2016-07-14T02%3A01%3A24.529%2B0000
```

Sample API Response

The following sample response shows the details of the specified event:

```
{
  "eventId": "2016-07-14T02:01:24.529+0000",
  "eventDate": "2016-07-14T02:01:24.529Z",
  "user": "admin",
  "changeType": [
    "BVT",
    "BO",
    "UNMERGE_AS_TARGET"
  ],
  "businessEntity": {
    "Person": {
      "rowidObject": "438243        ",
      "creator": "datasteward1",
      "createDate": "2016-07-08T20:42:47.402Z",
      "updatedBy": "admin",
      "lastUpdateDate": "2016-07-14T01:42:50.841Z",
      "consolidationInd": 1,
      "lastRowidSystem": "SYS0          ",
      "hubStateInd": 1,
      "label": "BE,AC",
      "partyType": "Person",
      "lastName": "BE",
      "displayName": "AC BE",
      "firstName": "AC"
    }
  }
}
```

Get DaaS Metadata

The Get DaaS Metadata REST API returns information about a DaaS provider, such as the name, the type, the business entity it works with, and the list of required fields.

The API uses the GET method.

Request URL

The Get DaaS Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/meta/daas/<providerName>
```

Make the following HTTP GET request to the Get DaaS Metadata URL:

```
GET http://<host>:<port>/<context>/<database ID>/meta/daas/<providerName>
```

Query Parameter

The `providerName` parameter is a required parameter. The parameter is the name of the configured DaaS provider.

Sample API Request

The following sample request returns the metadata information of the `dcp` DaaS provider:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/meta/daas/dcp
```

The following sample request returns the metadata information of the `ondemand` DaaS provider:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/meta/daas/ondemand
```

Sample API Response

The following example shows the metadata information of the `dcp` DaaS provider in JSON format:

```
{
                    providerName: "dcp",
                    providerType: "READ",
                    businessEntity: "Organization",
                    systemName: "SFA",
                    requiredFields: [
                                    "dunsNumber"
                    ]
}
```

The following example shows the metadata information of the `ondemand` DaaS provider in JSON format:

```
{
                    providerName: "ondemand",
                    providerType: "SEARCH",
                    businessEntity: "Organization",
                    systemName: "SFA",
                    requiredFields: [
                                "displayName"
                    ]
}
```

DaaS Search

The DaaS Search REST API uses some input fields of a business entity to call an external DaaS service and transforms the response into a list of records.

The API uses the POST method.

Request URL

The DaaS Search URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/daas/search/<daas provider>
```

Make the following HTTP POST request to the DaaS Search URL:

```
POST http://<host>:<port>/<context>/<database ID>/daas/search/<daas provider>
```

Note: In the request body, provide the details of the business entity with the required field.

Request Body

The request body must contain an XML element or a JSON element of the type `Organization` or `OrganizationView` from the `urn:co-ors.informatica.mdm` namespace.

Sample API Request

The following example is a request to the DaaS provider `ondemand` to search for the `organization` business entity with the display name `INFA`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/search/ondemand
{
  "Organization": {
    "displayname": "INFA"
  }
}
```

Sample API Response

The following sample response shows the results of a search that the DaaS provider returns for the organization business entity with the display name `INFA`:

```
{
  "link": [],
  "item": [
    {
      "businessEntity": {
        "Organization": {
          "displayName": "INFA LAB INC",
          "dunsNumber": "001352574",
          "TelephoneNumbers": {
            "link": [],
            "item": [
              {
                "phoneNum": "9736252265"
              }
            ]
          },
          "Addresses": {
            "link": [],
            "item": [
              {
                "Address": {
                  "cityName": "ROCKAWAY",
                  "addressLine1": "11 WALL ST",
                  "postalCd": "07866",
                  "stateCd": {
                    "stateAbbreviation": "NJ"
                  }
                }
              }
            ]
          }
        }
      },
```

```
  "label": "001352574-INFA LAB INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "007431013",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "5629019971"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "LONG BEACH",
              "addressLine1": "3569 GARDENIA AVE",
              "postalCd": "90807",
              "stateCd": {
                "stateAbbreviation": "CA"
              }
            }
          }
        ]
      }
    }
  },
  "label": "007431013-INFA INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "020591086",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "7186248777"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "BROOKLYN",
              "addressLine1": "16 COURT ST STE 2004",
              "postalCd": "11241",
              "stateCd": {
                "stateAbbreviation": "NY"
              }
            }
          }
        ]
      }
    }
  },
  "label": "020591086-INFA INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
```

```
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "604057286",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "8473580802"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "PALATINE",
              "addressLine1": "THE HARRIS BANK BLDG STE 614,800E NW HWY",
              "postalCd": "60074",
              "stateCd": {
                "stateAbbreviation": "IL"
              }
            }
          }
        ]
      }
    }
  },
  "label": "604057286-INFA INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "032785606",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "5629019971"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "SIMI VALLEY",
              "addressLine1": "3962 HEMWAY CT",
              "postalCd": "93063",
              "stateCd": {
                "stateAbbreviation": "CA"
              }
            }
          }
        ]
      }
    }
  },
  "label": "032785606-INFA INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA",
      "dunsNumber": "045228877",
      "TelephoneNumbers": {
        "link": [],
```

```
        "item": [
          {
            "phoneNum": "3304410033"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "NORTON",
              "addressLine1": "4725 ROCK CUT RD",
              "postalCd": "44203",
              "stateCd": {
                "stateAbbreviation": "OH"
              }
            }
          }
        ]
      }
    }
  },
  "label": "045228877-INFA",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "609028209",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "9084394655"
          }
        ]
      },
      "Addresses": {
        "link": [],
        "item": [
          {
            "Address": {
              "cityName": "CALIFON",
              "addressLine1": "21 FAIRMOUNT RD W",
              "postalCd": "07830",
              "stateCd": {
                "stateAbbreviation": "NJ"
              }
            }
          }
        ]
      }
    }
  },
  "label": "609028209-INFA INC",
  "systemName": "SFA"
},
{
  "businessEntity": {
    "Organization": {
      "displayName": "INFA INC",
      "dunsNumber": "195271796",
      "TelephoneNumbers": {
        "link": [],
        "item": [
          {
            "phoneNum": "7137824181"
          }
        ]
```

```
            },
            "Addresses": {
              "link": [],
              "item": [
                {
                  "Address": {
                    "cityName": "HOUSTON",
                    "addressLine1": "1800 AUGUSTA DR STE 200",
                    "postalCd": "77057",
                    "stateCd": {
                      "stateAbbreviation": "TX"
                    }
                  }
                }
              ]
            }
          }
        },
        "label": "195271796-INFA INC",
        "systemName": "SFA"
      },
      {
        "businessEntity": {
          "Organization": {
            "displayName": "INFA INC",
            "dunsNumber": "098605830",
            "Addresses": {
              "link": [],
              "item": [
                {
                  "Address": {
                    "cityName": "BELLFLOWER",
                    "postalCd": "90707",
                    "stateCd": {
                      "stateAbbreviation": "CA"
                    }
                  }
                }
              ]
            }
          }
        },
        "label": "098605830-INFA INC",
        "systemName": "SFA"
      }
    ]
}
```


DaaS Read

The DaaS Read REST API uses some fields from a business entity to request an external DaaS service and transforms the response into a full record.

The API uses the POST method. Specify the required field in the request to the DaaS provider.

Request URL

The DaaS Read URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/daas/read/<daas provider>
```

Make the following POST request to the URL:

```
POST http://<host>:<port>/<context>/<database ID>/daas/read/<daas provider>
```

Note: In the request body, provide the details of the record with the required field.

Query Parameter

The following table lists the query parameter that you can use:

Parameter	Description
logChanges	Optional. If set to true, the resulting record includes the (Service Data Object) SDO change summary which is passed to the Write Business Entity service. Default is false.

Request Body

The request body must contain an XML element or a JSON element of the type `OrganizationView` from the `urn:coors.informatica.mdm` namespace.

Sample API Request

The following example is a request to the DaaS provider `ondemand` to search for the `organization` business entity with the display name `INFA`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/search/ondemand
{
  "Organization": {
    "displayname": "INFA"
  }
}
```

Sample API Response

The following sample response shows the details that the DaaS provider returned for the organization whose D-U-N-S number is `001352574`:

```
{
  "Organization": {
    "displayName": "Infa Lab Inc",
    "dunsNumber": "001352574",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "phoneNum": "09736250550"
        }
      ]
    },
    "Addresses": {
      "link": [],
      "item": [
        {
          "Address": {
            "cityName": "Rockaway",
            "addressLine1": "11 WALL ST"
          }
        }
      ]
    }
  }
}
```

WriteMerge

The WriteMerge REST API accepts a list of records retrieved from a DaaS provider, persists them in separate XREFs with the appropriate source system, and merges them into a single record. All XREFs belong to the same record.

The API uses the POST method.

Request URL

The WriteMerge URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/daas/write-merge/<business entity name>
```

Make the following HTTP POST request to the WriteMerge URL:

```
POST http://<host>:<port>/<context>/<database ID>/daas/write-merge/<business entity name>
```

Request Body

The request body should contain an XML or JSON element of the type `DaaSEntity.Pager` from the `urn:cobase.informatica.mdm` namespace.

Response Header

When the response is successful, the API returns the interactionId and the processId in the response header.

Sample API Request

The following sample request uses the result of a DaaS search as input to create a record of the Organization business entity type:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/write-merge/Organization
{
  "item": [
    {
      "businessEntity": {
        "Organization": {
          "displayName": "INFA LAB INC",
          "dunsNumber": "001352574",
          "TelephoneNumbers": {
            "item": [
              {
                "phoneNum": "9736252265"
              }
            ]
          }
        }
      },
      "systemName": "DNB"
    },
    {
      "businessEntity": {
        "Organization": {
          "displayName": "INFA INC",
          "dunsNumber": "007431013",
          "TelephoneNumbers": {
            "item": [
              {
                "phoneNum": "5629019971"
              }
             ]
          }
        }
      },
```

```
        "systemName": "Admin"
      }
    ]
}
```

Sample API Response

The following sample response shows the response header and body after successfully merging the list of records into a single record:

```
{
  "Organization": {
    "key": {
      "rowid": "121921",
      "sourceKey": "140000000000"
    },
    "rowidObject": "121921",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "key": {
            "rowid": "21721",
            "sourceKey": "140000001000"
          },
          "rowidObject": "21721"
        }
      ]
    }
  }
}
```

DaaS Import

The DaaS Import REST API accepts data in an XML format and converts it into a record.

The API uses the POST method.

Request URL

The Import DaaS Data URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/daas/import/
FamilyTreeMemberOrganizationToOrgView
```

Make the following HTTP POST request to the Import DaaS Data URL:

```
POST http://<host>:<port>/<context>/<database ID>/daas/import/
FamilyTreeMemberOrganizationToOrgView
```

Query Parameters

The name of the source system is a required parameter.

The following table lists the parameters that you can use in the request:

Parameter	Description
systemName	Required. Name of the source system which performs the data change.
interactionId	Optional. Interaction ID to assign to all the changes. Usually, the Hub generates the ID when it creates a pending change as the result of a call.

Parameter	Description
effectivePeriod	Optional. Contains the start date and end date. Specifies the period of time for which the record is effective. Provide these parameters for a timeline-enabled record.
validateOnly	Optional. If set to TRUE, only validation rules are applied to the modified record and the changes are not persisted.
recordState	Optional. Hub state for the created or the updated records. The supported record states are ACTIVE and PENDING.
processId	Optional. ID of the workflow process that contains the task. When a workflow is started as result of the service call, the parameter contains the identifier of the started workflow process.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Request Body

The request body must contain an XML or a JSON element of the type `DaaSChangeFamilyTreeMemberOrganizationToOrgView` from the `urn:co-ors.informatica.mdm` namespace.

Response Header

When the response is successful, the API returns the interactionId and the processId in the response header and the record details in the response body.

If the process generates an interaction ID and uses it to create the record, the API returns the interaction ID. If the process starts a workflow instead of directly saving the record to the database, the API returns the process ID which is the ID of the workflow process.

The following example shows a response header with an interaction ID and a process ID:

```
BES-interactionId: 72200000242000
BES-processId: 15948
```

Sample API Request

The request is always in the XML format.

The following sample request shows XML data of the type `ChildAssociation` from the linkage namespace:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/import/linkage2org?
systemName=Admin
<FamilyTreeMemberOrganization xmlns="http://services.dnb.com/LinkageServiceV2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ChildAssociation">
    <AssociationTypeText>ParentSubsidiary</AssociationTypeText>
    <OrganizationName>
        <OrganizationPrimaryName>
            <OrganizationName>INFORMATICA JAPAN K.K.</OrganizationName>
        </OrganizationPrimaryName>
    </OrganizationName>
    <SubjectHeader>
        <DUNSNumber>697557825</DUNSNumber>
    </SubjectHeader>
    <Location>
        <PrimaryAddress>
            <StreetAddressLine>
```

```
                    <LineText>2-5-1, ATAGO</LineText>
                </StreetAddressLine>
                <StreetAddressLine>
                    <LineText>ATAGO GREEN HILLS MORI TOWER 26F.</LineText>
                </StreetAddressLine>
                <PrimaryTownName>MINATO-KU</PrimaryTownName>
                <CountryISOAlpha2Code>JP</CountryISOAlpha2Code>
                <TerritoryAbbreviatedName>TKY</TerritoryAbbreviatedName>
                <PostalCode>105-0002</PostalCode>
                <TerritoryOfficialName>TOKYO</TerritoryOfficialName>
            </PrimaryAddress>
        </Location>
        <OrganizationDetail>
            <FamilyTreeMemberRole>
                <FamilyTreeMemberRoleText>Subsidiary</FamilyTreeMemberRoleText>
            </FamilyTreeMemberRole>
            <FamilyTreeMemberRole>
                <FamilyTreeMemberRoleText>Headquarters</FamilyTreeMemberRoleText>
            </FamilyTreeMemberRole>
            <StandaloneOrganizationIndicator>false</StandaloneOrganizationIndicator>
        </OrganizationDetail>
        <Linkage>
            <LinkageSummary>
                <ChildrenSummary>
                    <ChildrenQuantity>1</ChildrenQuantity>
                    <DirectChildrenIndicator>false</DirectChildrenIndicator>
                </ChildrenSummary>
                <ChildrenSummary>
                    <ChildrenTypeText>Affiliate</ChildrenTypeText>
                    <ChildrenQuantity>29</ChildrenQuantity>
                    <DirectChildrenIndicator>false</DirectChildrenIndicator>
                </ChildrenSummary>
                <ChildrenSummary>
                    <ChildrenTypeText>Branch</ChildrenTypeText>
                    <ChildrenQuantity>1</ChildrenQuantity>
                    <DirectChildrenIndicator>true</DirectChildrenIndicator>
                </ChildrenSummary>
                <SiblingCount>29</SiblingCount>
            </LinkageSummary>
            <GlobalUltimateOrganization>
                <DUNSNumber>825320344</DUNSNumber>
            </GlobalUltimateOrganization>
            <DomesticUltimateOrganization>
                <DUNSNumber>697557825</DUNSNumber>
            </DomesticUltimateOrganization>
            <ParentOrganization>
                <DUNSNumber>825320344</DUNSNumber>
            </ParentOrganization>
            <FamilyTreeMemberOrganization>
                <AssociationTypeText>HeadquartersBranch</AssociationTypeText>
                <OrganizationName>
                    <OrganizationPrimaryName>
                        <OrganizationName>INFORMATICA JAPAN K.K.</OrganizationName>
                    </OrganizationPrimaryName>
                </OrganizationName>
                <SubjectHeader>
                    <DUNSNumber>692640710</DUNSNumber>
                    <SubjectHandling>
                        <SubjectHandlingText DNBCodeValue="11028">De-listed</
SubjectHandlingText>
                    </SubjectHandling>
                </SubjectHeader>
                <Location>
                    <PrimaryAddress>
                        <StreetAddressLine>
                            <LineText>2-2-2, UMEDA, KITA-KU</LineText>
                        </StreetAddressLine>
                        <PrimaryTownName>OSAKA</PrimaryTownName>
                        <CountryISOAlpha2Code>JP</CountryISOAlpha2Code>
                        <TerritoryAbbreviatedName>OSK</TerritoryAbbreviatedName>
                        <PostalCode>530-0001</PostalCode>
```

```
                    <TerritoryOfficialName>OSAKA</TerritoryOfficialName>
                </PrimaryAddress>
            </Location>
            <OrganizationDetail>
                <FamilyTreeMemberRole>
                    <FamilyTreeMemberRoleText>Branch</FamilyTreeMemberRoleText>
                </FamilyTreeMemberRole>
                <StandaloneOrganizationIndicator>false</StandaloneOrganizationIndicator>
            </OrganizationDetail>
            <Linkage>
                <GlobalUltimateOrganization>
                    <DUNSNumber>825320344</DUNSNumber>
                </GlobalUltimateOrganization>
                <DomesticUltimateOrganization>
                    <DUNSNumber>697557825</DUNSNumber>
                </DomesticUltimateOrganization>
                <HeadquartersOrganization>
                    <DUNSNumber>697557825</DUNSNumber>
                </HeadquartersOrganization>
                <FamilyTreeHierarchyLevel>2</FamilyTreeHierarchyLevel>
            </Linkage>
        </FamilyTreeMemberOrganization>
        <FamilyTreeHierarchyLevel>2</FamilyTreeHierarchyLevel>
    </Linkage>
</FamilyTreeMemberOrganization>
```

Sample API Response

The following sample response shows the response header and body after successfully importing and creating an Organization business entity:

```
BES-interactionId: 72202100242034
BES-processId: 156048
{
  "Organization": {
    "key": {
      "rowid": "101921",
      "sourceKey": "697557825"
    },
    "rowidObject": "101921"
  }
}
```

DaaS Update

The DaaS Update REST API accepts the data in an XML format before and after a change. The API applies the changes to the record.

The API uses the POST method.

Request URL

The DaaS Update URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/daas/update/
FamilyTreeMemberOrganizationToOrgView
```

Make the following HTTP POST request to the DaaS Update URL:

```
POST http://<host>:<port>/<context>/<database ID>/daas/update/
FamilyTreeMemberOrganizationToOrgView
```

Query Parameters

The name of the source system is a required parameter.

The following table lists the parameters that you can use in the request:

Parameter	Description
systemName	Required. Name of the source system which performs the data change.
interactionId	Optional. Interaction ID to assign to all the changes. Usually, the Hub generates the ID when it creates a pending change as the result of a call.
effectivePeriod	Optional. Contains the start date and end date. Specifies the period of time for which the record is effective. Provide these parameters for a timeline-enabled record.
validateOnly	Optional. If set to TRUE, only validation rules are applied to the modified record and the changes are not persisted.
recordState	Optional. Hub state for the created or the updated records. The supported record states are ACTIVE and PENDING.
processId	Optional. ID of the workflow process that contains the task. When workflow is started as result of the service call, the parameter contains the identifier of the started workflow process.

Related Topics:

- "Formats for Dates and Time in UTC" on page 27

Request Body

The request body must contain an XML or a JSON element of the type `DaaSChangeFamilyTreeMemberOrganizationToOrgView` from the `urn:co-ors.informatica.mdm` namespace.

Response Header

When the response is successful, the API returns the interactionId and the processId in the response header and the record details in the response body.

If the process generates an interaction ID and uses it to create the record, the API returns the interaction ID. If the process starts a workflow instead of directly saving the record to the database, the API returns the process ID which is the ID of the workflow process.

The following example shows a response header with an interaction ID and a process ID:

```
BES-interactionId: 72200000242000
BES-processId: 15948
```

Sample API Request

The API accepts two responses in XML format, one response is before a change and the other response is after the change. In the following request, a new phone number is added to the organization. The `before` XML data does not have the phone number, the `after` XML data has the phone number.

The following request contains the newly added phone number:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/update/linkage2org?
systemName=Admin
```

```
<urn:DaaSChangelinkage2org xmlns:urn="urn:cs-ors.informatica.mdm" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:type="urn:DaaSChangelinkage2org">
    <urn:before xmlns="http://services.dnb.com/LinkageServiceV2.0">
        <SubjectHeader>
            <DUNSNumber>697557825</DUNSNumber>
        </SubjectHeader>
    </urn:before>


    <urn:after xmlns="http://services.dnb.com/LinkageServiceV2.0">
        <SubjectHeader>
            <DUNSNumber>697557825</DUNSNumber>
        </SubjectHeader>
        <Telecommunication>
            <TelephoneNumber>
                <TelecommunicationNumber>09736250550</TelecommunicationNumber>
                <InternationalDialingCode>1</InternationalDialingCode>
                <UnreachableIndicator>true</UnreachableIndicator>
            </TelephoneNumber>
        </Telecommunication>
    </urn:after>
</urn:DaaSChangelinkage2org>
```

Sample API Response

The following example shows the rowId of the newly created telephone number of the organization:

```
{
  "Organization": {
    "key": {
      "rowid": "101921",
      "sourceKey": "697557825"
    },
    "rowidObject": "101921",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "key": {
            "rowid": "1722",
            "sourceKey": "09736250550"
          },
          "rowidObject": "1722"
        }
      ]
    }
  }
}
```

CHAPTER 4

Simple Object Access Protocol Business Entity Service Calls

This chapter includes the following topics:

- Simple Object Access Protocol Calls for Business Entity Services, 140
- Authentication method, 141
- Authentication Cookies for Login from Third-Party Applications, 141
- Web Services Description Language File, 142
- SOAP URL, 143
- SOAP Requests and Responses, 144
- Viewing Input and Output Parameters, 145
- SOAP API Reference, 146
- Sample SOAP Request and Response, 147

Simple Object Access Protocol Calls for Business Entity Services

Simple Object Access Protocol (SOAP) endpoint calls make all business entity services available as web services. You can make SOAP calls to create, delete, update, and search for records in a business entity. You can perform operations, such as merge, unmerge, and match records. You can also make SOAP calls to create, update, and search for tasks and perform tasks.

The SOAP endpoints for business entity services use the Web Services Security (WS-Security) UsernameToken to authenticate users.

Note: Before you use the SOAP APIs to call the business entity services, validate the Operational Reference Store.

Authentication method

All SOAP calls to the business entity services require user authentication. Provide the user name and password in the SOAP message header of the web service request.

The SOAP header element Security contains the security-related data. The Security element contains the UsernameToken element which has the following child elements:

username

User name associated with the token.

password

Password for the user name associated with the token.

Send the user name and password in the UsernameToken element.

The following example shows the Security header element in the SOAP message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:cs-ors.informatica.mdm">
   <soapenv:Header>
   <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <UsernameToken>
      <Username>admin</Username>
      <Password>admin</Password>
    </UsernameToken>
  </Security>
  </soapenv:Header>
<soapenv:Body>
.....
</soapenv:Body>
</soapenv:Envelope>
```

Authentication Cookies for Login from Third-Party Applications

Use authentication cookies to authenticate the MDM Hub users and call the business entity services from third-party applications. You can obtain a cookie based on the credentials of an authenticated user. Save the cookie and use it to call the SOAP APIs. You need not hard-code the user name and password.

Make the following POST request to log in to the Entity 360 View with your user name and password:

```
POST http://<host>:<port>/e360/com.informatica.tools.mdm.web.auth/login
{
    user: 'admin',
    password: 'user password'
}
```

When the login operation is successful, the server returns the authentication cookie in the `set-cookie` header field. The following sample code shows a `set-cookie` in the response header:

```
Set-Cookie: auth_hash_cookie="admin===QTc1RkNGQkNCMzc1RjIyOQ==";
Version=1; Path=/
```

Store the hash and use it in the request header of your API calls. You need not provide a user name and a password for the API calls.

The following example shows how to use an authentication cookie in your API request header:

```
GET http://<IP of host>/cmx/cs/localhost-orcl-DS_UI1
Cookie: auth_hash_cookie="admin===QTc1RkNGQkNCMzc1RjIyOQ=="
```

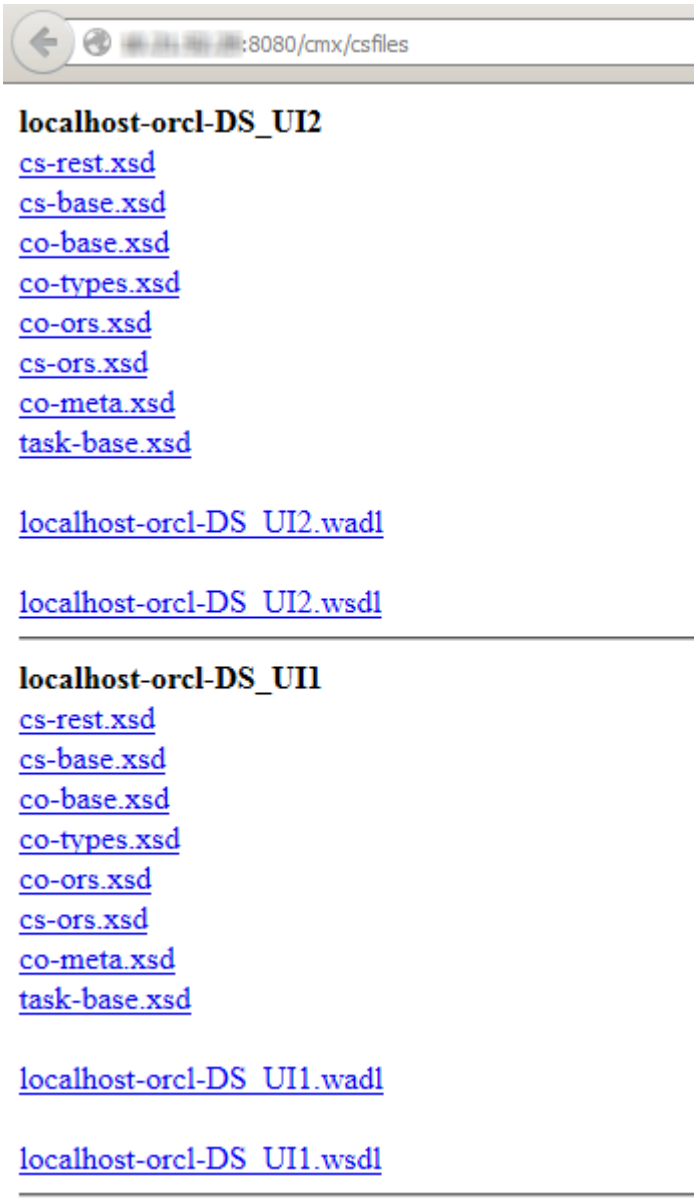
Web Services Description Language File

Web Services Description Language (WSDL) files contain the XML descriptions of the web services, formats of the SOAP requests and responses, and all parameters. The MDM Hub generates a WSDL file for each Operational Reference Store.

The WSDL files for each Operational Reference Store are in the following location:

```
http://<host>:<port>/cmx/csfiles
```

The following image shows the location where you can download the WSDL file for the Operational Reference Stores:



Click the link to download the WSDL file for the DS_UI1 or DS_UI2 Operational Reference Store.

SOAP URL

A SOAP URL is the address you use to connect to the SOAP server.

A SOAP URL has the following syntax:

```
http://<host>:<port>/<context>/<database ID>
```

The URL has the following fields:

host

The host that runs the database.

port

Port number that the database listener uses.

context

The context is always `cmx/services/BEServices`.

database ID

ID of the ORS as registered in the Databases tool in the Hub Console.

The following example shows a SOAP URL:

```
http://localhost:8080/cmx/services/BEServices/localhost-orcl-DS_UI1
```

SOAP Requests and Responses

Use the SOAP XML message format to send requests through a SOAP client to the business entity service and to receive responses from the business entity service to the client. The SOAP request and response format is the same.

A SOAP message contains the following elements:

Envelope

Defines the start and the end of the message.

Header

Optional. Contains additional attributes, such as authentication details for processing the message. If the Header element is present, it must be the first child element of the Envelope element.

Body

Contains the XML data that the client or the web service processes.

A SOAP message has the following format:

```
<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Header>
</env:Header>

<env:Body>
</env:Body>

</env:Envelope>
```

A SOAP request has the following format:

```
POST /<host>:<port>/<context>/<database ID> HTTP/1.0
Content-Type: text/xml; charset=utf-8

<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Header>
</env:Header>

<env:Body>
</env:Body>
```

```
</env:Envelope>
```

A SOAP response has the following format:

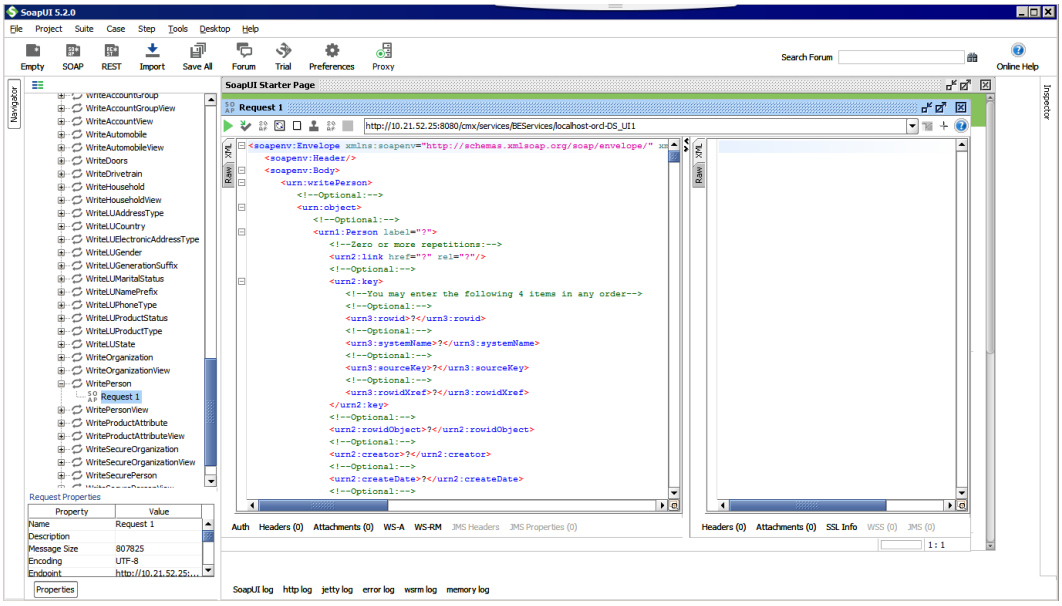
```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Header>
</env:Header>

<env:Body>
</env:Body>

</env:Envelope>
```

Viewing Input and Output Parameters

You can use a functional testing tool, such as SoapUI, to view SOAP API input and output parameters.

Create a SOAP project and import the WSDL file into the project. The operations you can perform using the business entity services appear as nodes in SoapUI. Each operation has a request and a response message format. SoapUI creates a sample request for each operation when you import the WSDL file.

Open the project and double-click a request to open the request editor. The following image shows the input parameters in SoapUI for the WritePerson SOAP API:



SOAP API Reference

The SOAP API reference for business entity services lists the SOAP APIs and provides a description for each API. Also refer to the WSDL file for descriptions of the business entity services.

Use the following SOAP APIs to perform operations on business entities:

Get Metadata

Returns the data structure of a business entity.

List Record

Returns the list of lookup values or foreign key values.

Read Record

Returns the details of a root record in the business entity.

Create Record

Creates a record in the specified business entity.

Update Record

Updates the specified root record and its child records.

Delete Record

Deletes a root record in a business entity.

Search Record

Searches a string value in a searchable root business entity and returns the root records that match the search criteria.

Preview Promote

Returns a preview of a resulting record if you promote pending changes based on the interaction ID of the change request.

Promote

Promotes all pending changes made to a record based on the interaction ID of the change request.

Delete Promote

Deletes all pending changes that you make to a record based on the interaction ID of the change request.

Preview Merge

Returns a preview of a consolidated root record if you merge two or more root records.

Merge Records

Merges two or more root records to create a single consolidated record.

Unmerge Records

Unmerges a root record into individual root records that existed before the records were merged.

Get Related Records

Returns a list of related records based on the relationships that you configure in the Hierarchy Manager.

Read Matched Records

Returns records that match a specified root record.

Update Matched Records

Creates or updates a record in the match table.

Delete Matched Records

Deletes matched records from the match table.

Get BPM Metadata

Returns the task types and two indicators that specify whether the workflow adapter can perform the Get Task Lineage service and the administration services.

List Tasks

Returns a list of workflow tasks.

Read Task

Returns the details of a task.

Create Task

Creates a task and starts a workflow.

Update Task

Updates a single task.

Execute Task Action

Performs a task action and sets the task back to the workflow for further processing.

List Assignable Users

Returns a list of users to whom you can assign the tasks that belong to a task type.

Task Complete

Closes a task workflow after you complete all the tasks in the workflow.

Sample SOAP Request and Response

The following sample SOAP request retrieves a list of assignable users:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:cs-ors.informatica.mdm">
   <soapenv:Header>
   <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <UsernameToken>
      <Username>admin</Username>
      <Password>admin</Password>
    </UsernameToken>
  </Security>
  </soapenv:Header>
   <soapenv:Body>
      <urn:listAssignableUsers>
         <!--Optional:-->
         <urn:parameters>
            <!--Optional:-->
            <urn:taskType>Update</urn:taskType>
            <!--Optional:-->
            <urn:businessEntity>Person</urn:businessEntity>
         </urn:parameters>
      </urn:listAssignableUsers>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample SOAP response lists the assignable users:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <soapenv:Body>
      <ns6:listAssignableUsersReturn xmlns:ns1="urn:cs-base.informatica.mdm"
xmlns:ns2="urn:co-base.informatica.mdm" xmlns:ns3="urn:co-ors.informatica.mdm"
xmlns:ns4="urn:co-meta.informatica.mdm" xmlns:ns5="urn:task-base.informatica.mdm"
xmlns:ns6="urn:cs-ors.informatica.mdm">
         <ns6:object>
            <ns1:users>
                                       <user>
                                       <userName>admin</userName>
                                       </user>
                                                  </users>
            <ns1:roles/>
         </ns6:object>
      </ns6:listAssignableUsersReturn>
   </soapenv:Body>
</soapenv:Envelope>
```

CHAPTER 5

Services for Cross-reference Records and BVT Calculations

This chapter includes the following topics:

- Overview of Services for Cross-reference Records and BVT Calculations, 149
- Getting Cross-reference Data and Investigating BVT Calculations, 149
- Filtering and Paginating Responses, 153
- Establish the Best Version of the Truth, 154

Overview of Services for Cross-reference Records and BVT Calculations

You can use the services for cross-reference records and best version of the truth (BVT) calculations to learn how the source data forms the master record.

You can use these services to perform the following tasks:

- gather information about the source data
- determine how the best version of the truth was determined
- override the BVT calculations to ensure the master records contain the best version of the truth

Getting Cross-reference Data and Investigating BVT Calculations

Master records in the MDM Hub maintain the best version of the truth (BVT). The MDM Hub consolidates the most trustworthy data from several source systems into each master record to achieve the best version of the truth. The MDM Hub stores source data in cross-reference records. You can use business entity services to read data from the cross-reference records and determine how the BVT was calculated.

Get Cross-reference Records

You can use a business entity service to get the cross-reference records for a particular master record.

The REST API URL to get cross-reference records has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
contentMetadata=XREF
```

The following sample request retrieves cross-reference records for a Person business entity record with a row ID of 123:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=XREF
```

.

Get Cross-reference Records Response

The following example shows the cross-reference records that are returned for the Person record with a row ID of 123:

```
GET /Person/123?contentMetadata=XREF

{
  "firstName": "Joe",
  "lastName": "Smith",
  "XREF": {
    "item": [
      {
        "rowidXref": 111,
        "firstName": "Joe",
        "lastName": "Smith",
      },
      {
        "rowidXref": 222,
        "firstName": "John",
        "lastName": "Smith"
      }
    ]
  }
}
```

Determine Contributors to the Master Record

You can use business entity services to see which cross-reference record field contributes to the master record. The contributing record to each field is identified by the row ID of the cross-reference record.

The REST API URL to determine contributors to the master record has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
contentMetadata=BVT
```

The following sample request retrieves BVT information for a Person record with a row ID of 123:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=BVT
```

Determine Contributors to the Master Record Response

The following example shows which cross-reference record contributed to each field in the master record:

```
{
  "firstName": "Joe",
  "lastName": "Smith",
  "BVT": {
    "firstName": {
      "rowidXref": 111
    },
```

```
    "lastName": {
      "rowidXref": 222
    }
  },
}
```

Get the Trust Scores of Contributing Cross-reference Record Fields

You can use business entity services to get the trust scores of cross-reference record fields that contribute to the master record.

The REST API URL to determine contributors and get the trust scores has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
contentMetadata=TRUST
```

The following sample request provides trust scores for a Person record with a row ID of 123:

```
GET http://localhost:8080/cmx/cs/ors/Person/123?contentMetadata=TRUST
```

Get the Trust Scores of Contributing Cross-reference Record Fields Response

The following response example provides trust scores for each field in a Person business entity record:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "TRUST": {
    "firstName": {
      "score": 75.0,
      "valid": true,
      "trustSetting" :{
          // custom settings
      }
    },
  },
}
```

Getting the Trust Scores of All Cross-reference Record Fields

Use the REST API with the contentMetadata parameter set to XREF_TRUST to get the trust scores and downgrade percentages of all cross-reference record fields.

The request URL for the Read REST API to determine contributors and get the trust scores:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
contentMetadata=XREF_TRUST
```

The following sample request retrieves cross-reference data for a Person record with a row ID of 123:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=XREF_TRUST
```

Get the Trust Scores of All Cross-reference Record Fields Response

The following example shows the trust scores and downgrade percentages of all cross-reference record fields for a Person business entity:

```
{
    "firstName": "Sergey",
    "lastName": "Ivanov",
    "XREF": {
        "item": [{
            "rowidXref": 111,
            "firstName": "Sergey",
            "lastName": "Petrov",
            "TRUST": {
```

```
                "firstName": {
                    "score": 75.0,
                    "valid": true
                },
                "lastName": {
                    "score": 60.0,
                    "valid": false,
                    "downgradePerCent": 20.0
                }
            }
        }, {
            "rowidXref": 222,
            "firstName": "Sergey",
            "lastName": "Ivanov",
            "TRUST": {
                "firstName": {
                    "score": 10.0,
                    "valid": true
                },
                "lastName": {
                    "score": 80.0,
                    "valid": true
                }
            }
        }]
    }
}
```

Get Information about Source Systems

You can get information about which source systems the cross-reference data comes from, and how many cross-reference records the source systems contribute for the whole record, for each node, or for each record.

The following parameters can be specified in the request:

describe

Set to `true` to return the description of the source system. Can be `true` or `false`. Default is `false`.

aggregate

Defines for which level to return source system information. Can be `ENTITY`, `NODE`, and `RECORD`. Default is `ENTITY`.

recordStates

Specifies the record state for which to return records. Can be `ACTIVE`, `PENDING`, or `DELETED`. Default is `ACTIVE`.

compact

When set to `no`, specifies that the entity level data is returned when the aggregate parameter is specified with `ENTITY` and other aggregate levels. Can be `yes` or `no`. For REST API requests only. Default is `yes`.

Get Information about Source Systems Example

The following sample request gets entity-level and node-level source system information for the Person business entity with a row ID of 123:

```
GET http://localhost:8080/cmx/cs/ors/Person/123?
action=getSourceSystems&aggregate=ENTITY,NODE&compact=no
```

The following sample request gets record-level source system information and the source system descriptions for the Person business entity with a row ID of 456:

```
GET http://localhost:8080/cmx/cs/ors/Person/123/Address/456?
action=getSourceSystems&aggregate=ENTITY,NODE&compact=no
```

Get Information about Source Systems Response

The following example shows entity-level and node-level information for a Person business entity:

```
{
      "name": "Admin",
      "xrefCount": 120
    }
  ],
  Person: {
    "rowidObject": "456",
    "sourceSystem":
      {
        "name": "Admin",
        "xrefCount": 30
      }
    ]
  }
}
```

Filtering and Paginating Responses

You can select the fields to return in the response, filter results by several criteria, and paginate results.

Filtering Request Examples

The following table shows sample requests for the Person Business entity with a variety of filters applied and a description of the results that are returned in the response:

Request	Description of returned results
/Person/123	All user-defined fields
/Person/123?readSystemFields=true	All user-defined fields and all system fields
/Person/123?fields=firstName	One user-defined field
/Person/123?fields=updatedBy	One system field
Person/123?fields=firstName,updatedBy	One user-defined field and one system field
/Person/123?fields=firstName&readsystemFields=true	One user-defined field and all system fields

Establish the Best Version of the Truth

After a data steward investigates the source data in the cross-reference records, they can then make adjustments to how source data is consolidated to ensure that the master record represents the best version of the truth.

You can use business entity services to perform the following actions to establish the best version of the truth:

- Update the trust settings
- Remove mismatched source data
- Select the correct contributing field
- Write the correct value to the master record

Select the Correct Contributing Field

If the field with the highest trust score does not contain the best version of the truth, a data steward can select the field that does contain the correct data to contribute the data to the master record.

The URL and request body to select the correct contributing field based on the system name and source key has the following format:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
systemName=<source system name>
{
    BVT: {
        <field name>: {
            systemName: "<source system name>",
            sourceKey: "<source key>"
        }
    }
}
```

The URL and request body to select the correct contributing field based on the cross-reference record ID has the following format:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
systemName=<source system name>
{
    BVT: {
        <field name>: {
            rowidXref: "<row ID>"
        }
    }
}
```

Select the Correct Contributing Field Example

The following URL and request body selects the first name field of the cross-reference record from the Sales source system with a source key of 0001 to contribute to the master record:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=Admin
{
    BVT: {
        firstName: {
            systemName: "Sales",
            sourceKey: "0001"
        }
    }
}
```

The following URL and request body selects the first name field of the cross-reference record with a row ID of 789 to contribute to the master record:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=Admin
{
    BVT: {
        firstName: {
            rowidXref: "789"
        }
    }
}
```

Write the Correct Value to the Master Record

When you use a business entity service call to write a correct value to a master record, you also can establish the trust settings for the value. If you do not specify trust settings, the MDM Hub uses the administrator system settings.

The URL and request body to write the correct value with the administrator trust setting has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
systemName=<source system providing the correct value>{
    "<field name>": "<correct value>",
     "$original": {
        "<field name>": "<current value>",

    },
    "TRUST": {
        "<field name>": {
            "trustSetting" : {
                custom: false
            }
        }
    }
}
```

The URL and request body to write the correct value with defined trust settings has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
systemName=<source system providing the correct value>{
    "<field name>": "<correct value>",
     "$original": {
        "<field name>": "<current value>",

    },
    "TRUST": {
        "firstName": {
            "trustSetting" : {
                custom: true, // if custom=true, all other trustSetting fields
                              //are mandatory. If they are not set,
                              //the service will return an error.
                minimumTrust: <minimum trust percent>,
                maximumTrust: <maximum trust percent>,
                timeUnit: "<units for measuring trust decay>",
                maximumTimeUnits: <number of units>,
                graphType: "<name of graph type>"
            }
        }
    }
}
```

Trust Parameters

You can define the following trust parameters:

minimumTrust

Trust level that a data value has when it is old (after the decay period has elapsed). This value must be less than or equal to the maximum trust.

Note: If the maximum and minimum trust are equal, then the decay curve is a flat line, and the decay period and decay type have no effect.

maximumTrust

Trust level that a data value has if it has just been changed. For example, if source system X changes a phone number field from 555-1234 to 555-4321, the new value gets system X's maximum trust level for the phone number field. By setting the maximum trust level relatively high, you can ensure that changes in the source systems are applied to the base object.

timeUnit

Specifies the units used in calculating the decay period—day, week, month, quarter, or year.

maximumTimeUnits

Specifies the number — of days, weeks, months, quarters, or years — used in calculating the decay period.

graphType

Decay follows a pattern in which the trust level decreases during the decay period. The graph types can be one of the following decay patterns:

Graph Type Parameter	Description
LINEAR	Simplest decay. Decay follows a straight line from the maximum trust to the minimum trust.
RISL	Most of the decrease occurs toward the beginning of the decay period. Decay follows a concave curve. If a source system has this graph type, then a new value from the system will probably be trusted, but this value might be overridden.
SIRL	Most of the decrease occurs toward the end of the decay period. Decay follows a convex curve. If a source system has this graph type, it will be relatively unlikely for any other system to override the value in the master record until the value is near the end of its decay period.

Write the Correct Value to the Master Record Example

Example 1

You want to change the name in the master record from Sam Brown to John Smith. The change is attributed to the Sales source system. The trust settings are set to the administrator trust settings.

The following code shows the URL and command for Example 1.

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=Sales
{
    "firstName": "John",
    "lastName": "Smith"
    "$original": {
        "firstName": "Sam",
        "lastName": "Brown"
    },
```

```
    "TRUST": {
        "firstName": {
            "trustSetting" : {
                custom: false
            }
            {
        "lastName": {
            "trustSetting" : {
                custom: false
            }
        }
    }
}
```

Example 2

You want to change the name in the master record from Sam Brown to John Smith. The change is attributed to the SFA source system. The trust settings are set to a minimum trust of 60 and a maximum trust of 90, and the trust decays linearly over a decay period of three months.

The following code shows the URL and command for Example 2.

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=SFA
{
    "firstName": "John",
    "lastName": "Smith"
    "$original": {
        "firstName": "Sam",
        "lastName": "Brown"
    },
    "TRUST": {
        "firstName": {
            "trustSetting" : {
                custom: true,
                minimumTrust: 60,
                maximumTrust: 90,
                timeUnit: "Month",
                maximumTimeUnits: 3,
                graphType: "LINEAR"
            }
            {
        "lastName": {
            "trustSetting" : {
                custom: true,
                minimumTrust: 60,
                maximumTrust: 90,
                timeUnit: "Month",
                maximumTimeUnits: 3,
                graphType: "LINEAR"
            }
        }
    }
}
```

Remove Mismatched Source Data

If a cross-reference record is incorrectly associated with a particular master record, a data steward can unmerge the cross-reference record. A new master record is created from the unmerged cross-reference record.

Only one cross-reference record can be unmerged in an unmerge call. If several cross-reference records need to be unmerged, make an unmerge call for each cross-reference record.

If a trigger is configured for an unmerge event, then an unmerge task is created. Otherwise, the cross-reference record is unmerged.

The URL and command to unmerge a record based on the system name and source key has the following format:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
action=unmerge&systemName=<source system name>
{
name: "<object name>",
key: {rowid: "<rowid value>", sourcekey: "<source key>", systemName: "<source system
name>" }
}
```

The URL and command to unmerge a record based on the cross-reference record ID has the following format:

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?
action=unmerge&systemName=<source system name>
{
name: "<object name>",
key: {rowid: "<rowid value>", rowidXref: "<row ID of xref>"}
}
```

Remove Mismatched Source Data Example

REST API Example

The following code shows the URL and command to unmerge the cross-reference record at the child level from an Address record:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/Person/181921?
action=unmerge&systemName=Admin
{
"name":"Person.Address",
"key":{
"rowid":"41721 ",
"rowidXref":41722
}
}
```

Where:

- the cross-reference record to unmerge has a row ID of 41722
- the row ID of the master record to unmerge the cross-reference record from is 41721
- the row ID of the root record is 181921

SOAP/EJB Example

The following code shows the URL and command to unmerge the cross-reference record at the child level from an Address record:

```
<ns9:UnMerge xmlns:ns2="urn:co-base.informatica.mdm" xmlns:ns7="urn:co-
meta.informatica.mdm" xmlns:ns3="http://services.dnb.com/LinkageServiceV2.0"
xmlns:ns8="urn:task-base.informatica.mdm" xmlns:ns6="urn:co-ors.informatica.mdm"
xmlns:ns1="urn:cs-base.informatica.mdm" xmlns:ns9="urn:cs-ors.informatica.mdm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns9:UnMerge">
   <ns9:parameters>
      <ns9:businessEntityKey name="Person">
         <ns1:key>
            <ns1:rowid>181921</ns1:rowid>
        </ns1:key>
      </ns9:businessEntityKey>
      <ns9:unmergeKey name="Person.TelephoneNumbers">
         <ns1:key>
            <ns1:rowid>41721 </ns1:rowid>
            <ns1:rowidXref>41722</ns1:rowidXref>
         </ns1:key>
      </ns9:unmergeKey>
      <ns9:treeUnmerge>true</ns9:treeUnmerge>
   </ns9:parameters>
</ns9:UnMerge>
```

Where:

- the cross-reference record to unmerge has a row ID of 41722
- the row ID of the master record to unmerge the cross-reference record from is 41721
- the row ID of the root record is 181921

Unmerge Response

The unmerge response contains the row ID of the base object that is created from the unmerged cross-reference record.

Response example 1

The following example shows the response when you a cross-reference record from a Person root node.

```
{
    Person: {
        rowidObject: "7777"
    }
}
```

Response example 2

The following example shows the response when you a cross-reference record from an Address child node.

```
{
    Person: {
        Address: {
            item: [
                rowidObject: "55555"
            ]
        }
    }
}
```

CHAPTER 6

Supporting Corporate Linkage Service

This chapter includes the following topics:

- Overview, 160
- Business Entity Services for DaaS Import and Update, 160
- Configuring Linkage Support, 161
- Custom Application for Linkage Data Splitting, 161

Overview

The corporate linkage service from Duns & Bradstreet (D&B) returns the parent of a requested organization and all its related entities. You can use the linkage service from D&B through which you can get information of all branches and divisions of an organization. You can create and update records with the data from the linkage service.

You can import the corporate linkage data into the MDM Hub. You must develop a custom application that can use the linkage service from the DaaS provider custom component in an Entity View.

You require a business entity service to import the data from the D&B service and create a record with the data. When there is a change in the data in the external storage, you must be able to make the corresponding changes in your record. D&B provides a monitoring service that notifies you of changes in data. You require a service that can accept data before and after a change, and apply the changes to the corresponding record.

Business Entity Services for DaaS Import and Update

The DaaS Import business entity service accepts data from the linkage service in XML format and converts it into a record. The DaaS Update business entity service accepts data from the external service in the form of

two XML files. The two XML files correspond to data before and after a change. The Update service applies the changes to the corresponding record.

Related Topics:

- "DaaS Import" on page 134
- "DaaS Update" on page 137

Configuring Linkage Support

To use the linkage service from D&B to create and update records, you must add configuration in the Provisioning tool and create a custom application to split the response from the linkage service.

Perform the following tasks to configure support for the linkage service of D&B:

1. Use the Provisioning tool to upload the WSDL for the linkage service.
2. Use the Provisioning tool to configure an XML document to business entity transformation and expose it as a service. When you expose the transformation as a service, the process creates the DaaS Import and Update business entity services.
3. Create a custom application that can request data from the linkage service and split the response into record details and linkage details.
4. Develop a user interface that sends the request to the custom application.

Note: For more information about how to upload the WSDL and configure an XML to business entity transformation, see the Integrating Data as a Service chapter in the *Multidomain MDM Provisioning Tool Guide*.

Custom Application for Linkage Data Splitting

To use the linkage services from D&B, you must design a custom application that can split the linkage information into record details and linkage details.

The custom application must perform the following functions:

1. Accept request for a linkage service from the Entity View.
2. Send the request to D&B and receive the response.
3. Convert the response to XML.
4. Split the response into record details and linkage details.
5. Send the XML information to business entity services to save as a record in the database.
6. Monitor changes to the data and call the List Change Notice function of the external service.

CHAPTER 7

External Calls to Cleanse, Analyze, and Transform Data

This chapter includes the following topics:

- Overview, 162
- Supported Events, 163
- How to Configure External Calls, 163
- Example: Custom Validation and Logic for Business Entity Services, 164

Overview

External providers provide web services to cleanse, analyze, and transform record data. Use the external web services for custom validation, such as checking if the address field is empty when you add a record. Use the external web services for custom logic to transform record data. For example, when you merge two records, you can merge addresses, but not telephone numbers.

An external web service exposes one or more operations that the business entity services can call. Each operation has a request and response type. Business entity services send the record data with the required service parameters to external services. You can configure calls to the external web services for certain steps in the execution logic. Based on the logic that you implement, requests go from Data Director to update the record data. The external services might modify the data, if required.

In the Provisioning tool, configure the business entity and the events for which you want to call the external service. In the Provisioning tool, upload the WSDL file for the external service and register the SOAP service and operation. Bind the service to specific business entities and events.

Use the WSDL file in the Resource Kit to understand the service, operations, methods, and the data types that the service methods exchange. The `custom-logic-service.wsdl` file for the external web services is in the following Resource Kit location: `C:\<infamdm installation directory>\hub\resourcekit\samples\BESExternalCall\source\resources\webapp\WEB-INF\wsdl\`

The Resource Kit includes sample code that implements custom logic and validation. When you install the Resource Kit, the `bes-external-call.ear` file for the sample custom logic and validation is deployed on the application server.

Supported Events

A business entity service consists of service steps. You can apply custom logic and validation to any of the steps.

You can make external calls for the following events:

- WriteCO.BeforeEverything
- WriteCO.BeforeValidate
- WriteCO.AfterValidate
- WriteCO.AfterEverything
- WriteView.BeforeEverything
- WriteView.BeforeValidate
- WriteView.AfterValidate
- WriteView.AfterEverything
- MergeCO.BeforeEverything
- MergeCO.AfterEverything
- PreviewMergeCO.BeforeEverything
- PreviewMergeCO.AfterEverything
- ReadCO.BeforeEverything
- ReadCO.AfterEverything
- ReadView.BeforeEverything
- ReadView.AfterEverythingEvents

How to Configure External Calls

A business entity service has service steps. An incoming request passes through each service step. You can configure calls to external services for certain steps in the business entity service execution logic.

Perform the following steps to configure the external calls:

1. Build and deploy the `bes-external-call.ear` file.
2. In the Provisioning tool, perform the following tasks:
    a. Upload the WSDL file for the external service.
    b. Register the web service as a SOAP service.
    c. Configure an external call.

For more information about uploading the WSDL file, registering the SOAP service, and configuring external calls, see the *Multidomain MDM Provisioning Tool Guide*.

For more information about building and deploying the EAR file, see the *Multidomain MDM Resource Kit Guide*.

Example: Custom Validation and Logic for Business Entity Services

You can test the custom validation and logic when you add and merge the Person records. The custom validation checks if the Person record has an address. The custom logic does not allow you to merge two telephone numbers. Use REST APIs to create and merge the Person records.

1. To check the validation when you create a Person record, perform the following steps:
   a. Use the Create API to create a Person record without an address. You get a validation error.
   b. Use the Create API to create a Person record with an address. The operation is successful.
2. To check the custom logic when you merge records, perform the following steps:
   a. Use the Create API to create two Person records with addresses and telephone numbers.
   b. Use the Preview Merge API to merge the two Person records. Add `overrides` to the preview merge request to merge the addresses and the telephones. The response shows a single address, but two telephone numbers. The custom logic prevents the merging of telephone numbers.

Prerequisites

To check the custom logic and validation, you must upload the WSDL file in the Provisioning tool. You must register the SOAP service and operation. Bind the service to the business entities and events for which you want to use custom logic and validation. You can test the logic and validation for the specified business entities and events.

Step 1. Test Custom Validation

Use the Create API to create the following Person record without an address:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_Sample/Person?systemName=Admin
{
        firstName: "John"
    }
```

You get a validation error.

Use the Create API to create the following Person record with an address:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_Sample/Person?systemName=Admin
{
        firstName: "John",
        Addresses: {
            item: [
                {
                    cityName: "Toronto"
                }
            ]
        }
    }
```

The request creates a Person record.

Step 2. Test Custom Logic

Perform the following steps to test the custom logic:

1. Use the Create API to create two Person records with addresses and telephones:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_sample/Person?systemName=Admin
    {
        firstName: "John",
        Addresses: {
            item: [
                {
                    cityName: "Toronto"
                }
            ]
        },
        TelephoneNumbers: {
            item:[
                {
                    phoneNum: "111-11-11"
                }
            ]
        }
    }
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_sample/Person?systemName=Admin
    {
        firstName: "John",
        Addresses: {
            item: [
                {
                    cityName: "Ottawa"
                }
            ]
        },
        TelephoneNumbers: {
            item:[
                {
                    phoneNum: "222-22-22"
                }
            ]
        }
    }
```

   The response contains the following rowIds:

   - Person: 161923, 161924
   - Addresses: 2123, 2124
   - TelephoneNumbers: 101723, 101724

2. Run the PreviewMerge API to merge both the Person records:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_sample/Person/161923?
action=previewMerge&depth=2
{
        keys: [
            {
                rowid: "161924"
            }
        ]
    }
```

   The response is a Person record with two Addresses and two Telephone numbers.

```
{
  "Person": {
    "rowidObject": "161923        ",
    "creator": "admin",
    "createDate": "2016-10-20T09:50:35.878-04:00",
    "updatedBy": "admin",
    "lastUpdateDate": "2016-10-20T09:50:35.879-04:00",
```

```
 "consolidationInd": 4,
 "lastRowidSystem": "SYS0          ",
 "hubStateInd": 1,
 "label": "Person: , Bill",
 "partyType": "Person",
 "displayName": "Bill",
 "firstName": "Bill",
"TelephoneNumbers": {
   "link": [],
   "firstRecord": 1,
   "recordCount": 2,
   "pageSize": 2,
   "item": [
     {
       "rowidObject": "101723        ",
       "creator": "admin",
       "createDate": "2016-10-20T09:50:35.904-04:00",
       "updatedBy": "admin",
       "lastUpdateDate": "2016-10-20T09:50:35.905-04:00",
       "consolidationInd": 4,
       "lastRowidSystem": "SYS0          ",
       "hubStateInd": 1,
       "label": "PhoneNumbers",
       "phoneNum": "111-1111     ",
       "phoneCountryCd": "1"
     },
     {
       "rowidObject": "101724        ",
       "creator": "admin",
       "createDate": "2016-10-20T09:50:54.768-04:00",
       "updatedBy": "admin",
       "lastUpdateDate": "2016-10-20T09:50:54.769-04:00",
       "consolidationInd": 4,
       "lastRowidSystem": "SYS0          ",
       "hubStateInd": 1,
       "label": "PhoneNumbers",
       "phoneNum": "222-2222     ",
       "phoneCountryCd": "1"
     }
   ]
 },
 "Addresses": {
   "link": [],
   "firstRecord": 1,
   "recordCount": 2,
   "pageSize": 2,
   "item": [
     {
       "rowidObject": "2123          ",
       "creator": "admin",
       "createDate": "2016-10-20T09:50:37.956-04:00",
       "updatedBy": "admin",
       "lastUpdateDate": "2016-10-20T09:50:37.956-04:00",
       "consolidationInd": 4,
       "lastRowidSystem": "SYS0          ",
       "hubStateInd": 1,
       "label": "Addresses",
       "Address": {
         "rowidObject": "2121          ",
         "creator": "admin",
         "createDate": "2016-10-20T09:50:36.922-04:00",
         "updatedBy": "admin",
         "lastUpdateDate": "2016-10-20T09:50:37.923-04:00",
         "consolidationInd": 4,
         "lastRowidSystem": "SYS0          ",
         "hubStateInd": 1,
         "label": "Address",
         "cityName": "Toronto"
       }
     },
```

```
        {
          "rowidObject": "2124          ",
          "creator": "admin",
          "createDate": "2016-10-20T09:50:54.790-04:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2016-10-20T09:50:54.790-04:00",
          "consolidationInd": 4,
          "lastRowidSystem": "SYS0          ",
          "hubStateInd": 1,
          "label": "Addresses",
          "Address": {
            "rowidObject": "2122          ",
            "creator": "admin",
            "createDate": "2016-10-20T09:50:54.777-04:00",
            "updatedBy": "admin",
            "lastUpdateDate": "2016-10-20T09:50:54.777-04:00",
            "consolidationInd": 4,
            "lastRowidSystem": "SYS0          ",
            "hubStateInd": 1,
            "label": "Address",
            "cityName": "Ottawa"
          }
        }
      ]
    }
  }
}
```

3. Run the PreviewMerge API to merge both the Person records, with overrides to merge the addresses and telephone numbers:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/Person/161923?
action=previewMerge&depth=3
{
    keys: [
        { rowid: "161923" }
    ],
    overrides: {
        Person: {
            Addresses: {
                item:[
                    {
                        rowidObject: "2123",
                        MERGE: {
                            item:[{key:{rowid: "2124"}}],
                            $original: {
                                item:[null]
                            }
                        }
                    }
                ]
            },
            TelephoneNumbers: {
                item:[
                    {
                        rowidObject: "101723",
                        MERGE: {
                            item:[{key:{rowid: "101724"}}],
                            $original: {
                                item:[null]
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

The response shows a single address, but two telephone numbers:


```
{
  "Person": {
    "rowidObject": "161923        ",
    "creator": "admin",
    "createDate": "2016-10-20T09:50:35.878-04:00",
    "updatedBy": "admin",
    "lastUpdateDate": "2016-10-20T09:50:35.879-04:00",
    "consolidationInd": 4,
    "lastRowidSystem": "SYS0          ",
    "hubStateInd": 1,
    "label": "Person: , Bill",
    "partyType": "Person",
    "displayName": "Bill",
    "firstName": "Bill",
   "TelephoneNumbers": {
      "link": [],
      "firstRecord": 1,
      "recordCount": 2,
      "pageSize": 2,
      "item": [
        {
          "rowidObject": "101723        ",
          "creator": "admin",
          "createDate": "2016-10-20T09:50:35.904-04:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2016-10-20T09:50:35.905-04:00",
          "consolidationInd": 4,
          "lastRowidSystem": "SYS0          ",
          "hubStateInd": 1,
          "label": "PhoneNumbers",
          "phoneNum": "111-1111     ",
          "phoneCountryCd": "1"
        },
        {
          "rowidObject": "101724        ",
          "creator": "admin",
          "createDate": "2016-10-20T09:50:54.768-04:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2016-10-20T09:50:54.769-04:00",
          "consolidationInd": 4,
          "lastRowidSystem": "SYS0          ",
          "hubStateInd": 1,
          "label": "PhoneNumbers",
          "phoneNum": "222-2222     ",
          "phoneCountryCd": "1"
        }
      ]
    },
     "Addresses": {
      "link": [],
      "firstRecord": 1,
      "recordCount": 1,
      "pageSize": 1,
      "item": [
        {
          "rowidObject": "2123          ",
          "creator": "admin",
          "createDate": "2016-10-20T09:50:37.956-04:00",
          "updatedBy": "admin",
          "lastUpdateDate": "2016-10-20T09:50:37.956-04:00",
          "consolidationInd": 4,
          "lastRowidSystem": "SYS0          ",
          "hubStateInd": 1,
          "label": "Addresses"
        }
      ]
    },
    "PersonDetails": {
```

```
      "link": [],
      "recordCount": 0,
      "pageSize": 0,
      "item": []
    }
  }
}
```

APPENDIX A

Using REST APIs to Add Records

This appendix includes the following topics:

- Using REST APIs to Add Records Overview, 170
- Person Business Entity Structure, 171
- Step 1. Get Information about the Schema, 171
- Step 2. Create a Record, 177
- Step 3. Read the Record, 179

Using REST APIs to Add Records Overview

After you create a business entity model and configure the business entity structure, you can use the REST APIs to add the records.

The following sections use the example of the Person business entity to illustrate how you can add records by using REST APIs. The Person business entity contains data for the employees in your company.

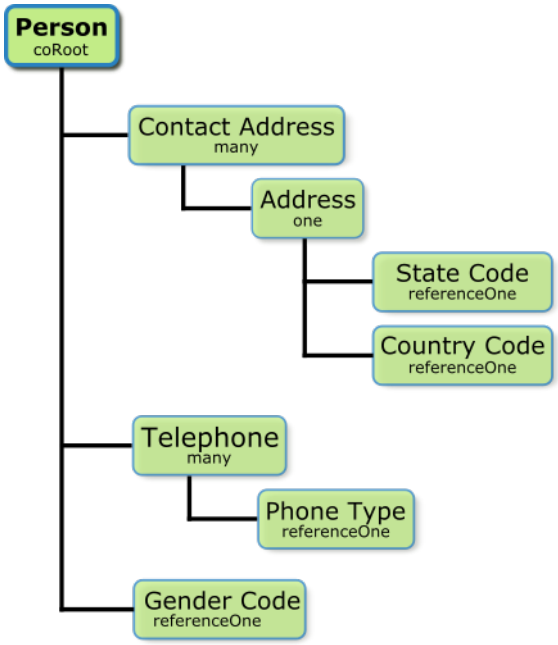
Use the following APIs to add the details of your employees:

1. Get information about the schema. Use the Get Metadata REST API to get information about the data structure of a business entity, including the structure, the list of fields, the field types, and the details of the lookup fields. Or, you can access the XML Schema Definition (XSD) files which describe what elements and attributes you can use. The XSD files are in the `http://<host>:<port>/cmx/csfiles` location.
2. Create a record. Use the Create Record REST API to create the record.
3. Read the data from the record that you have added. Use the Read Record REST API to retrieve the data from the record.

Person Business Entity Structure

We will add a Person record by using REST APIs. The Person root node is the uppermost node in the Person business entity structure. Under the Person root node, there are nodes for the employee details such as gender, address, and phone.

The following image shows the structure of the Person business entity:



Person is the root node of the Person business entity. The node type, listed below the node name, indicates the relationship between the parent node and the child node. There is a one-to-one relationship between the Contact Address and the Address, which indicates that each contact address can only have one address associated with it. There is a one-to-many relationship between the Person and the Telephone, which indicates that a Person record can have many telephone number records associated with it. There is a one-to-one relationship between the Person and the Gender, which indicates that a person record can have only one gender value. The gender values reside in a lookup table. Similarly, the state code and country code values reside in lookup tables.

Step 1. Get Information about the Schema

Use the Get Metadata REST API to get information about a schema. The Get Metadata API returns the data structure of a business entity. The metadata lists the business entity fields, field types, and details of the lookup fields.

The Get Metadata URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?action=meta
```

The following sample request retrieves metadata information for the Person business entity:

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?action=meta
```

Get Metadata Response

The following example shows some excerpts of the data structure of the Person business entity:


```
{
    "object": {
        "field": [
            {
                "allowedValues": [
                    "Person"
                ],
                "name": "partyType",
                "label": "Party Type",
                "dataType": "String",
                "length": 255,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "imageUrl",
                "label": "Image URL",
                "dataType": "ImageURL",
                "length": 255,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "statusCd",
                "label": "Status Cd",
                "dataType": "String",
                "length": 2,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "displayName",
                "label": "Display Name",
                "dataType": "String",
                "length": 200,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "birthdate",
                "label": "Birthdate",
                "dataType": "Date",
                "length": 0,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "firstName",
                "label": "First Name",
                "dataType": "String",
                "length": 50,
                "totalDigits": 0,
```

```
    "fractionDigits": 0,
    "readOnly": false,
    "required": false,
    "system": false
},
{
    "name": "lastName",
    "label": "Last Name",
    "dataType": "String",
    "length": 50,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": false,
    "required": false,
    "system": false
},
{
    "name": "middleName",
    "label": "Middle Name",
    "dataType": "String",
    "length": 50,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": false,
    "required": false,
    "system": false
},
{
    "name": "dirtyIndicator",
    "label": "Dirty Indicator",
    "dataType": "Integer",
    "length": 38,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": true,
    "required": false,
    "system": true
},
{
    "name": "hubStateInd",
    "label": "Hub State Ind",
    "dataType": "Integer",
    "length": 38,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": true,
    "required": false,
    "system": true
},
{
    "name": "cmDirtyInd",
    "label": "Content metadata dirty Ind",
    "dataType": "Integer",
    "length": 38,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": true,
    "required": false,
    "system": true
},
{
    "name": "lastRowidSystem",
    "label": "Last Rowid System",
    "dataType": "String",
    "length": 14,
    "totalDigits": 0,
    "fractionDigits": 0,
    "readOnly": true,
    "required": false,
    "system": true
},
```

```
--------------------------------------------------------------------------------------------

            {
                "name": "genderCd",
                "label": "Gender Cd",
                "dataType": "lookup",
                "readOnly": false,
                "required": false,
                "system": false,
                "lookup": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/LUGender?action=list&idlabel=genderCode%3AgenderDisp",
                            "rel": "lookup"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/LUGender?action=list",
                            "rel": "list"
                        }
                    ],
                    "object": "LUGender",
                    "key": "genderCode",
                    "value": "genderDisp"
                }
            }
        ],

--------------------------------------------------------------------------------------------
                "child": [
                    {
                        "field": [
                            {
                                "name": "cityName",
                                "label": "City Name",
                                "dataType": "String",
                                "length": 100,
                                "totalDigits": 0,
                                "fractionDigits": 0,
                                "readOnly": false,
                                "required": false,
                                "system": false
                            },
                            {
                                "name": "addressLine2",
                                "label": "Address Line2",
                                "dataType": "String",
                                "length": 100,
                                "totalDigits": 0,
                                "fractionDigits": 0,
                                "readOnly": false,
                                "required": false,
                                "system": false
                            },
                            {
                                "name": "addressLine1",
                                "label": "Address Line1",
                                "dataType": "String",
                                "length": 100,
                                "totalDigits": 0,
                                "fractionDigits": 0,
                                "readOnly": false,
                                "required": false,
                                "system": false
                            },
                            {
                                "name": "isValidInd",
                                "label": "Is Valid Ind",
```

```
                "dataType": "String",
                "length": 1,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },
            {
                "name": "postalCd",
                "label": "Postal Cd",
                "dataType": "String",
                "length": 10,
                "totalDigits": 0,
                "fractionDigits": 0,
                "readOnly": false,
                "required": false,
                "system": false
            },

-------------------------------------------------------------------------------

            {
                "name": "countryCode",
                "label": "Country Code",
                "dataType": "lookup",
                "readOnly": false,
                "required": false,
                "system": false,
                "dependents": [
                    "Person.Address.Address.stateCd"
                ],
                "lookup": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/
localhost-hub101-ds_ui1/LUCountry?action=list",
                            "rel": "list"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/
localhost-hub101-ds_ui1/LUCountry?action=list&idlabel=countryCode%3AcountryNameDisp",
                            "rel": "lookup"
                        }
                    ],
                    "object": "LUCountry",
                    "key": "countryCode",
                    "value": "countryNameDisp"
                }
            },
            {
                "name": "stateCd",
                "label": "State Cd",
                "dataType": "lookup",
                "readOnly": false,
                "required": false,
                "system": false,
                "parents": [
                    "Person.Address.Address.countryCode"
                ],
                "lookup": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/
localhost-hub101-ds_ui1/LUCountry/{Person.Address.Address.countryCode}/LUState?
action=list",
                            "rel": "list"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/
```

```
localhost-hub101-ds_ui1/LUCountry/{Person.Address.Address.countryCode}/LUState?
action=list&idlabel=stateAbbreviation%3AstateNameDisp",
                                        "rel": "lookup"
                                    }
                                ],
                                "object": "LUCountry.LUState",
                                "key": "stateAbbreviation",
                                "value": "stateNameDisp"
                            }
                        }
                    ],
                    "name": "Address",
                    "label": "Address",
                    "many": false
                }
            ],
            "name": "Address",
            "label": "Contact Address",
            "many": true
        },
        {
            "field": [
                {
                    "name": "phoneNum",
                    "label": "Phone Number",
                    "dataType": "String",
                    "length": 13,
                    "totalDigits": 0,
                    "fractionDigits": 0,
                    "readOnly": false,
                    "required": false,
                    "system": false
                },
```

```
                {
                    "name": "phoneTypeCd",
                    "label": "Phone Type",
                    "dataType": "lookup",
                    "readOnly": false,
                    "required": false,
                    "system": false,
                    "lookup": {
                        "link": [
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-
hub101-ds_ui1/LUPhoneType?action=list&idlabel=phoneType%3AphoneTypeDisp",
                                "rel": "lookup"
                            },
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-
hub101-ds_ui1/LUPhoneType?action=list",
                                "rel": "list"
                            }
                        ],
                        "object": "LUPhoneType",
                        "key": "phoneType",
                        "value": "phoneTypeDisp"
                    }
                }
            ],
            "name": "Telephone",
            "label": "Telephone",
            "many": true
        }
    ],
    "name": "Person",
    "label": "Person",
    "many": false
```

```
    }
}
```

Step 2. Create a Record

Use the Create Record REST API to create a record. The name of the business entity and the name of the source system are required parameters. Send data for the record in the request body.

The Create Record URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>?systemName=<name of the source system>
```

The `systemName` parameter is a required parameter and specifies the name of the source system.

The Person business entity has the Person root node, and the second level address, gender, and phone nodes.

The following sample request creates a Person record:

```
POST http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person?systemName=Admin
{
    "firstName": "Boris",
    "lastName": "Isaac",
    "genderCd": {
        "genderCode": "M"
    },
    "Address": {
        "item": [
            {
                "Address": {
                    "addressLine1": "B-203, 101 Avenue, New York",
                    "stateCd": {
                        "stateAbbreviation": "NY"
                    },
                    "countryCode": {
                        "countryCode": "US"
                    }
                }
            }
        ]
    },
    "Telephone": {
        "item": [
            {
                "phoneNum": "1234567",
                "phoneTypeCd": {
                    "phoneType": "HOM"
                }
            },
            {
                "phoneNum": "7654321",
                "phoneTypeCd": {
                    "phoneType": "MOB"
                }
            }
        ]
    }
}
```

The request body specifies the following details of the Person record:

- First name.
- Last name.

- Gender.
- Address with the state code and the country code.
- Phone numbers along with the phone type, such as home phone and mobile phone.

Create Record Response

The following sample response shows the response after successfully creating a Person record:


```
{
    "Person": {
        "key": {
            "rowid": "658248",
            "sourceKey": "66240000025000"
        },
        "rowidObject": "658248",
        "genderCd": {
            "key": {
                "rowid": "2             "
            },
            "rowidObject": "2              "
        },
        "Address": {
            "link": [],
            "item": [
                {
                    "key": {
                        "rowid": "101526",
                        "sourceKey": "66240000028000"
                    },
                    "rowidObject": "101526",
                    "Address": {
                        "key": {
                            "rowid": "121506",
                            "sourceKey": "66240000027000"
                        },
                        "rowidObject": "121506",
                        "countryCode": {
                            "key": {
                                "rowid": "233           "
                            },
                            "rowidObject": "233            "
                        },
                        "stateCd": {
                            "key": {
                                "rowid": "52            "
                            },
                            "rowidObject": "52             "
                        }
                    }
                }
            ]
        },
        "Telephone": {
            "link": [],
            "item": [
                {
                    "key": {
                        "rowid": "20967",
                        "sourceKey": "66240000029000"
                    },
                    "rowidObject": "20967",
                    "phoneTypeCd": {
                        "key": {
                            "rowid": "8             "
                        },
                        "rowidObject": "8              "
```

```
                    }
                },
                {
                    "key": {
                        "rowid": "20968",
                        "sourceKey": "66240000030000"
                    },
                    "rowidObject": "20968",
                    "phoneTypeCd": {
                        "key": {
                            "rowid": "6             "
                        },
                        "rowidObject": "6              "
                    }
                }
            ]
        }
    }
}
```


Note: The response body contains the record with the generated rowIds.

If you configure a workflow process to start when you create a record, the following things happen:

- Record is created in a pending state.
- Workflow process is started.
- Workflow process ID is returned in the response header.

If you do not configure a workflow process, then by default, the record is created as Active.

The API returns an interaction ID in the response header if you process the request using an interaction ID.

Step 3. Read the Record

Use the Read Record REST API to retrieve the details of a root record that you added. You can use the API to retrieve the details of the child records of a root record.

The Read Record URL has the following format:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root record>
```

Use the `depth` parameter to specify the number of child levels to return. Specify 2 to return the root node and its direct children, and 3 to return the root node, direct children, and grandchildren. Use the following URL to return the details of the child records:

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the record>?
depth=n
```

The following sample request returns the details of the root node, the direct children, and the grandchildren:

```
GET http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248?depth=3
```

The request returns the details of an active record.

Note: If a workflow is started when you create a record, the record created is in pending state. By default, the Read Record request reads active records. Use the `recordStates` parameter to specify the pending state of the record.

The following sample request reads the details of a pending record:

```
GET http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248?
depth=3&recordStates=PENDING
```

Read the Record Response

The following sample response shows the details of the record that you added:


```
{
    "link": [
        {
            "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248",
            "rel": "self"
        },
        {
            "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248?
depth=2",
            "rel": "children"
        }
    ],
    "rowidObject": "658248        ",
    "label": "Person",
    "partyType": "Person",
    "displayName": "BORIS ISAAC",
    "firstName": "BORIS",
    "lastName": "ISAAC",
    "genderCd": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248/genderCd/2",
                "rel": "self"
            },
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248",
                "rel": "parent"
            },
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248/genderCd/2?depth=2",
                "rel": "children"
            }
        ],
        "rowidObject": "2             ",
        "label": "LU Gender",
        "genderCode": "M",
        "genderDisp": "MALE"
    },
    "Address": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248/Address",
                "rel": "self"
            },
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248",
                "rel": "parent"
            }
        ],
        "firstRecord": 1,
        "pageSize": 10,
        "searchToken": "SVR1.PCWJ",
        "item": [
            {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Address",
                        "rel": "parent"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
```

```
Person/658248/Address/101526?depth=2",
                        "rel": "children"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Address/101526",
                        "rel": "self"
                    }
                ],
                "rowidObject": "101526        ",
                "label": "Contact Address",
                "Address": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506?depth=2",
                            "rel": "children"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526",
                            "rel": "parent"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506",
                            "rel": "self"
                        }
                    ],
                    "rowidObject": "121506        ",
                    "label": "Address",
                    "addressLine1": "B-203, 101 Avenue, New York",
                    "countryCode": {
                        "link": [
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506/countryCode",
                                "rel": "self"
                            },
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506",
                                "rel": "parent"
                            },
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506/countryCode?depth=2",
                                "rel": "children"
                            }
                        ],
                        "countryCode": "US"
                    },
                    "stateCd": {
                        "link": [
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506",
                                "rel": "parent"
                            },
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506/stateCd?depth=2",
                                "rel": "children"
                            },
                            {
                                "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Address/101526/Address/121506/stateCd",
                                "rel": "self"
                            }
                        ],
                        "stateAbbreviation": "NY"
```

```
                    }
                }
            }
        ]
    },
    "Telephone": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248",
                "rel": "parent"
            },
            {
                "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/
658248/Telephone",
                "rel": "self"
            }
        ],
        "firstRecord": 1,
        "pageSize": 10,
        "searchToken": "SVR1.PCWK",
        "item": [
            {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Telephone",
                        "rel": "parent"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Telephone/20967",
                        "rel": "self"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Telephone/20967?depth=2",
                        "rel": "children"
                    }
                ],
                "rowidObject": "20967         ",
                "label": "Telephone",
                "phoneNum": "1234567      ",
                "phoneTypeCd": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20967",
                            "rel": "parent"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20967/phoneTypeCd/8",
                            "rel": "self"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20967/phoneTypeCd/8?depth=2",
                            "rel": "children"
                        }
                    ],
                    "rowidObject": "8             ",
                    "label": "LU Phone Type",
                    "phoneTypeDisp": "HOME",
                    "phoneType": "HOM"
                }
            },
            {
                "link": [
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
```

```
Person/658248/Telephone/20968",
                        "rel": "self"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Telephone/20968?depth=2",
                        "rel": "children"
                    },
                    {
                        "href": "http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/
Person/658248/Telephone",
                        "rel": "parent"
                    }
                ],
                "rowidObject": "20968         ",
                "label": "Telephone",
                "phoneNum": "7654321      ",
                "phoneTypeCd": {
                    "link": [
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20968/phoneTypeCd/6",
                            "rel": "self"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20968",
                            "rel": "parent"
                        },
                        {
                            "href": "http://localhost:8080/cmx/cs/localhost-hub101-
ds_ui1/Person/658248/Telephone/20968/phoneTypeCd/6?depth=2",
                            "rel": "children"
                        }
                    ],
                    "rowidObject": "6             ",
                    "label": "LU Phone Type",
                    "phoneTypeDisp": "MOBILE",
                    "phoneType": "MOB"
                }
            }
        ]
    }
}
```

APPENDIX B

Using REST APIs to Upload Files

This appendix includes the following topics:

- Using REST APIs to Upload Files Overview, 184
- REST APIs for Files, 184
- File Components, 185
- Storage Types, 185
- Attaching Files to Records, 186
- Attaching Files to Tasks, 188
- Uploading Resource Bundle Files, 190

Using REST APIs to Upload Files Overview

You can use the REST APIs to upload files to a storage type. After you upload a file, you can attach the file to a record or a task, or use the file to localize the Data Director user interface.

Based on how you want to use the files, the combination of REST APIs, the file components, and the storage type you use might differ. For example, to attach files to records or tasks, you create the metadata for the file and upload the file to a temporary storage. After you upload the file, you can attach the file to a record in a database or attach the file to a task in the BPM storage. To localize the Data Director user interface, you download the ZIP file, modify the zipped files, and then upload the modified ZIP bundle to the bundle storage.

REST APIs for Files

You can use a set of general purpose REST APIs to upload and manage files for attachments or localization.

The following table lists the REST APIs for files:

REST API	Description	Supported Storage Type
List File Metadata	Returns list of file metadata in a storage.	BPM or TEMP
Create File Metadata	Creates metadata for a file in a storage.	DB or TEMP

REST API	Description	Supported Storage Type
Get File Metadata	Returns the metadata for a file.	BPM, BUNDLE, DB, or TEMP
Update File Metadata	Updates the metadata for a file.	DB or TEMP
Upload File Content	Uploads the content for a file to a storage.	BUNDLE, DB, or TEMP
Get File Content	Downloads the content for a file.	BPM, BUNDLE, DB, or TEMP
Delete File	Deletes a file in a storage, including components associated to the file such as the file metadata or content.	BUNDLE, DB, or TEMP

File Components

To attach files to records or tasks, create metadata for the file and then upload the file content. To localize the Data Director user interface, you download the resource bundle file and then upload the modified resource bundle file.

File metadata

Information about the file, such as the file name, file type, and content type. Depending on your storage type, you might need to include other parameters, such as creator, create time, and upload date.

File content

The content for the file. For example, the text, image, document, or resource bundle.

Storage Types

Upload and store files in a supported storage implementation. The storage type that you use depends on whether you want to localize the Data Director user interface or attach files to records or tasks.

The following list describes the supported storage types:

BPM

Stores files attached to tasks together with the task data. When you attach a file to a task, the process stores the file to a BPM storage from the TEMP storage.

Files stored in the BPM storage use the following file ID format: `taskId::filename`.

Note: To attach a file to triggered tasks or existing tasks, in the Provisioning tool, enable attachments for task triggers, task types, and task actions. For more information, see the *Multidomain MDM Provisioning Tool Guide*.

BUNDLE

Stores resource bundle files that localize the Data Director user interface.

Files stored in the BUNDLE storage use the following file ID format: `besMetadata`.

DB

Stores file attachments for records in the C_REPOS_ATTACHMENTS table. When you attach a file to a record, the process stores the file to a DB storage from the TEMP storage.

Files stored in the DB storage use the following file ID format: `DB_<RowID>`.

Note: To attach a file to a record, in the Provisioning tool, configure a field with `FileAttachment` as the data type. For more information about configuring the data type, see the *Multidomain MDM Provisioning Tool Guide*.

TEMP

Temporarily stores files in the C_REPOS_ATTACHMENTS table and marks the files as TEMP. Files are deleted from the TEMP storage after they are successfully uploaded to the BPM or DB storage or after a preconfigured expiration time.

Files stored in the TEMP storage use the following file ID format: `TEMP_<ROWID_ATTACHMENT>`.

For more information about configuring the expiration time, see the *Multidomain MDM Configuration Guide*.

Attaching Files to Records

Before you attach a file to a record, create the metadata of the file and then upload the file to the temporary storage.

1. To create the metadata of a file, use the Create File Metadata REST API with TEMP as the storage type. For example, the following request creates the metadata for the `Document_3.pdf` file:

```
POST http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP
Content-Type: application/json
{
    "fileName": "Document_3.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

   Note: Always create the file metadata in the TEMP storage.

   The Create File Metadata REST API returns an ID for the file. The file ID is in the following format: `<Storage Type>_<RowID>`. Where the RowID refers to the row ID of the file that you upload to the storage.

   In the example, the API call returns the following ID for the `Document_3.pdf` file: `TEMP_SVR1.OJU3`

   You can use the file ID to upload, attach, update, download, and delete the file.

2. To upload the file, use the Upload File Content REST API with TEMP as the storage type.
   For example, the following request uploads the file to the TEMP storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.OJU3/
content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

   Note: After you upload a file, the TEMP storage stores the file for a pre-configured period of 60 minutes. You must attach the file to a record before the pre-configured period expires.

3. To create a record and attach the file to a new record, use the Create Record REST API.

For example, the following request creates a record and attaches the file with the file ID, TEMP_SVR1.OJU3:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/Person?systemName=Admin
Content-Type: application/json
{
   "frstNm":"John",
   "lstNm":"Smith",
   "addrLn1":"2100 Breverly Road",
   "addrTyp":{
      "addrTyp":"Billing",
      "addrTypDesc":"Billing"
   },
   "cntryCd":{
      "cntryCd":"AX",
      "cntryDesc":"Aland"
   },
   "attachments":{
      "item":[
         {
            "fileId":"TEMP_SVR1.OJU3"
         }
      ]
   }
}
```

Note: When you attach a file to a record, the process stores the file to the database. The ID of the file changes to DB_<RowID>, where DB indicates that the file is stored in the database.

4. To replace a file attached to a record, use the Upload File Content REST API with DB as the storage type. For example, the following request replaces the attached file with the file ID, DB_SVR1.OJU3, in the database:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU3/content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

Note: The storage type in the request URL is DB.

5. To edit the file metadata after you attach a file to a record, use the Update File Metadata REST API with DB as the storage type.
For example, the following request updates the file metadata of a file associated with the file ID, DB_SVR1.OJU3, in the DB storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU3
Content-Type: application/json
{
    "fileName": "Document_4.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

6. To download a file attached to a record, use the GET File Content REST API with DB as the storage type. For example, the following request downloads a file associated with the file ID, DB_SVR1.OJU3, from the DB storage:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU3/content
```

7. To delete a file attached to a record, use the Delete File REST API with DB as the storage type.
For example, the following request deletes a file associated with the file ID, DB_SVR1.OJU3, from the DB storage:

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.OJU3
```

Attaching Files to Tasks

Create the metadata for a file and then upload the file content to a temporary storage. After you upload the file, attach the file to a triggered task or an existing task.

Note: To attach a file to triggered tasks or existing tasks, in the Provisioning tool, enable attachments for task triggers, task types, and task actions. For more information, see the *Multidomain MDM Provisioning Tool Guide*.

1. To create the metadata of a file, use the Create File Metadata REST API with TEMP as the storage type. For example, the following request creates the metadata for the `file1.txt` file:

```
POST http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP

{
    "fileName": "file1.txt",
    "fileType": "text",
    "fileContentType": "text/plain"
}
```

   Note: Always create the file metadata in the TEMP storage.

   The Create File Metadata REST API returns an ID for the file. The file ID is in the following format: `<Storage Type>_<RowID>`. Where the RowID refers to the row ID of the file that you upload to the storage.
   In the example, the API call returns the following ID for `file1.txt`: `TEMP_SVR1.1VDVS`

   You can use the file ID to upload, attach, update, and delete the file.

2. To upload the file, use the Upload File Content REST API with TEMP as the storage type.
   For example, the following request uploads the file to the TEMP storage:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS/
content

Test attachment content: file 1
```

   Note: After you upload a file, the TEMP storage stores the file for a pre-configured period of 60 minutes. You must attach the file to a task before the pre-configured period expires.

3. Attach the file to the task that is triggered when you manage records.

   - To attach the file to the task that is triggered when you create a record, use the Create Business Entity REST API with the `taskattachments` parameter.
     For example, the following request creates a record and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person?
systemName=Admin&taskAttachments=TEMP_SVR1.1VDVS
Content-Type: application/json

{
     firstName: "John",
     lastName: "Smith",
     Phone: {
          item: [
               {
                    phoneNumber: "111-11-11"
               }
          ]
     }
}
```

   - To attach the file to the task that is triggered when you update a record, use the Update Business Entity REST API with the `taskattachments` parameter.

For example, the following request updates a record and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
PUT http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?
systemName=Admin&taskAttachments=TEMP_SVR1.1VDVS
{
    rowidObject: "233",
    firstName: "BOB",
    lastName: "LLOYD",
    Phone: {
        item: [
            {
                rowidObject: "164",
                phoneNumber: "777-77-77",
                $original: {
                    phoneNumber: "(336)366-4936"
                }
            }
        ]
    },
    $original: {
        firstName: "DUNN"
    }
}
```

- To attach the file to the task that is triggered when you merge a record, use the Merge Business Entity REST API with the `taskattachments` parameter.
  For example, the following request merges a record and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245?
action=merge&taskAttachments=TEMP_SVR1.1VDVS
Content-Type: application/<json/xml>
{
  keys: [
    {
        rowid: "2478246"
    }
  ],
  overrides: {
    Person: {
       firstName: "Charlie"
    }
  }
}
```

- To attach the file to the task that is triggered when you unmerge a record, use the Unmerge Business Entity REST API with the `taskattachments` parameter.
  For example, the following request unmerges a record and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478248?
action=unmerge&taskAttachments=TEMP_SVR1.1VDVS
{
        rowid: "4880369"
}
```

4. Attach the file to an existing task.
   - To attach the file when you update a task, use the Update Task REST API with `attachments` in the request body.
     For example, the following request updates a task and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
PUT http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/task/urn:b4p2:15934
{
     taskType: {
```

```
            name: "UpdateWithApprovalWorkflow"
        },
        taskId: "urn:b4p2:15934",
        owner: "John",
        title: "Smoke test task - updated",
        comments: "Smoke testing - updated",
        "attachments": [
            {
                "id": "TEMP_SVR1.1VDVS"
            }
        ],
        ...
    }
```

- To attach the file when you execute a task action, use the Execute Task Action REST API with `attachments` in the request body.
  For example, the following request executes a task action and attaches the file with the file ID `TEMP_SVR1.1VDVS`:

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/task/urn:b4p2:15934?
taskAction=Cancel
{
    taskType: {
        name:"UpdateWithApprovalWorkflow",
        taskAction: [{name: "Cancel"}]
    },
    taskId: "urn:b4p2:15934",
    owner: "manager",
    title: "Smoke test task 222",
    comments: "Smoke testing",
    "attachments": [
        {
            "id": "TEMP_SVR1.1VDVS"
        }
    ],
    ...
}
```

After you attach a file to a task, the process moves the file from the TEMP storage and stores the file together with the task data in the BPM storage. The ID of the file changes to `taskId::filename`.

Uploading Resource Bundle Files

To localize the Data Director user interface, download the resource bundle ZIP file, modify the files in the ZIP file, and then upload the modified ZIP file to the bundle storage.

1. To download the resource bundle ZIP file, use the Get File Content REST API with BUNDLE as the storage type.
   For example, the following request downloads the resource bundle ZIP file:

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/
content
```

2. Modify the ZIP file by adding language specific bundle files.
   For example, to localize field names, labels, and table names to Russian, add the `besMetadata_ru.properties` file.
3. To upload the modified resource bundle ZIP file, use the Upload File Content REST API with BUNDLE as the storage type.
   For example, the following request uploads the resource bundle ZIP file:

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/
content
```

```
Content-Type: application/octet-stream
Body: binary stream – zip file with besMetadata bundle
```

INDEX

A

authentication
  basic HTTP 22
  method 22
  using cookies 22, 141

B

business entity service
  ReadBE 10
business entity service steps
  SearchBE 10
  WriteBE 10
business entity services
  DaaS import 161
  DaaS update 161
  EJB endpoint 11
  endpoints 11
  REST API reference 32
  REST endpoint 11
  REST endpoints 21
  SOAP endpoint 12
  SOAP endpoints 140

C

configuring
  external calls 163
corporate linkage
  supporting 160
create record
  response body 49
  response header 49
  URL parameters 48
create Record
  request URL 48
create task
  request URL 79

D

date format
  about 27
delete record
  request URL 53
  URL parameter 54
deleted record
  restore 10

E

effective periods
  WriteBE 10
example
  custom logic 164
  external calls 164
execute task action
  request body 87
external calls
  overview 162

G

get event details
  query parameters 124
get record history events
  query parameters 122

L

linkage data
  custom application 161
  splitting 161
linkage service
  configuration 161
list task
  request URL 72
list tasks
  query parameters 72
  sort parameters 73

P

preface 7

Q

query parameters
  depth 27
  firstRecord 27
  recordsToReturn 27
  returnTotal 27
  searchToken 27

R

read record
  query parameters 43
read task
  request URL 77

ReadBE
business entity service 10
records
adding 170
using REST APIs 170
REST APIs
body 24
create a relationship 110
create file metadata 90
create record 47
create task 79
DaaS import 134
DaaS read 131
DaaS search 126
DaaS update 137
delete a relationship 113
delete file 94
delete matched records 120
delete pending 97
delete record 53
execute task action 86
get BPM metadata 71
get DaaS metadata 126
get event details 124
get file content 94
get file metadata 91
get matched records 118
get metadata 32
get record history events 121
get related records 114
header 24
list assignable users 89
list file metadata 89
list match columns 40
list metadata 35
list record 54
list tasks 72
merge records 105
pending merge 103
preview merge 98
preview promote 95
promote merge 104
promote record 97
read record 41
read relationship 108
read task 77
request body 25
request header 25
search record 56
SearchMatch 66
SearchQuery 62
suggester 61
task complete 84
unmerge records 107
update a relationship 112
update file metadata 92
update matched records 119
update record 50
update task 81
REST body
JSON format 26
REST body *(continued)*
XML format 25
REST methods
DELETE 22
GET 22
PATCH 22
POST 22
PUT 22
supported 22
restore
soft-deleted record 10
root records
identifying 12

S

SearchBE
about 10
SearchMatch
exporting results 70
SearchQuery
exporting results 66
SOAP APIs
authentication 141
request 144
response 144
WSDL 142
supported events
list 163

T

testing external calls
prerequisites 164
time zone
about 27
trust
WriteBE 10

U

update record
response body 52
response header 52
URL parameters 50
UTC
about 27

W

WriteBE
business entity service step 10