Informatica®

Informatica® Multidomain MDM
10.3 HotFix 1

业务实体服务指南

Informatica Multidomain MDM 业务实体服务指南
10.3 HotFix 1
2019 年 3 月

发布日期: 2019-05-28

目录

前言

欢迎使用*《Multidomain MDM 业务实体服务指南》*。本指南介绍如何在 Informatica(R) MDM Hub 中执行业务实体服务调用以对业务实体执行操作。

本指南面向专业技术人员，他们负责配置用于向 MDM Hub 发出业务实体服务调用的自定义用户界面。

Informatica 资源

Informatica 通过 Informatica Network 和其他在线门户为您提供一系列产品资源。使用这些资源，可以充分利用 Informatica 产品和解决方案，并向其他 Informatica 用户和主题专家学习。

Informatica Network

在 Informatica Network 中可以获得许多资源，包括 Informatica 知识库和 Informatica 全球客户支持。要进入 Informatica Network，请访问 https://network.informatica.com。

作为 Informatica Network 成员，您可以选择以下服务：

- 在知识库中搜索产品资源。
- 查看产品可用性信息。
- 创建并检查您的支持案例。
- 查找当地的 Informatica 用户组网络并与您的伙伴进行协作。

Informatica 知识库

使用 Informatica 知识库可查找产品资源，例如操作方法文章、最佳实践、视频教程以及常见问题的答案。

要搜索知识库，请访问 https://search.informatica.com。如果您对知识库有任何疑问、意见或建议，请与 Informatica 知识库团队联系，电子邮件地址为 KB_Feedback@informatica.com。

Informatica 文档

使用 Informatica 文档门户可浏览大量当前与最近产品版本的文档库。要浏览文档门户，请访问 https://docs.informatica.com。

除文档门户之外，Informatica 还在 Informatica 知识库中维护了许多产品的文档。如果在文档门户上找不到您产品或产品版本的文档，可以搜索知识库，网址为 https://search.informatica.com。

如果您对产品文档有任何疑问、意见或建议，请与 Informatica 文档团队联系，电子邮件地址为 infa_documentation@informatica.com。

Informatica 产品可用性矩阵

产品可用性矩阵 (PAM) 指明了产品版本支持的操作系统版本、数据库以及数据源和目标的类型。您可以在以下网址中浏览 Informatica PAM：https://network.informatica.com/community/informatica-network/product-availability-matrices。

Informatica Velocity

Informatica Velocity 是由 Informatica 专业服务根据数百个数据管理项目的实际经验所开发出来的，其中汇集了大量使用技巧和最佳实践。Informatica Velocity 代表了 Informatica 顾问的集体知识，这些顾问与世界各地的组织合作，共同计划、开发、部署和维护成功的数据管理解决方案。

您可以在以下网址中找到 Informatica Velocity 资源：http://velocity.informatica.com。如果您对 Informatica Velocity 有任何疑问、意见或建议，请通过 ips@informatica.com 与 Informatica 专业服务联系。

Informatica Marketplace

Informatica Marketplace 是一个论坛，该论坛中提供的解决方案可扩展和增强您的 Informatica 实施。利用 Informatica 开发人员和合作伙伴在 Marketplace 中提供的数以百计的解决方案，可提高您的工作效率并加快项目实施时间。您可以在以下网址中找到 Informatica Marketplace：https://marketplace.informatica.com。

Informatica 全球客户支持部门

您可以通过电话或 Informatica Network 与全球支持中心联系。

要查找您当地的 Informatica 全球客户支持部门电话号码，请访问 Informatica 网站，链接为：https://www.informatica.com/services-and-training/customer-success-services/contact-us.html。

要在 Informatica Network 上查找在线支持资源，请访问 https://network.informatica.com，然后选择 eSupport 选项。

第 1 章

业务实体服务简介

本章包括以下主题：

业务实体服务概览

业务实体服务是一组操作，可运行 MDM Hub 代码以创建、更新、删除和搜索业务实体中的基础对象记录。您可以开发自定义用户界面，在其中运行 Java 代码或 JavaScript 代码来执行业务实体服务调用。

例如，您可以创建业务实体服务，为供应商记录增加 Dun and Bradstreet 数据。为此，请配置业务实体服务，将供应商记录用作输入，接着从 Dun and Bradstreet 检索某些信息，并更新该记录，然后输出更新后的供应商记录。

业务实体中的基础对象包含以下业务实体服务：

读取

每个业务实体都包含用于执行读取操作的业务实体服务。

写入

每个业务实体都包含用于执行写入操作的业务实体服务。

搜索

任何具有可搜索字段的业务实体都包含用于执行搜索操作的业务实体服务。

例如，Person 业务实体具有可搜索字段。MDM Hub 可生成 ReadPerson、WritePerson 和 SearchPerson 业务实体服务。读取、写入和搜索业务实体服务步骤可用于读取、创建、更新、删除和搜索业务实体中的记录。

业务实体服务

一个业务实体服务执行一项操作。您可以使用 ReadBE、WriteBE 和 SearchBE 业务实体服务。

一个业务实体服务包含若干服务步骤。传入请求通过每个服务步骤传递。一个步骤的输出是下一个步骤的输入。一个步骤的输出可将信息传递到下一个步骤的输入。所有业务实体服务步骤都作为一个 Enterprise Java Bean 调用在单个事务中运行。MDM Hub 可处理异常。

注意: 使用业务实体服务之前，请验证操作引用存储。

ReadBE 业务实体服务

ReadBE 业务实体服务可从业务实体中的基础对象记录读取数据。

您可以使用 ReadBE 步骤指定分页参数，以设置要返回的记录数和要查看的结果的页码。

ReadBE 服务的结果不包括软删除的记录。

如果未在业务实体服务请求中传递 EffectiveDate 参数，则 MDM Hub 会假定生效日期为空，并且业务实体服务会从基础对象读取数据。如果传递了 EffectiveDate 参数，则 MDM Hub 会根据交叉引用记录计算最佳数据版本，并且读取业务实体服务会返回最新的最佳数据版本。

WriteBE 业务实体服务

WriteBE 业务实体服务可以更新业务实体元素中的数据，以及创建或删除业务实体子元素。

注意: WriteBE 业务实体服务使用现有信任设置来计算对基础对象的信任。使用此服务无法执行信任替代。

可选参数

下表介绍了可用于 WriteBE 业务实体服务的可选参数。

参数	说明
recordState	将记录状态设置为 ACTIVE、PENDING 或 DELETED。 **注意**: 当您设置 `recordState=ACTIVE` 并在软删除的记录上运行该服务时，该服务会将此记录还原为活动状态。
EffectivePeriod	指定一个有效期。如果未传递 `EffectivePeriod` 参数，则 MDM Hub 会假定时间段不受限制。 MDM Hub 不会检查根对象的有效期是否与子对象的有效期保持一致。创建或更新记录时，请确保父记录的有效期与子记录的有效期保持一致。

SearchBE 业务实体服务

可以使用 SearchBE 业务实体服务搜索业务实体中的根记录。

有关为智能搜索配置业务实体的信息，请参阅*《Multidomain MDM 配置指南》*。

业务实体服务端点

您可以通过 Enterprise JavaBeans (EJB) 端点、具象状态传输 (REST) 端点或简单对象访问协议 (SOAP) 端点访问业务实体服务。

REST 端点是从 EJB 的角度构建的。REST 业务实体服务配置定义了 REST URL 如何映射到 EJB 业务实体服务调用。

业务实体服务的 Enterprise JavaBeans 端点

Enterprise JavaBeans (EJB) 端点是所有类型的业务实体服务调用的基础端点。所有其他端点均映射到 EJB 端点

业务实体服务公开为无状态 EJB。无状态 EJB 容器可以共用和分配实例，还可以应用负载平衡策略，在域中的不同服务器之间分配负载。

EJB 端点接受使用用户名和密码进行身份验证。

业务实体服务的 REST 端点

具象状态传输 (REST) 端点调用会使业务实体服务可用作 Web 服务。

Web 应用程序描述语言 (WADL) 文件包含 REST Web 服务的 XML 描述以及所有 REST URL 和所有 REST 参数。MDM Hub 将为每个操作引用存储生成 WADL 文件。

您可从以下位置下载每个操作引用存储的 WADL 文件：

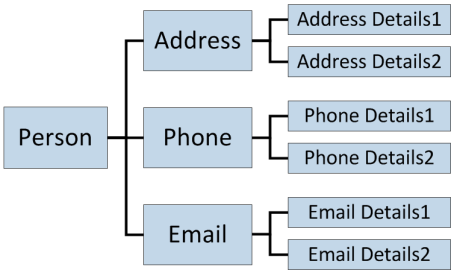
```
http://<host>:<端口>/cmx/csfiles
```

REST 和 EJB 业务实体服务调用

执行业务实体服务调用时，您可以指定某些子分支，而不请求整个业务实体。

例如，假如您要对包含一个 Person 根记录和多个子分支的业务实体执行读取操作，而 Person 基础对象包含 Address、Phone 和 Email 子基础对象，同时每个子基础对象又包含两个孙基础对象。

下图显示了包含多个分支的业务实体的结构：



您可以通过单个请求从位于不同深度的多个子分支读取数据。例如，您可以通过单个请求读取 Person、Phone、Phone Details1、Phone Details 2、Email 和 Email Details 2。

以下 URL 示例显示了如何发出 REST 读取请求以获取行 ID 为 1242 的 Person 记录，以及 Address Details 1 和 Email 子记录：

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

业务实体服务的 SOAP 端点

简单对象访问协议 (SOAP) 端点调用会使业务实体服务可用作 Web 服务。

Web 服务描述语言 (WSDL) 文件包含 Web 服务的 XML 描述、SOAP 请求与响应的格式以及所有参数。MDM Hub 将为每个操作引用存储生成 WSDL 文件。

标识根记录

可以使用以下方法之一来标识根记录：

- rowid。记录的 ROWID_OBJECT 列中的值。
- systemName 和 sourceKey。systemName 是记录所属的系统的名称。sourceKey 是记录的 PKEY_SRC_OBJECT 列中的值。
- 对象的全局业务标识符 (GBID)。GBID 可以是复合值，在这种情况下，您必须传递所有的值。

  注意: GBID 方法仅适用于 ReadBE 服务。

以下示例代码使用了 systemName 和 sourceKey 来标识记录：

```
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);
```

安全和数据筛选器

如果基础对象和资源具有用户角色特权，则业务实体会继承这些特权。要访问业务实体记录，用户角色必须对业务实体的根基础对象以及其他资源具有相应特权。

业务实体服务还会继承在业务实体字段上设置的任何数据筛选器。

有关安全和数据筛选器的详细信息，请参阅*《Multidomain MDM 置备工具指南》*。

第 2 章

Enterprise Java Bean 业务实体服务调用

本章包括以下主题：

Enterprise Java Bean 业务实体服务调用概览

您可以执行 Enterprise Java Bean (EJB) 业务实体服务调用，在业务实体中创建、更新、删除和搜索基础对象记录。您还可以创建 Java 代码来运行 EJB 业务实体服务调用。

您可以创建基于标准服务数据对象 (SDO) 类的 Java 代码，或者也可以创建基于 java 类（由 MDM Hub 基于业务实体配置和业务实体服务配置生成）的 Java 代码。

使用标准 SDO 类的 Java 代码示例

本示例显示的 Java 代码会运行基于标准服务数据对象 (SDO) 类的 Enterprise Java Bean (EJB) 调用。

该示例位于资源工具包中的以下文件：C:\<MDM Hub 安装目录>\hub\resourcekit\samples\COS\source\java\com\informatica\mdm\sample\cs\DynamicSDO.java

以下 Java 代码基于标准 SDO 类，并会运行 EJB 业务实体服务调用，从而创建 Person 基础对象记录、添加多个子记录、删除一个子记录以及删除该 Person 记录和所有子记录：

```
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();
    }
```

```
}
```

使用生成的 SDO 类的 Java 代码示例

本示例显示的 Java 代码会基于 Java 类（由 MDM Hub 基于业务实体配置和业务实体服务配置生成）运行 Enterprise Java Bean (EJB) 调用。

该示例位于资源工具包中的以下文件：C:\<MDM Hub 安装目录>\hub\resourcekit\samples\COS\source\java\com\informatica\mdm\sample\cs\GeneratedSDO.java

以下 Java 代码基于生成的类，并会运行 EJB 业务实体服务调用，从而创建 Person 基础对象记录、添加多个子记录、删除一个子记录以及删除该 Person 记录和所有子记录：

```
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();
    }
}
```

第 3 章

具象状态传输业务实体服务调用

本章包括以下主题：

业务实体服务的 REST API 概览

REST 端点调用会使所有业务实体服务可用作 Web 服务。

您可以执行 REST 调用，在业务实体中创建、更新、删除和搜索基础对象记录和相关子记录。您可以执行合并、取消合并和匹配记录等操作。您可以执行 REST 调用，创建、更新、搜索和执行任务。此外，您还可以执行 REST 调用，创建、更新和删除文件，例如任务或记录的附件。

REST 业务实体服务调用是使用统一资源定位器 (URL) 格式的 Web 服务请求。MDM Hub 将为业务实体中的每个基础对象分配唯一的 URL。您可以使用这个唯一的 URL 来标识要更新或删除的基础对象。

注意: 使用 REST API 调用业务实体服务之前，请验证操作引用存储。

支持的 REST 方法

业务实体服务的 REST API 使用标准 HTTP 方法对资源（例如记录、任务和文件）执行操作。

业务实体服务的 REST API 支持以下 HTTP 请求方法：

方法	说明
GET	检索有关记录、任务或文件的信息。
POST	创建任务、根记录、子记录或文件。 **注意:** POST 请求中的操作引用存储 (ORS) 名称区分大小写。如果 ORS 名称与 MDM Hub 中的名称不匹配，则会发生错误。
PUT	更新根记录、子记录、任务或文件。
PATCH	更新任务的一部分。
DELETE	删除根记录、子记录或文件。

身份验证方法

业务实体服务的 REST 端点使用基本 HTTP 身份验证方法对用户进行身份验证。首次使用浏览器连接到业务实体服务时，您必须提供自己的 MDM Hub 用户名和密码。身份验证成功后，您就可以使用业务实体服务 REST API 执行操作。

浏览器将缓存用户凭据，并将这些凭据用于随后的每次业务实体服务请求。

用于从第三方应用程序进行登录的身份验证 Cookie

使用身份验证 Cookie 可以对 MDM Hub 用户进行身份验证，并从第三方应用程序调用业务实体服务。您可以获取基于经过身份验证的用户的凭据的 Cookie。保存该 Cookie 后，可以用它来调用 REST API，而不需要对用户名和密码进行硬编码。

发出以下 POST 请求，用您的用户名和密码登录到 Entity 360 视图：

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

登录操作成功后，服务器会在 set-cookie 表头字段中返回身份验证 Cookie。以下示例代码显示了响应表头中的 set-cookie：

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

请存储该哈希值并在 API 调用的请求表头中使用该哈希值。您不需要为 API 调用提供用户名和密码。

以下示例显示了如何在 API 请求表头中使用身份验证 Cookie：

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

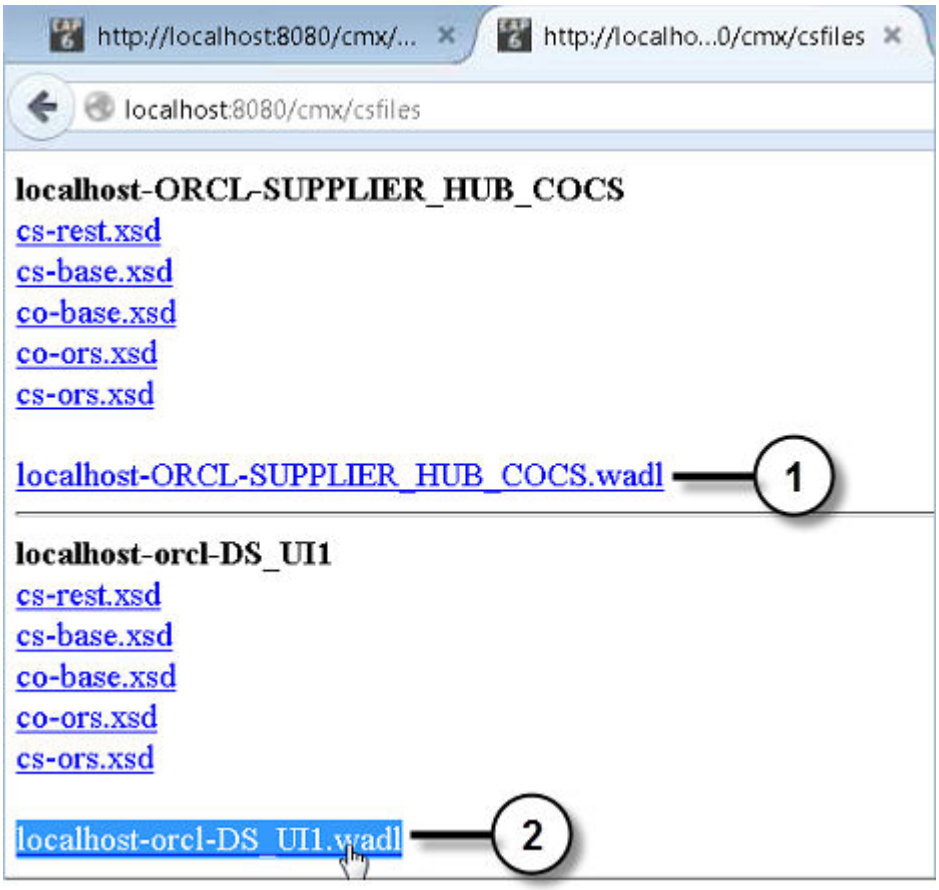
Web 应用程序描述语言文件

Web 应用程序描述语言 (WADL) 文件包含 REST Web 服务的 XML 描述、所有 REST URL 和所有 REST 参数。MDM Hub 将为每个操作引用存储生成 WADL 文件。

每个操作引用存储的 WADL 文件位于以下位置：

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

下图显示了每个操作引用存储的 WADL 文件的下载位置：



1. SUPPLIER_HUB_COCS 操作引用存储的 WADL 文件的下载链接
2. DS_UI1 操作引用存储的 WADL 文件的下载链接

REST 统一资源定位器

您可以使用 REST URL 对业务实体服务执行 REST 调用。

REST URL 使用以下语法：

```
http://<host>:<端口>/<上下文>/<数据库 ID>/<路径>
```

URL 包含以下字段：

主机

正在运行数据库的主机。

端口

数据库侦听器所使用的端口号。

context

业务实体、搜索、查询、匹配和任务 API 的上下文为 cmx/cs。

匹配列 API 的上下文为 cmx。

文件 API 的上下文为 cmx/file。

注意: 在托管 MDM 环境中，请在上下文中加入租户名称。例如，上下文可以是 <租户名称>/cmx/cs 或 <租户名称>/cmx/file。

database ID

在 Hub 控制台的“数据库”工具中注册的 ORS 的 ID。

path

您要使用 API 的对象，例如记录、任务或文件。

如果 URL 用于根记录，则路径是根对象名称后跟唯一标识符。

以下是 Person 根记录的路径的示例：Person/798243.json。

如果 URL 用于根对象直接子代的记录，则路径还包含子记录名称和唯一标识符。

以下是帐单地址记录（属于 Person 根记录的子代）的路径示例：

Person/798243/BillAddresses/121522.json。

如果 URL 用于深度为 2 或以上的子记录，则路径还包含深度。

以下 URL 是深度为 2 的子记录的 REST URL 示例：

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

注意: 参数区分大小写。请确保 REST URL 中参数名称的大小写与 REST 配置中参数名称的大小写匹配。

表头和主体配置

REST 操作会将 HTTP 方法与指向资源的完整 URL 结合起来。对于完整请求，请将 REST 操作与相应的 HTTP 表头和任何必需数据结合起来。REST 请求包含表头部分和主体部分。您可以使用 JSON 或 XML 格式定义请求。

请求表头

您可以使用请求表头定义 REST 操作的操作参数或元数据。表头包含一系列字段-值对。API 请求行包含方法和 URL。请在请求行之后指定表头字段。

要构造 REST API 请求表头，请在 <METHOD> <<host>:<port>/<context>/<database ID>/<Path> 请求行之后添加表头字段，如以下示例所示：

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

下表介绍了部分常用的请求表头字段：

请求的组成部分	说明
Content-Type	请求中的数据的媒体类型。如果在 REST 请求中包含主体，则必须在 Content-Type 表头字段中指定主体的媒体类型。请在 PUT 和 POST 请求中包含 Content-Type 表头字段。
Accept	响应中的数据的媒体类型。要指定请求格式，请在表头中使用 **application/<json/xml>**，或者将 **.json** 或 **.xml** 添加到 URL。默认值为 XML。

请求主体

使用 REST API 请求主体在请求中发送数据。请求主体与某种可附加主体的方法（例如 POST 和 PUT 方法）结合使用。数据的主体写到表头行之后。如果请求消息包含主体，请使用 Content-Type 表头字段在请求表头中指定主体的格式。

XML 架构定义 (XSD) 文件介绍了可用的元素和属性。请求主体的内容取决于在 XSD 文件中定义的元素类型。

XSD 文件位于以下位置：

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

XML 格式

如果使用 XML 请求格式，请将请求对象定义为结束标记集。

请使用以下 XML 格式定义请求对象：

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

以下示例显示了请求对象的 XML 表示：

```
<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 格式

如果使用 JSON 请求格式，请为请求对象定义类型属性。

请使用以下 JSON 格式指定请求对象：

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

以下示例显示了请求对象的 JSON 表示：

```
{
     "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: ''
         }
       }
     }]
}
```

标准查询参数

业务实体服务 REST API 使用标准查询参数筛选、分页和展开结果。

使用问号 (?) 分隔查询参数和其他参数。查询参数是以等号分隔的键-值对。使用与号 (&) 分隔一系列查询参数。

以下 REST 请求 URL 显示了如何使用查询参数：

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

请使用以下标准查询参数：

参数	说明
recordsToReturn	指定返回的行数。默认值为 10。
firstRecord	指定结果中的第一行。默认值为 1。后续的调用使用此参数读取更多页。
searchToken	指定随上一个请求一起返回的搜索标志。您可以使用搜索标志提取后续的搜索结果页。例如，以下查询将列出第一页：`/Person/123/Phone` 以下查询将返回第二页：`/Person/123/Phone?searchToken=SVR1.AZAM5&firstRecord=10`
returnTotal	如果设置为 true，则返回结果中的记录数。默认值为 false。
depth	指定包含在结果中的子级数。

UTC 格式的日期和时间

在请求和响应中，所有日期和时间都采用 UTC（协调世界时）格式，而且日期和时间可含有或不含有特定时区的时差。

在请求主体中指定日期和时间时，请使用符合 ISO 规范 8601 的 Date and Time Formats (NOTE-datetime) 中定义的一个格式。

以下准则选自 NOTE-datetime 文档：

类型	语法	示例
日期：年	YYYY	1997
日期：年月	YYYY-MM	1997-07
日期：年月日	YYYY-MM-DD	1997-07-16
日期加小时和分钟	YYYY-MM-DDThh:mmTZD	1997-07-16T19:20+01:00
日期加小时、分钟和秒钟	YYYY-MM-DDThh:mm:ssTZD	1997-07-16T19:20:30+01:00
日期加小时、分钟、秒钟和秒钟的小数部分	YYYY-MM-DDThh:mm:ss.sTZD	1997-07-16T19:20:30.45+01:00

其中：

- YYYY = 四位数年份
- MM = 月份，从 01 到 12 的两位数
- DD = 几号，从 01 到 31 的两位数

- T = 在日期后时间前的文本值
- hh = 小时，从 00 到 23 的两位数
- mm = 分钟，从 00 到 59 的两位数
- ss = 秒钟，从 00 到 59 的两位数
- s = 一或多位数，表示秒的小数部分
- TZD = 时区指示符（Z 或 +hh:mm 或 -hh:mm）

  - Z 表示 UTC 时间

  - +hh:mm 表示比 UTC 时间快的本地时区

  - -hh:mm 表示比 UTC 时间慢的本地时区

配置 WebLogic 以运行业务实体服务 REST 调用

由于业务实体服务 REST 调用使用基本 HTTP 身份验证，因此必须为 REST 调用禁用 WebLogic 服务器身份验证。要配置 WebLogic 以运行业务实体服务 REST 调用，请编辑 WebLogic `config.xml` 文件。

1. 导航到以下 WebLogic 目录：

   在 UNIX 中。

   ```
   <WebLogic 安装目录>/user_projects/domains/base_domain/config
   ```

   在 Windows 中。

   ```
   <WebLogic 安装目录>\user_projects\domains\base_domain\config
   ```

2. 在文本编辑器中打开以下文件：

   ```
   config.xml
   ```

3. 在 </security-configuration> 结束标记之前，添加以下 XML 代码：

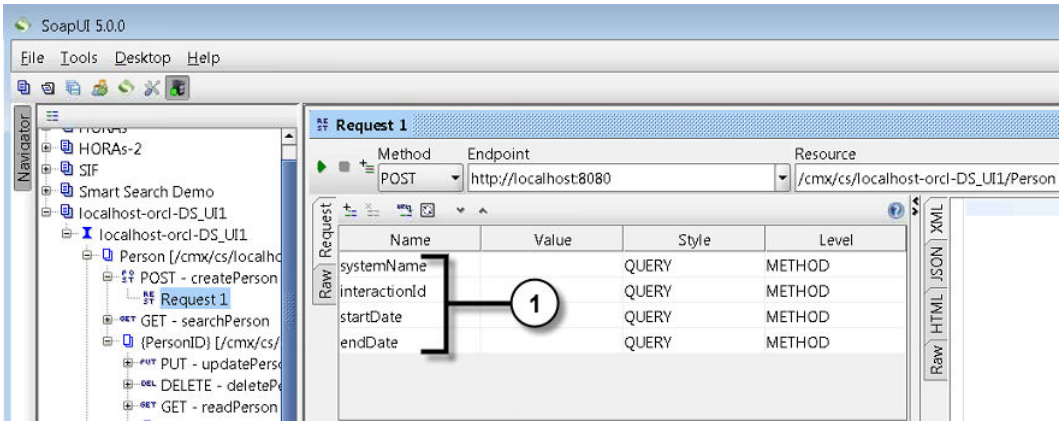
   ```
   <enforce-valid-basic-auth-credentials>
      false
   </enforce-valid-basic-auth-credentials>
   ```

查看输入参数和输出参数

您可以使用诸如 SoapUI 等功能测试工具查看 REST API 输入参数和输出参数。

请下载 WADL 文件，然后将该文件导入到功能测试工具中以创建 REST 项目。

下图显示了 SoapUI 中 createPerson REST API 的输入参数：



1. createPerson REST API 的输入参数

注意: 在 Websphere 环境中生成的 WADL 文件可能无法正常导入到 SoapUI。如果输入参数未显示在 SoapUI 中，请编辑 WADL 文件，从每个 param 元素中删除 xmlns 属性，然后重新导入该 WADL 文件。

JavaScript 模板

以下代码示例显示了一个基本模板，您可以通过修改此模板来为 REST 业务实体服务调用创建 JavaScript 代码。您需要使用 jQuery java 脚本库。

```
(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 示例

资源工具包含有示例 Java 源代码，该代码显示了如何执行 REST 业务实体服务调用。

该示例代码位于以下文件：

<MDM Hub 安装目录>\hub\resourcekit\samples\COS\source\resources\webapp\rest-api.html

以下代码显示的 REST API 调用可以创建 Person 根记录、添加多个子记录、删除一个子记录以及删除该 Person 记录和所有子记录：

```
<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>
```

业务实体服务的 REST API 参考

“业务实体服务的 REST API 参考”列出了 REST API 并提供了每个 API 的说明。API 参考还包含有关 URL、查询参数、示例请求和示例响应的信息。

获取元数据

获取元数据 REST API 会返回业务实体或业务实体关系的数据结构。

此 API 使用 GET 方法返回业务实体的以下元数据：

- 业务实体的结构
- 字段列表
- 字段类型（例如数据类型和大小）
- 操作的安全设置，例如创建、更新和合并
- 节点或字段的本地化标签
- 代码名称和查找字段的显示字段
- 字段和查找字段的默认值

  注意: 相关查找字段不支持多个默认值。

此 API 会返回业务实体关系的以下详细信息：

- 关系的名称及其标签。
- “开始”和“结束”业务实体。
- 关系方向。

请求 URL

针对业务实体的获取元数据 URL 的格式如下：

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

针对关系的获取元数据 URL 的格式如下：

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

使用查询参数“action=meta”检索元数据信息。请务必正确指定业务实体名称。

向获取元数据 URL 发出 HTTP GET 请求：

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

您可以向请求添加 HTTP 表头。

示例 API 请求

以下示例请求将检索 Person 业务实体和根节点的元数据信息：

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

以下示例请求包含表头，可检索 JSON 格式的 Person 业务实体的元数据信息：

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

以下示例请求检索 HouseholdContainsMemberPerson 关系的元数据信息：

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

示例 API 响应

实体和关系的示例响应包含安全权限。第一个 operations 部分定义了可能的权限。object 部分列出了您对整个业务实体或关系的权限。fields 部分定义了您在字段级别的权限。

以下示例以 JSON 格式显示了“人员”业务实体的部分数据结构。


```
{
  "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
    }
  }
```

列出元数据

列出元数据 REST API 会返回您定义的业务实体列表或关系列表。响应包含时间轴信息和安全信息（如果业务实体包含此信息）。您可以使用此 API 检索从某个实体开始并结束或从指定实体开始并结束的关系列表。

此 API 使用 GET 方法。

列出元数据 URL

列出元数据 URL 具有以下格式的实体元数据：

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

列出元数据 URL 具有以下格式的关系元数据：

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

向列出元数据 URL 发出以下 HTTP GET 请求：

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

查询参数

您可以将查询参数附加到请求 URL 来筛选业务实体关系的搜索结果。您可以指定方向并搜索关系的起始和结束业务实体。

下表列出了查询参数：

参数	说明
start	可选。指定关系源自哪个实体。 例如，meta/relationship?start=Organization 查询会返回从 Organization 业务实体开始的所有关系。
finish	可选。指定关系在哪个实体结束。 例如，meta/relationship?finish=Person 查询会返回在 Person 业务实体结束的所有关系。

您可以指定两个参数以便检索两个业务实体之间的所有关系。例如，meta/relationship?start=Organization&finish=Person 查询会返回从 Organization 业务实体开始并以 Person 业务实体结束的所有关系。

示例 API 请求

以下示例请求检索了配置后的业务实体列表：

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

以下示例请求检索了配置后的关系列表：

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

以下示例请求检索了以 Organization 业务实体开始并以 Person 业务实体结束的关系列表：

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

示例 API 响应

以下示例显示了配置的关系列表的摘录：

```
{
  "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
      }
    },
    ...
  ]
}
```

列出匹配列

列出匹配列 REST API 可以返回一个指定业务实体的匹配规则集列表或一个指定匹配规则集的匹配列列表。您可以生成一个匹配列列表然后在 SearchMatch REST API 中使用这些匹配列。

有关配置匹配列和匹配规则集的信息，请参阅*《Multidomain MDM 配置指南》*。

此 API 使用 GET 方法。

请求 URL

列出匹配列 URL 的上下文为 cmx。在托管 MDM 环境中，请在上下文中加入租户名称，例如 <租户名称>/cmx。

列出匹配列 URL 使用以下格式：

返回所有匹配规则集的 URL

使用以下 URL 列出指定业务实体的所有匹配规则集：

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

向列出匹配列 URL 发出以下 HTTP GET 请求：

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

返回用于指定匹配规则集的匹配列的 URL

使用以下 URL 列出一个指定匹配规则集中的所有匹配列：

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

示例 API 请求

请求匹配规则集

以下示例请求将列出 Person 业务实体的匹配规则集：

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

请求匹配列

以下示例请求将列出用于匹配规则集 IDL2 中的匹配列：

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

示例 API 响应

以下示例响应包含 IDL2 匹配规则集中包含的匹配列：

```
{
 "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
    }
  ]
 }
]
}
```

读取记录

读取记录 REST API 可返回业务实体中根记录的详细信息。您可以使用此 API 返回根记录的子记录的详细信息，还可以使用此 API 查看记录的内容元数据。

此 API 使用 GET 方法。

可以对结果集进行排序，以便按升序或降序查看信息。如果需要更多复杂参数，请使用 POST 方法。例如，如果要检索数据并按一组字段对子元素进行排序。

请求 URL

使用行 ID 或源系统的名称和源键可以在请求 URL 中指定记录。

读取记录 URL 可以使用以下格式：

带有 rowId 的 URL

指定行 ID 时，请使用以下 URL 格式：

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

向 URL 发出以下 HTTP GET 请求：

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

带有源系统名称和源键的 URL

指定源系统名称和源键时，请使用以下 URL 格式：

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

带有对象的系统名称和全局业务标识符 (GBID) 的 URL

指定源系统名称和 GBID 时，请使用以下 URL 格式：

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

仅带有 GBID 的 URL

仅指定 GBID 时，请使用以下 URL 格式：

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

带有多个 GBID 的 URL

指定多个 GBID 时，请使用以下 URL 格式：

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

用于返回子节点的详细信息的 URL

使用以下 URL 格式可返回子节点的详细信息：

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

用于返回子节点的详细信息的 URL

使用以下 URL 格式可返回子节点的详细信息：

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

例如，children= BillAddresses/Address,Email

用于返回特定节点的详细信息的 URL

使用以下 URL 格式可返回特定节点的详细信息：

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

用于返回特定节点的子项详细信息的 URL

使用以下 URL 格式可返回特定节点的子项的详细信息：

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

用于返回记录的内容元数据的 URL

使用以下 URL 格式返回记录的内容元数据：

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

例如，您可以使用以下 GET 请求来检索子记录的匹配项：

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

用于按字段对子元素进行排序的 URL

使用以下 URL 格式可按字段对子元素进行排序：

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

使用字符 - 作为后缀可指定降序。

查询参数

您可将查询参数附加到请求 URL，以筛选记录的详细信息。

下表列出了查询参数：

参数	说明
depth	返回的子级数。指定 2 将返回根节点及其直接子代；指定 3 将返回根节点及其直接子代和孙代。默认值为 1。
effectiveDate	要为其检索数据的日期。
readSystemFields	指示是否在结果中返回系统字段。默认值为 false。
recordStates	记录的状态。请提供逗号分隔的状态列表。支持的记录状态为 ACTIVE、PENDING 和 DELETED。默认值为 ACTIVE。

参数	说明
contentMetadata	记录的元数据。请提供逗号分隔的列表。例如，XREF、PENDING_XREF、DELETED_XREF、HISTORY、XREF_HISTORY 和 MATCH。 选择 MATCH 时，响应将包含从 _MTCH 表检索的匹配记录的列表。
historyDate	要为其检索历史记录数据的日期。响应包含从 _HIST 表检索的指定日期的记录数据。 您可以使用 historyDate 与 contentMetadata 参数来检索历史元数据。将 contentMetadata 设置为 XREF、BVT 或 TRUST。 - XREF。响应包含 _HXRF 表中的历史交叉引用数据。 - BVT。响应包含 _HCTL 表中的历史最佳数据版本。 - TRUST。响应包含 _HCTL 和 _HVXR 表的历史信任设置。
children	子节点名称或路径的逗号分隔列表。
suppressLinks	指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。默认值为 false。 例如，**Person/1242?depth=10&suppressLinks=true** 查询最多显示 10 个子级的记录详细信息，并且响应中没有任何父子链接可见。
order	带有可选前缀 + 或 - 的字段名称的逗号分隔列表。前缀 + 表示按升序对结果排序，前缀 - 表示按降序对结果排序。默认值为 +。如果指定多个参数，则结果集将先按列表中的第一个参数进行排序，接着按第二个参数进行排序，依此类推。 例如，**Person/1242/Names?order=-name** 查询将显示按姓名降序排序的结果。 **Person/1242/BillAddresses?order=rowidObject,-effStartDate** 查询将显示按行 ID 升序排序且按有效期开始日期降序排序的帐单地址。

以下示例显示了如何筛选记录的详细信息：

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

相关主题：

- “UTC 格式的日期和时间” 页面上 27

用于指定子元素的排序顺序的 POST 请求

可以使用 POST 请求按多个字段对结果集进行排序。请在 POST 主体中包含参数或字段。

以下示例请求显示了如何使用 POST 请求执行读取操作并按多个字段对数据进行排序：

```
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"}]
       }
```

```
    ]}
  }
 }
}
```

注意: 在业务实体的每个级别，仅允许对每种子项类型使用一种排序顺序。

排序顺序的注意事项

读取记录 API 支持对每个业务实体子节点按一个或多个字段进行排序。下一节介绍了指定排序顺序时需要了解的某些注意事项。

- 如果为孙元素而非子元素指定排序顺序，则孙元素将按指定的顺序进行排序。子元素不会按照为孙元素指定的排序顺序进行排序。以下是一个示例请求：

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

  在此示例请求中，虽然为孙元素 PhonePermissions 指定了降序排序顺序，但没有为子元素 Phone 指定任何顺序，因此 Phone 不会按照 PhonePermissions 排序顺序进行排序。

- 如果为子元素而非孙元素指定排序顺序，则子元素将按指定的顺序进行排序。孙元素不会按照为子元素指定的排序顺序进行排序。以下是一个示例请求：

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

  在此示例请求中，为子元素 Phone 而非孙元素 PhonePermissions 指定了排序顺序，因此子元素 Phone 将按指定的顺序进行排序。

- 如果同时为子元素和孙元素指定排序顺序，则子元素和孙元素都将按该排序顺序进行排序。以下示例请求同时为 Phone（子元素）和 PhonePermissions（孙元素）指定了排序顺序：

  ```
  {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"}]},
      ]}
  }}}
  ```

- 子元素只能按子元素本身中的列进行排序，而孙元素可以按孙元素中的列进行排序。在以下示例请求中，Phone 按 PhoneType 进行排序，而 PhonePermissions 按列 1 进行排序。PhoneType 是 Phone（子元素）中的一个列，而列 1 是 PhonePermissions（孙元素）中的一个列。

  ```
  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
  ```

- 在业务实体的每个级别，仅允许对每种子项类型使用一种排序顺序。在以下请求中，您为不同父元素的 PhonePermissions 子元素指定了不同的排序顺序。但是，由于第一排序顺序指定为降序，因此两个父元素（行 ID 861 和行 ID 862）的 PhonePermissions 子元素都按降序进行排序。

  ```
  {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"}]}
    ]}
}}}
```

示例 API 请求

以下示例请求将返回 Person 业务实体中的根记录的详细信息。

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

以下示例请求将返回 rowId 为 2 的子记录的详细信息。子基础对象是 genderCd，并且子记录的深度为 2。

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

以下示例请求使用系统名称和源键返回根记录的详细信息：

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

以下示例请求将返回根记录及其 XREF 记录的详细信息：

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

以下示例请求将返回根记录的详细信息并以降序对姓名进行排序。

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

以下示例请求将返回按行 ID 升序排序且按有效期开始日期降序排序的帐单地址：

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

示例 API 响应

以下示例显示了 Person 业务实体中的根记录的详细信息。


```
{
    "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"
        }
}
```


以下示例显示了 genderCd 基础对象中的子记录的详细信息。


```
{
    "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"
}
```


创建记录

创建记录 REST API 可在指定的业务实体中创建记录。请在请求主体中发送记录数据。然后使用升级 API 在业务实体中升级并添加记录。

此 API 使用 POST 方法创建记录。

请求 URL

创建记录 URL 使用以下格式：

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

注意: 源系统的名称是 URL 中的必需参数。

向创建记录 URL 发出以下 HTTP POST 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

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

URL 参数

源系统的名称是请求 URL 中的必需参数。

下表列出了可以在 URL 中使用的参数：

参数	说明
systemName	源系统的名称。
interactionId	交互 ID。您可以将多个请求分组为单个交互。所有更改都基于交互 ID 进行。
startDate 和 endDate	指定记录保持有效的时间段。请为启用了时间轴的基础对象提供这些参数。
validateOnly	指示写入业务实体服务是否对传入数据进行验证。默认值为 false。
recordState	记录的状态。使用此参数指定记录的初始状态。可以使用 ACTIVE 或 PENDING。默认值为 ACTIVE。
taskComment	向 API 触发的工作流任务添加备注。
taskAttachments	如果启用了任务附件，则向 API 触发的工作流任务附加文件。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

请求主体

在 REST 请求主体中发送记录的数据。使用 JSON 格式或 XML 格式发送数据。您可以使用获取元数据 API 获取业务实体的结构，并在请求主体中提供必需的参数值。

响应表头和主体

如果响应成功，API 将在响应表头中返回 interactionId 和 processId，并在响应主体中返回记录详细信息。

如果进程生成某个交互 ID 并使用它创建记录，API 将返回该交互 ID。如果进程启动工作流而不直接将记录保存到数据库，API 将返回进程 ID，即工作流进程的 ID。

以下示例显示了带有交互 ID 和进程 ID 的响应表头：

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

响应主体包含记录和生成的行 ID。

示例 API 请求

以下示例请求将在“人员”业务实体中创建一个记录。该请求向 API 触发的工作流任务添加备注和附件。

```
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"
            }
        ]
    }
}
```

示例 API 响应

以下示例响应显示了成功创建记录后的响应表头和主体：

```
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"
                }
            ]
        }
    }
}
```

更新记录

更新记录 REST API 可更新指定的根记录及其子记录。请在请求 URL 中发送记录的 ID。请在请求主体中发送更改摘要。

更改后，如果记录处于挂起状态，请使用升级 API 来升级更改。例如，如果更新触发了审阅工作流，记录则会处于挂起状态直至审阅完成。

此 API 使用 POST 方法。

对于根记录，您可以选择使用简化的 PUT 版本，其中请求主体包含已更改字段，而不是更改摘要。要更新根记录和相关的子记录或孙记录，请使用 POST 版本。您还可以选择将 PUT 版本与 PKEY 一起使用，或者指定子记录的 rowidObject。

请求 URL

更新记录 URL 使用以下格式：

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

注意: 源系统的名称是 URL 中的必需参数。

向更新记录 URL 发出以下 HTTP POST 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

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

查询参数

源系统的名称是必需的查询参数。

您可以在请求中使用以下查询参数：

参数	说明
systemName	源系统的名称。
interactionId	交互 ID。您可以将多个请求分组为单个交互。所有更改都基于交互 ID 进行。
startdate 和 enddate	指定记录保持有效的时间段。请为启用了时间轴的基础对象提供这些参数。
validateOnly	指示写入业务实体服务是否对传入数据进行验证。默认值为 false。
recordState	设置记录的状态。可以使用 ACTIVE、PENDING 或 DELETED。 例如，如果设置 `recordState=ACTIVE`，且请求在软删除的记录上运行，则请求会将记录还原为活动状态。
taskComment	向 API 触发的工作流任务添加备注。
taskAttachments	如果启用了任务附件，则向 API 触发的工作流任务附加文件。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

请求主体

请在 REST 请求主体中发送要更新的数据。使用 JSON 格式或 XML 格式发送数据。

请提供新的参数值。您可以使用 $original 参数指示要更新的参数的旧值。

您还可以将以下属性与子记录结合使用：

属性/元素	类型	说明
MATCH	对象	如果您想为子记录将候选项添加到匹配表或从匹配表中移除候选项，请将 MATCH 对象添加到子记录。
MERGE	对象	如果您想合并子记录或将候选项从合并中移除，请将 MERGE 对象添加到子记录中。

以下 JSON 代码示例将根记录中的名字更改为 Bob：

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

以下 JSON 代码示例移除了地址子记录的匹配候选项，并且定义了两个电话号码子记录的合并：

```
{
  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
            ]
          }
        }
      ]
    }
  }
```

响应表头

如果响应成功，API 将在响应表头中返回 interactionId 和 processId，并在响应主体中返回记录详细信息。

如果进程生成某个交互 ID 并使用它创建记录，API 将返回该交互 ID。如果进程启动工作流而不直接将记录保存到数据库，API 将返回进程 ID，即工作流进程的 ID

以下示例显示了带有交互 ID 和进程 ID 的响应表头：

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

响应主体包含记录和生成的 rowId。

示例 API 请求

以下示例请求将更新业务实体中的根记录及其子记录。Person 是业务实体，而 Phone 是子基础对象。该请求向 API 触发的工作流任务添加备注和附件。

```
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"
    }
}
```

示例 API 响应

以下示例响应显示了成功更新记录后的响应表头和主体：

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

```
    }
}
```

删除记录

删除记录 REST API 可删除业务实体中的根记录。使用此 API 可删除根记录的子记录。

此 API 使用 DELETE 方法删除记录。

请求 URL

删除记录 URL 使用以下格式：

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

注意: 源系统的名称是 URL 中的必需参数。

向删除记录 URL 发出以下 HTTP DELETE 请求：

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

使用以下 URL 格式删除根记录的子记录：

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

查询参数

源系统的名称是必需的 URL 参数。请使用 systemName 参数指定源系统。

示例 API 请求

以下示例请求将删除 Person 业务实体中的一个根记录：

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

示例 API 响应

以下示例响应显示了成功删除 Person 业务实体中的根记录后的响应：

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

列出记录

列出记录 REST API 可返回查找值或外键值列表。查找可提供引用数据并列出给定列的可能值列表。

此 API 使用 GET 方法。

使用此 API 还可以获取查找代码值和查找代码说明。您可以为查找指定排序顺序。如果需要更多复杂参数，请使用 POST 方法。

请求 URL

列出记录 REST URL 使用以下格式：

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

向 URL 发出以下 HTTP GET 请求：

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

使用以下 URL 格式列出查找值的代码：

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

注意: 使用获取元数据 API 可获取用于列出查找值的精确 URL。

查询参数

可以将查询参数附加到请求 URL，以筛选结果。

下表列出了查询参数：

参数	说明
suppressLinks	指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。默认值为 false。 例如，`Person/1242?depth=10&suppressLinks=true` 查询最多显示 10 个子级的记录详细信息，并且响应中没有任何父子链接可见。
顺序	用于以升序或降序列出查找值。使用字符 + 作为前缀可指定升序，而使用字符 - 作为前缀可指定降序。默认情况下，如果未指定前缀，则结果集将按升序进行排序。 例如，`LUNamePrefix?action=list&order=-namePrefixDisp` 查询将列出名称的前缀，并按前缀的显示名称以降序对这些前缀进行排序。

用于指定排序顺序的 POST 请求

可以使用 POST 请求来指定查找值的排序顺序。请在 POST 主体中包含参数或字段。

以下示例显示了如何使用 POST 请求执行列出操作：

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

示例 API 请求

以下示例请求将列出查找值：

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

以下示例请求将列出性别查找值的代码：

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

以下示例请求将列出名称的前缀，并以降序列出这些前缀的显示名称：

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

示例 API 响应

以下示例响应显示了查找值的列表：

```
{
    "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"
        }
    ]
}
```

以下示例响应显示了查找值的代码：

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

搜索记录

搜索记录 REST API 在可搜索的根记录及所有子记录中搜索已编制索引的值。您可以使用筛选器和构面查看搜索结果的子集。构面可将搜索结果分组为各个类别，而筛选器可缩小搜索结果范围。此 API 将搜索配置为可搜索字段的所有字段，并返回与搜索条件匹配的记录。

此 API 使用 GET 方法搜索可搜索字段的索引。

请求 URL

搜索记录 URL 使用以下格式：

用于简单搜索的 URL

请对简单搜索使用以下 URL：

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

向搜索记录 URL 发出以下 HTTP GET 请求：

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

用于字段搜索的 URL

请对字段搜索使用以下 URL：

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

如果为业务实体字段指定负数值，例如 -120，则返回的记录排名可能会受到影响。

用于构面搜索的 URL

请对包含构面的简单搜索使用以下 URL：

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

请对包含构面的字段搜索使用以下 URL：

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

用于筛选器搜索的 URL

请对包含筛选器的简单搜索使用以下 URL：

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

请在搜索中使用 q 或 fq 参数。

URL 编码

请使用 URL 编码，因为 URL 包含空格和单引号等字符。

以下示例显示了搜索记录 URL 的 URL 编码表示形式：

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

查询参数

使用 q 或 fq 查询参数可以为搜索提供字符串值。q 和 fq 查询参数是互斥的。请对字段搜索使用 fq 参数，并对多个条件使用 AND 逻辑运算符。

下表列出了可以在 URL 中使用的参数：

参数	说明
q	指定字符串值或搜索词。查询会搜索在记录中任何位置出现的搜索词。可以在简单搜索中使用。 例如，**Person?q=STEVE** 查询将搜索包含词语 STEVE 的记录。 要同时搜索两个或更多词语，请用双引号将这些词语括起来。请在您希望包含于搜索结果中的每个词语之前添加字符 +。如果字段值包含空格，请用单引号将字段值括起来。 使用以下查询可搜索 WILLIAM JOHN LAWSON 的完全匹配项： **Person?q="WILLIAM JOHN LAWSON"** 使用以下查询可搜索 WILLIAM、JOHN 或 LAWSON： **Person?q=WILLIAM JOHN LAWSON** 使用以下查询可搜索 WILLIAM、JOHN 和 LAWSON： **Person?q=WILLIAM JOHN LAWSON&queryOperator=AND**
fq	指定特定字段中的字符串值或搜索词。查询只会在记录的该部分搜索相关词语。可以对已编制索引的字段执行有针对性的搜索。 例如，**Person?fq=displayname=STEVE** 查询搜索显示名称为 STEVE 的记录。
facets	指定应视为构面或类别的字段，搜索结果将按这些构面或类别进行分组。请仅指定可搜索字段。与 q 和 fq 参数结合使用。语法为 **&facets=FieldName1,FieldName2,FieldNameN** 例如，**Person?q=STEVE&facets=department** 查询将搜索具有显示名称 STEVE 的人员并按部门对搜索结果进行分组。此搜索将显示具有显示名称 STEVE 的人员记录，而且这些记录会按部门进行分组。
筛选器	指定可用于缩小搜索结果范围的字段。请仅指定可筛选字段。与 q 和 fq 参数结合使用。 例如，**Person?fq=STEVE&filters=birthdate='1980-11-27T08:00:00Z'** 查询将搜索具有显示名称 STEVE 的人员并按出生日期筛选搜索结果。此搜索将显示具有显示名称 STEVE 且出生日期为 1980 年 11 月 27 日的人员记录。 **注意:** 请用单引号将指定的日期括起来。
depth	指定返回的子级数。指定 2 将返回根节点及其直接子代；指定 3 将返回根节点及其直接子代和孙代。指定 1 将仅返回根节点。默认情况下未指定任何深度。 如果未指定任何深度，搜索结果将返回根节点以及发现搜索词匹配项的子节点。 例如，**Person?q=STEVE&depth=2** 查询将搜索包含词语 STEVE 的记录并返回有关根记录及其直接子代的信息。
queryOperator	指定搜索是查找搜索词中的任意字符串还是所有字符串。 该参数使用以下值之一： - **OR**。搜索 **f** 或 **fq** 参数中列出的任意字符串。 - **AND**。搜索 **f** 或 **fq** 参数中列出的所有字符串。 如果不指定该参数，则默认值为 **OR**。 例如，**Person?q=WILLIAM JOHN LAWSON&queryOperator=AND** 查询搜索包含 WILLIAM、JOHN 和 LAWSON 的记录。

参数	说明
suppressLinks	指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。默认值为 false。 例如，**Person?q=STEVE&suppressLinks=true** 查询将搜索包含词语 STEVE 的记录并返回没有任何可见父子链接的响应。
readSystemFields	指示是否在结果中返回系统字段。默认值为 false。
order	带有可选前缀 + 或 - 的字段名称的逗号分隔列表。前缀 + 表示按升序对结果排序，前缀 - 表示按降序对结果排序。默认值为 +。 如果想使用子字段来对结果进行排序，请使用字段的全名。例如，**BillAddresses.Address.cityName**。 如果指定多个参数，则结果集将先按列表中的第一个参数进行排序，接着按第二个参数进行排序，依此类推。例如，**Person?order=displayName,-BillAddresses.Address.cityName** 查询会先按显示名称以升序对结果排序，再按城市名称以降序对结果排序。
maxRecordsToSort	要排序的搜索结果的最大数量。默认值为 1000。

使用 filters 参数指定范围：
您可以使用 filters 参数将搜索结果缩小至指定的范围。可以为数值和日期数据类型的可筛选字段指定范围。

请对整数数据类型使用以下格式：

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

范围介于 fromValue 到 toValue 之间。请确保 fromValue 低于 toValue。 例如，filters=age=[35,45] 查询会缩小搜索结果范围并搜索 35 到 45 岁年龄组中的记录。

请对日期数据类型使用以下格式：

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

范围介于 fromDate 到 toDate 之间。例如，filters=birthdate=[2000-06-12T12:30:00Z,2015-06-12T12:30:00Z] 查询指定了介于 2000 年 6 月 12 日到 2015 年 6 月 12 日之间的出生日期。

注意: 指定完全匹配日期筛选器时，请用单引号将其括起来。指定日期范围时，请勿使用引号。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

包含 q 参数的请求

以下示例请求将在 Person 业务实体中搜索名字为 STEVE 的记录。

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

包含 fq 参数的请求

以下示例请求将在 Person 业务实体中搜索具有显示名称 STEVE 的记录。displayName 字段是已编制索引的字段。

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

具有排序选项的请求

以下示例请求在“人员”业务实体中搜索具有显示名称 STEVE 的记录，并按照城市以升序对结果进行排序：

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

包含 fq 参数和 AND 逻辑运算符的请求

以下示例请求将在 Person 业务实体中搜索具有显示名称 STEVE 和税号 DM106 的记录：

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

包含构面的请求

以下示例请求将在 Person 业务实体中搜索具有显示名称 STEVE 的记录，并通过将这些记录分组为部门来缩小结果范围：

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

包含筛选器（精确筛选器）的请求

以下示例请求将在 Person 业务实体中搜索具有显示名称 STEVE 的记录，并按指定的城市和国家/地区进行筛选：

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

包含筛选范围的请求

以下示例请求将在 Person 业务实体中搜索具有显示名称 STEVE 的记录，并按 35 到 45 岁年龄组进行筛选：

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

示例 API 响应

以下示例响应显示了按名字 STEVE 搜索的结果：


```
{
    "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"
        }
    }
]
}
```


建议者

建议者 REST API 可根据数据库中存在的数据，返回搜索字符串的相关术语列表。使用此 API 可接受在用户界面字段中键入的字符并返回用于自动完成键入内容的建议。可以从建议列表中查找并选择字符串。建议者 API 适用于可搜索字段。

此 API 使用 GET 方法。

注意: 要使用此 API 为可搜索字段提供自动完成建议列表，请将该字段的 suggester 属性设置为 true 并为数据重新编制索引。

请求 URL

建议者 REST URL 使用以下格式：

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

向 URL 发出以下 HTTP GET 请求：

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

注意: 如果使用 Solr 搜索引擎，则每次重新启动进程服务器时，建议者索引都必须进行重建。 为确保每次重新启动进程服务器时都会重建该索引，请将 buildIndex 参数设置为 true：

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

查询参数

下表列出了查询参数：

参数	说明
suggest	必需。指定要为其获取建议的字符串。
recordsToReturn	指定返回的行数。
buildIndex	可选。指示是否构建索引。重新启动系统后，将此参数设置为 true 可显式构建集合的索引。将来的版本可能会弃用此参数。

示例 API 请求

以下示例请求将返回可在用户界面中使用的建议的列表：

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

示例 API 响应

以下示例响应显示了建议列表：

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

}
```

SearchQuery

SearchQuery REST API 搜索与在查询中指定的字段值完全匹配的记录。您可以使用 SearchQuery API 检索指定业务实体的所有记录或基于指定字段值检索记录。与在所有可搜索字段搜索指定值的搜索记录 API 不同，SearchQuery API 只在指定字段内搜索指定值。

您可以筛选查询结果，只显示根业务实体记录和子记录中特定的值。在查询中可以使用 AND、IN 和 RANGE 运算符。

要在查询结果中包含指定字段，请指定字段或指定包含这些字段的业务实体视图。可以对查询结果进行排序，以便按升序或降序查看记录。

此 API 使用 GET 方法。

请求 URL

SearchQuery URL 的上下文为 cmx/cs。在 Hosted MDM 环境中，将租户名称包括在上下文中，例如 <租户名称>/cmx/cs。

SearchQuery URL 使用以下格式：

返回一个指定业务实体类型的所有记录的 URL

使用以下 URL 搜索指定业务实体类型的所有记录：

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

向 SearchQuery URL 发出以下 HTTP GET 请求：

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

返回匹配指定字段值的记录的所有详细信息的 URL

使用以下 URL 搜索与您指定的字段值相匹配的记录：

```
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>'
```

返回匹配指定字段值的记录的指定详细信息的 URL

使用以下 URL 搜索记录并在搜索结果中显示指定的记录字段：

```
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>
```

查询参数

将查询定义为字段-值对的列表。

下表介绍了可以在 URL 中使用的查询参数：

参数	说明
action	必需。返回查询结果中指定业务实体类型的所有记录。设置为 **query**，然后将此参数与 **filter** 参数结合使用。不使用 **filter** 参数时，此查询会搜索指定业务实体类型的所有参数。 例如，使用下列查询搜索所有 Person 业务实体记录： `Person?action=query`
filter	必需。指定一个用运算符分隔的字段-值对列表。有效运算符为 AND、IN 和 Range。 例如，使用以下查询搜索名字为 STEVE 且姓氏为 SMITH 的 Person 记录： `Person?action=query&filter=firstName='STEVE' AND lastName='SMITH'`
depth	指定返回的子记录级数。例如，您可以指定以下层级： - 1. 返回根记录。 - 2. 返回根记录及其直接子记录。 - 3. 返回根记录、其直接子记录和孙记录。 例如，使用以下查询搜索名字为 STEVE 的记录，并返回有关根记录及其直接子记录的信息： `Person?action=query&filter=firstName='STEVE' AND lastName='SMITH'&depth=2`
suppressLinks	指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。默认值为 false。 例如，使用以下查询搜索名字为 STEVE 的记录，并在不存在可见父子链接时返回响应： `Person?action=query&filter=firstName='STEVE'&suppressLinks=true`
readSystemFields	指示是否在结果中返回系统字段。默认值为 false。
fields	指定要在查询结果中显示的字段。
outputView	指定要用于显示查询结果的业务实体视图。为查询结果配置业务实体视图时，应当包括要在查询结果中显示的字段。
Order	指定查询结果的排序顺序。使用加号字符 (+) 作为前缀可指定升序，而使用减号字符 (-) 作为前缀可指定降序。查询结果默认为升序。 如果指定多个参数，则结果集将先按列表中的第一个参数进行排序，接着按第二个参数进行排序，依此类推。

在筛选参数中可以使用以下运算符：

AND

搜索包含筛选参数中列出的所有字段值的记录。

例如，使用以下查询搜索名字为 STEVE 且姓氏为 SMITH 的记录：

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

IN

搜索包含筛选参数中列出的任何字段值的记录。

例如，使用以下查询搜索名字为 STEVE 或 JOHN 的记录：

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

Range

搜索指定范围内的记录。可以为数字和日期数据类型的字段指定范围。

请对整数数据类型使用以下格式：

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

范围介于 fromValue 到 toValue 之间。请确保 fromValue 低于 toValue。

例如，使用以下查询搜索属于 35 到 45 岁年龄组中的记录：

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

请对日期数据类型使用以下格式：

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

范围介于 fromDate 到 toDate 之间。

例如，使用以下查询搜索出生日期处于 2000 年 6 月 12 日 和 2015 年 6 月 12 日之间的记录：

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

示例 API 请求

以下示例请求将向 Person 业务实体查询名字为 STEVE、姓氏为 SMITH 的记录：

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

示例 API 响应

以下示例响应显示名字为 STEVE、姓氏为 SMITH 的 Person 记录查询结果。

```
{
  "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"
      }
    }
  ]
}
```

将 SearchQuery 结果导出到 CSV 文件

要将 SearchQuery 请求的结果导出到 CSV 文件，请在请求 URL 路径中将业务实体的名称指定为一个 .CSV 文件。在请求 URL 中可以使用所有查询参数。

例如，使用下列请求 URL 导出与您指定的字段值相匹配的记录的搜索结果：

```
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>'
```

示例 API 请求

以下示例请求将查询包含名字 STEVE 和姓氏 SMITH 的记录，并以 CSV 格式返回查询结果：

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

示例 API 响应

以下示例显示 CSV 格式的对名字 STEVE 和姓氏 SMITH 的查询结果：

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

SearchMatch

SearchMatch REST API 搜索模糊匹配的记录。此 API 基于在 MDM Hub 配置为匹配列的特定业务实体字段识别匹配记录。在使用 SearchMatch API 前，使用列出匹配列 API 识别业务实体的匹配列。

您可以选择指定匹配规则集，而不指定匹配列。要在查询中使用匹配规则集，必须确保已为匹配规则集启用“启用按规则搜索”选项。有关配置匹配列和匹配规则集的信息，请参阅*《Multidomain MDM 配置指南》*。

在查询中可以使用 AND、IN 和 RANGE 运算符。

要在查询结果中包含指定字段，请指定字段或指定包含这些字段的业务实体视图。可以指定排序顺序，以便在查询结果中按照升序或降序查看记录。

此 API 使用 GET 方法查询业务实体字段，并返回模糊匹配的记录及其匹配得分和相关联的匹配规则。

请求 URL

SearchMatch URL 的上下文为 cmx/cs。在 Hosted MDM 环境中，将租户名称包括在上下文中，例如 <租户名称>/cmx/cs。

SearchMatch URL 使用以下格式：

返回基于配置为匹配列的指定字段值的匹配记录的 URL

使用以下 URL 搜索与您指定的字段值相匹配的记录：

```
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>'
```

向 SearchMatch URL 发出以下 HTTP GET 请求：

```
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>'
```

返回基于匹配规则集的匹配记录的 URL

使用以下 URL 搜索基于指定匹配规则集的匹配记录：

```
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>
```

查询参数

使用 fuzzyFilter 参数指定想要查询的字段值。将 fuzzyFilter 参数与 action 参数结合使用。

下表介绍了可以在 URL 中使用的查询参数：

参数	说明
action	必需。返回指定的业务实体的匹配记录。设置为 **match**，然后将此参数与 **fuzzyFilter** 参数结合使用。 例如，使用以下查询搜索名字为 STEVE 的人： `Person?action=match&fuzzyFilter=STEVE`
fuzzyFilter	必需。指定要用于查询特定业务实体类型记录的字段名和字段值对的逗号分隔列表。 例如，使用以下查询搜索名字为 STEVE、地址为 Toronto 的记录： `Person?action=match&fuzzyFilter=firstName=STEVE,Address.Address.City=TORONTO`
matchRuleSet	指定要用于识别匹配记录的匹配规则集。 如果您没有指定的匹配规则集，请指定 NONE。将使用自动和手动合并匹配规则。
filter	指定用于筛选模糊搜索结果的字段值。 例如，使用以下查询搜索名字为 STEVE、地址为 Toronto 的记录： `Person?action=match&fuzzyFilter=firstName='STEVE',lastName="SMITH'&filter=city=Toronto`
depth	指定返回的子记录级数。例如，您可以指定以下层级： - 1. 返回根记录。 - 2. 返回根记录及其直接子记录。 - 3. 返回根记录、其直接子记录和孙记录。 例如，使用以下查询搜索名字为 STEVE 的记录，并返回有关根记录及其直接子记录的信息： `Person?action=match&fuzzyFilter=firstName='STEVE'&filter=city=Toronto`
suppressLinks	指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。默认值为 false。 例如，使用以下查询搜索名字为 STEVE 的记录，并在不存在可见父子链接时返回响应： `Person?action=match&fuzzyFilter=firstName='STEVE'&suppressLinks=true`
readSystemFields	指示是否在结果中返回系统字段。默认值为 false。
fields	指定要在查询结果中显示的字段。
outputView	指定要用于显示查询结果的业务实体视图。为查询结果配置业务实体视图时，应当包括要在查询结果中显示的字段。

在筛选参数中可以使用以下运算符：

AND

搜索包含筛选参数中列出的所有字段值的记录。

例如，使用以下查询搜索名字为 STEVE 且姓氏为 SMITH 的记录：

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

IN

搜索包含筛选参数中列出的任何字段值的记录。

例如，使用以下查询搜索名字为 STEVE 或姓氏为 JOHN、地址为多伦多或渥太华的记录：

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

Range

搜索指定范围内的记录。可以为数字和日期数据类型的字段指定范围。

请对整数数据类型使用以下格式：

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

范围介于 fromValue 到 toValue 之间。请确保 fromValue 低于 toValue。

例如，使用以下查询搜索属于 35 到 45 岁年龄组中的记录：

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

请对日期数据类型使用以下格式：

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

范围介于 fromDate 到 toDate 之间。

例如，使用以下查询搜索出生日期处于 2000 年 6 月 12 日 和 2015 年 6 月 12 日之间的记录：

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

示例 API 请求

以下示例请求将使用匹配规则集 IDL2 向 Person 业务实体查询名字为 STEVE 的记录：

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

示例 API 响应

以下示例响应显示使用匹配规则集 IDL2 查询名字为 STEVE 的记录的查询结果：

```
{
  "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": []
}
```

将 SearchMatch 结果导出到 CSV 文件

要将 SearchMatch 请求的结果导出到 CSV 文件，请在请求 URL 路径中将业务实体的名称指定为一个 .CSV 文件。在请求 URL 中可以使用所有查询参数。

例如，使用下列请求 URL 导出基于匹配规则集搜索匹配记录的结果：

```
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>
```

示例 API 请求

以下示例请求将搜索名字为 STEVE、姓氏为 SMITH 的记录，并以 CSV 格式返回查询结果：

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

示例 API 响应

以下示例显示与名字 STEVE、姓氏 SMITH 匹配的记录的 CSV 格式查询结果：

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

获取 BPM 元数据

获取 BPM 元数据 REST API 可返回任务类型和两个指示器（用来指定 BPM 工作流工具是否能够执行获取任务沿袭服务和管理服务）。

此 API 使用 GET 方法。

请求 URL

获取 BPM 元数据 URL 使用以下格式：

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

向获取 BPM 元数据 URL 发出以下 HTTP GET 请求：

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

示例 API 请求

以下示例请求将返回有关任务类型和 BPM 工作流工具的信息：

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

示例 API 响应

以下示例响应显示了任务类型和 BPM 工作流工具两个指示器的值：

```
{
    "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"
                }
            ]
        }
    }
}
```

列出任务

列出任务 API 可返回工作流任务的列表。工作流定义了业务进程中的活动以及通过活动执行的路径。每个活动称为一个任务。

此 API 使用 GET 方法返回已排序且已分页的任务列表。

请求 URL

列出任务 URL 使用以下格式：

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

向列出任务 URL 发出以下 HTTP GET 请求：

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

您可以向请求添加 HTTP 表头。

查询参数

使用任务数据字段作为查询参数可筛选任务列表。

您可以使用以下查询参数：

参数	说明
taskType	可对记录执行的一组操作。请使用名称属性指定任务类型。有关任务类型的详细信息，请参阅 *《Multidomain MDM Data Director 实施指南》*。
taskId	任务的 ID。
processId	包含任务的工作流进程的 ID。
owner	执行任务的用户。

参数	说明
title	任务的简短说明。
status	工作流中任务的状态。使用以下两个值之一： - Open：任务未开始或正在进行。 - Closed：任务已完成或已取消。
priority	任务的重要性级别。请使用以下值之一：high、normal 和 low。
creator	创建任务的用户。
createDateBefore 和 createDateAfter	日期范围。可以按 createDate 字段筛选任务。
dueDateBefore 和 dueDateAfter	日期范围。可以按 dueDate 字段筛选任务。

请在请求 URL 中使用查询参数作为名称-值对。

以下示例显示了如何使用查询参数筛选任务：

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

相关主题：

- “UTC 格式的日期和时间” 页面上 27

排序参数

要确定 REST API 响应中的排序顺序，请使用通用排序参数并提供逗号分隔的任务字段列表。您可为每个字段指定排序顺序。使用连接号 (-) 指定降序顺序。默认的排序顺序是升序。

以下示例显示了如何对结果进行排序：

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

示例 API 请求

以下示例请求将检索任务列表：

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

该请求不包含主体。

示例 API 响应

以下示例响应显示了 JSON 格式的任务列表：

```
{
    "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"
            }
        ]
    }
```


读取任务

读取任务 REST API 可返回任务的详细信息，例如，任务类型、优先级和状态。

此 API 使用 GET 方法。

请求 URL

读取任务 URL 使用以下格式：

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

注意: 使用列出任务 API 获取任务的 ID。

向读取任务 URL 发出以下 HTTP GET 请求：

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

示例 API 请求

以下示例请求将返回任务的详细信息：

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

示例 API 响应

以下示例响应显示了任务的详细信息：


```
{
    "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"
}
```

创建任务

创建任务 REST API 可创建任务并启动工作流。

此 API 使用 POST 方法创建任务并返回包含该任务的工作流进程的 ID。

请求 URL

创建任务 URL 使用以下格式：

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

向创建任务 URL 发出以下 HTTP POST 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

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

请求主体

请在创建任务时指定任务属性。使用 JSON 格式或 XML 格式在请求中发送任务数据。

下表介绍了请求主体中的任务参数：

参数	说明
taskType	可对记录执行的一组操作。请使用名称属性指定任务类型。有关任务类型的详细信息，请参阅*《Multidomain MDM Data Director 实施指南》*。
owner	创建者将任务分配给的用户。
title	任务的简短说明。
comments	任务的注释。
attachments	任务的附件。
dueDate	所有者必须完成任务的日期。
status	工作流中任务的状态。使用以下两个值之一： - Open：任务未开始或正在进行。 - Closed：任务已完成或已取消。

参数	说明
priority	任务的重要性级别。请使用以下值之一：high、normal 和 low。默认值为 normal。
creator	创建任务的用户。
createDate	任务的创建日期。
orsId	在 Hub 控制台的“数据库”工具中注册的操作引用存储 (ORS) 的 ID。
processId	ActiveVOS(R) 任务类型 ID。有关详细信息，请参阅*《Multidomain MDM Data Director 实施指南》*。
taskRecord	与任务关联的业务对象根记录或交叉引用记录。使用行 ID 或源系统和源键可以指定记录。
businessEntity	taskRecord 所属的业务实体的名称。
interactionId	交互 ID。使用交互 ID 来维持任务与记录之间的任务上下文关系。
groups	为指定用户组中的所有用户分配任务。您可以在 MDM Hub 控制台中定义用户组。可以把组指定为数列。

以下示例代码使用 rowId 指定 taskRecord：

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

请求主体使用以下格式：

```
{
     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.
         }
```

```
        }
      }]
}
```

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例请求将为根记录创建任务：

```
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"
         }
       }
     }]
}
```

示例 API 响应

以下示例显示了成功创建任务后的响应。此 API 将返回包含该任务的工作流进程的 ID。

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

更新任务

更新任务 REST API 可更新单个任务。

此 API 使用 PATCH 方法更新部分任务字段，并使用 PUT 方法更新整个任务。需提供任务的 ID 作为 URL 参数。

请求 URL

更新任务 URL 使用以下格式：

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

注意: 使用列出任务 API 获取任务的 ID。

向更新任务 URL 发出以下 HTTP PUT 请求以更新整个任务：

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

向更新任务 URL 发出以下 HTTP PATCH 请求以更新部分任务字段：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

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

请求主体

您可以使用读取任务 API 获取任务的详细信息。请在更新任务时指定任务属性。使用 JSON 格式或 XML 格式在请求中发送要更新的数据。

下表介绍了请求主体中的任务参数：

参数	说明
taskType	可对记录执行的一组操作。请使用名称属性指定任务类型。有关任务类型的详细信息，请参阅*《Multidomain MDM Data Director 实施指南》*。
taskId	任务的 ID。
owner	执行任务的用户。
title	任务的简短说明。
comments	任务的注释。
attachments	任务的附件。
dueDate	所有者必须完成任务的日期。
status	工作流中任务的状态。使用以下两个值之一： - Open：任务未开始或正在进行。 - Closed：任务已完成或已取消。
priority	任务的重要性级别。请使用以下值之一：high、normal 和 low。默认值为 normal。
creator	创建任务的用户。
createDate	任务的创建日期。
updatedBy	更新任务的用户。
lastUpdateDate	任务的上次更新日期。
orsId	在 Hub 控制台的“数据库”工具中注册的 ORS 的 ID。
processId	包含任务的工作流进程的 ID。
taskRecord	与任务关联的根记录或交叉引用记录。使用行 ID 或源系统和源键可以指定记录。
businessEntity name	taskRecord 所属的业务实体的名称。

以下示例代码使用 rowId 指定 taskRecord：

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

对于 PATCH 请求，请求主体包含要更改的任务字段。您可以更改任务的标题、优先级、到期日期和所有者。

对于 PUT 请求，请求主体包含所有任务字段。请将以下请求主体用于 PUT 请求：

```
{
     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: ''
         }
       }
     }]
}
```

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例 PUT 请求将更新整个任务：

```
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: ''
            }
          }
        }]
}
```

以下示例 PATCH 请求将更新部分任务字段：

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

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

示例 API 响应

此 API 将在成功更新任务后返回“200 OK”响应代码。响应主体为空。

任务完成

任务完成 REST API 将在完成工作流中的所有任务后结束任务工作流。在处理所有与任务相关的记录后，您可以使用此 API 结束工作流。例如，如果选择合并候选项，则您可以创建启动合并工作流的任务。预览每个候选项并合并该候选项或将该候选项标记为非匹配项后，合并任务便已完成。使用此 API 可以结束合并工作流。

此 API 使用 PUT 方法。

请求 URL

任务完成 URL 使用以下格式：

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

向任务完成 URL 发出以下 HTTP PUT 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

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

请求主体

请在请求主体中发送任务详细信息。您可以使用读取任务 API 获取任务的详细信息。

下表介绍了请求主体中的任务参数：

参数	说明
taskType	可对记录执行的一组操作。请使用名称属性指定任务类型。有关任务类型的详细信息，请参阅*《Multidomain MDM Data Director 实施指南》*。
taskId	任务的 ID。
owner	执行任务的用户。
title	任务的简短说明。
comments	任务的注释。
dueDate	所有者必须完成任务的日期。
status	工作流中任务的状态。使用以下两个值之一： - Open：任务未开始或正在进行。 - Closed：任务已完成或已取消。
priority	任务的重要性级别。
creator	创建任务的用户。
createDate	任务的创建日期。
updatedBy	更新任务的用户。
lastUpdateDate	任务的上次更新日期。
orsId	在 Hub 控制台的“数据库”工具中注册的 ORS 的 ID。
processId	包含任务的工作流进程的 ID。
taskRecord	与任务关联的根记录或交叉引用记录。使用行 ID 或源系统和源键可以指定记录。
businessEntity name	taskRecord 所属的业务实体的名称。

以下示例代码使用 rowId 指定 taskRecord：

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

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例请求将完成合并工作流：

```
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": ""
         }
       }
     }]
}
```

示例 API 响应

此 API 将在成功完成任务工作流后返回“200 OK”响应。响应主体为空。

执行任务操作

执行任务操作 REST API 可将任务重新设置为工作流以进行进一步处理。每个任务类型具有一组任务操作和一个定义了任务顺序的工作流。执行某个任务操作后，任务将移到工作流中的下一个步骤。如果某个任务操作没有后续任务，则执行该任务操作后，工作流将终止。

此 API 使用 POST 方法执行操作，例如，批准、提升或取消任务。

请求 URL

以下 URL 指定了执行任务操作 URL 的格式：

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

注意: 使用列出任务 API 获取任务的 ID。

向执行任务操作 URL 发出以下 HTTP POST 请求：

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

如果要在执行任务操作之前编辑任务，请添加 Content-Type 表头指定请求数据的媒体类型：

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

请求主体

如果要在执行任务操作之前更改任务详细信息，请在请求主体中提供任务数据。

下表介绍了请求主体中的参数：

参数	说明
taskType	可对记录执行的一组操作。请使用名称属性指定任务类型。有关任务类型的详细信息，请参阅*《Multidomain MDM Data Director 实施指南》*。
taskId	任务的 ID。
owner	执行任务的用户。
title	任务的简短说明。
comments	任务的注释。
attachments	任务的附件。
dueDate	所有者必须完成任务的日期。
status	工作流中任务的状态。使用以下两个值之一： - Open：任务未开始或正在进行。 - Closed：任务已完成或已取消。
priority	任务的重要性级别。请使用以下值之一：high、normal 和 low。
creator	创建任务的用户。
createDate	任务的创建日期。
updatedBy	更新任务的用户。
lastUpdateDate	任务的上次更新日期。
businessEntity	业务实体的名称。
orsId	在 Hub 控制台的“数据库”工具中注册的 ORS 的 ID。
processId	包含任务的工作流进程的 ID。
taskRecord	与任务关联的根记录或交叉引用记录。使用行 ID 或源系统和源键可以指定记录。
businessEntity name	taskRecord 所属的业务实体的名称。

以下示例代码使用 rowId 指定 taskRecord：

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

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例请求将取消任务并结束工作流：

```
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: ''
         }
       }
     }]
}
```

示例 API 响应

此 API 将在成功执行任务操作后返回“200 OK”响应代码。响应主体为空。

列出可分配用户

列出可分配用户 REST API 可返回可以向其分配任务（属于某个任务类型）的用户的列表。使用此 API 可以获取任务的目标用户。

此 API 使用 GET 方法。

请求 URL

列出可分配用户 URL 使用以下格式：

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

向列出可分配用户 URL 发出以下 HTTP GET 请求：

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

查询参数

下表列出了 URL 中的必需参数：

参数	说明
taskType	可对记录执行的一组操作。任务类型包括更新并批准、更新并可选择批准、合并、取消合并、审阅但不批准、最终审阅和更新已拒绝记录。
businessEntity	业务实体的名称。

示例 API 请求

以下示例请求将检索可分配用户的列表：

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

示例 API 响应

以下示例响应显示了任务类型 ReviewNoApprove 的可分配用户的列表：

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

列出文件元数据

列出文件元数据 REST API 返回存储中的文件元数据列表。

列出文件元数据 REST API 适用于 BPM 或 TEMP 存储。

此 API 使用 GET 方法。

请求 URL

列出文件元数据 URL 使用以下格式：

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

向列出文件元数据 URL 发出以下 HTTP GET 请求：

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

示例 API 请求

以下示例请求检索了 TEMP 存储中的文件元数据列表：

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

示例 API 响应

以下示例响应显示了文件元数据列表：

```
{
    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” ,
        },
        ...
    ]
}
```

创建文件元数据

创建文件元数据 REST API 可创建文件的元数据，并返回该文件的文件 ID。

可以使用文件 ID 来上载、附加、更新、下载和删除文件。

创建文件元数据 REST API 适用于 DB 或 TEMP 存储。

此 API 使用 POST 方法。

请求 URL

创建文件元数据 URL 使用以下格式：

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

向创建文件元数据 URL 发出以下 HTTP POST 请求：

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

请求主体

指定文件的元数据。

下表介绍了请求主体中的文件元数据参数：

参数	说明
fileName	文件的名称。例如 **file.txt**。
fileType	文件类型的类别。例如 **text** 或 **image**。
fileContentType	文件的内容类型。内容类型包含以 / 分隔的类型和子类型。例如 **image/png**。

注意: 创建文件元数据 REST API 请求所需的参数特定于存储。

示例 API 请求

以下示例请求为 TEMP 存储中的文件创建了元数据。

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

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

示例 API 响应

以下示例显示了在 TEMP 存储中成功创建文件元数据后的响应。此 API 返回了文件的文件 ID。

```
TEMP_SVR1.1VDVS
```

注意: 文件 ID 的格式为 <storage type>_<uniqueID>。

获取文件元数据

获取文件元数据 REST API 会返回与某个文件 ID 关联的文件的元数据。

获取文件元数据 REST API 适用于 BPM、BUNDLE、DB 或 TEMP 存储。

此 API 使用 GET 方法。

请求 URL

获取文件元数据 URL 使用以下格式：

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

向获取文件元数据 URL 发出以下 HTTP GET 请求：

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

示例 API 请求

以下示例请求返回了 TEMP 存储中文件 ID 为 TEMP_SVR1.1VDVS 的文件元数据：

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS
```

以下示例请求返回了 BUNDLE 存储中文件 ID 为 besMetadata 的资源包文件元数据：

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

示例 API 响应

以下示例响应显示了 TEMP 存储中文件 ID 为 TEMP_SVR1.1VDVS 的文件元数据：

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

以下示例响应显示了 BUNDLE 存储中资源包文件的元数据 besMetadata：

```
{
    "fileName": "besMetadata.zip",
    "fileType": "BES Metadata Bundle",
    "fileContentType": "application/zip",
    "digest": "a08c5d97da7e6a780ed7c427ff14a8d2d396438cd65b654ad67424e226f64a41"
}
```

更新文件元数据

更新文件元数据 REST API 更新与文件 ID 关联的文件的元数据。

更新文件元数据 REST API 适用于 DB 或 TEMP 存储。

此 API 使用 PUT 方法。

请求 URL

更新文件元数据 URL 使用以下格式：

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

向更新文件元数据 URL 发出以下 HTTP PUT 请求：

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

示例 API 请求

以下示例请求更新了 TEMP 存储中文件 ID 为 TEMP_SVR1.1VDVS 的文件元数据：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS

{
     “fileName” :  “file2.txt” ,
     “fileType” :  “text” ,
     “fileContentType” :  “text/plain”
}
```

以下示例请求更新了 DB 存储中文件 ID 为 DB_SVR1.0JU1 的文件元数据：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU1
{
    "fileName": "Document_2.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

示例 API 响应

此 API 将在成功更新文件元数据后返回“200 OK”响应代码。响应主体为空。

上载文件内容

上载文件内容 REST API 上载与文件 ID 关联的文件的内容。

上载文件内容 REST API 适用于 BUNDLE、DB 或 TEMP 存储。

此 API 使用 PUT 方法。

请求 URL

上载文件内容 URL 使用以下格式：

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

向上载文件内容 URL 发出以下 HTTP PUT 请求：

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

示例 API 请求

以下示例请求将文件 ID 为 TEMP_SVR1.1VDVS 的文件内容上载到了 TEMP 存储：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS/content

Test attachment content: file 1
```

以下示例请求将文件 ID 为 DB_SVR1.0JU1 的文件内容上载到了 DB 存储：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU1/content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

以下示例请求将文件 ID 为 besMetadata 的资源包文件内容上载到了 BUNDLE 存储：

```
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
```

示例 API 响应

此 API 将在成功上载文件内容后返回“200 OK”响应代码。响应主体为空。

获取文件内容

获取文件内容 REST API 会返回与某个文件 ID 关联的文件的内容。

获取文件内容 REST API 适用于 BPM、BUNDLE、DB 或 TEMP 存储。

此 API 使用 GET 方法。

请求 URL

获取文件内容 URL 使用以下格式：

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

向获取文件内容 URL 发出以下 HTTP GET 请求：

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

示例 API 请求

以下示例请求返回了 BPM 存储中文件 ID 为 urn:b4p2:22203::file1.txt 的文件内容：

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BPM/urn:b4p2:22203::file1.txt/content
```

注意: 使用读取任务 REST API 检索 BPM 存储中的任务附件的文件 ID。

以下示例请求返回了 DB 存储中文件 ID 为 DB_SVR1.0JU1 的文件内容：

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU1/content
```

注意: 使用读取记录 REST API 检索附加到记录的文件的文件 ID。

以下示例请求返回了 BUNDLE 存储中文件 ID 为 besMetadata 的资源包文件内容：

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/content
```

示例 API 响应

以下示例响应显示了 BPM 存储中 TXT 文件的内容：

```
Test attachment content: file 1
```

以下示例响应显示了 BUNDLE 存储中资源包文件的内容：

```
Content-Disposition → attachment; filename=besMetadata.zip
Content-Type → application/octet-stream
Transfer-Encoding → chunked
```

删除文件

删除文件 REST API 会删除与某个文件 ID 关联的文件，包括文件元数据和内容。

删除文件 REST API 适用于 BUNDLE、DB 或 TEMP 存储。

此 API 使用 DELETE 方法。

请求 URL

删除文件 URL 使用以下格式：

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

向删除文件 URL 发出以下 HTTP DELETE 请求：

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

示例 API 请求

以下示例请求删除了 TEMP 存储中与文件 ID TEMP_SVR1.1VDVS 关联的文件，包括文件元数据和内容：

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS
```

以下示例请求删除了 DB 存储中与文件 ID DB_SVR1.0JU1 关联的文件，包括文件元数据和内容：

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU1
```

以下示例请求删除了 BUNDLE 存储中与文件 ID besMetadata 关联的资源包文件，包括文件元数据和内容：

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata
```

示例 API 响应

此 API 将在成功删除文件后返回“200 OK”响应代码。响应主体为空。

预览升级

升级挂起的更改时，预览升级 REST API 可返回生成的记录的预览。

此 API 使用 GET 方法。将挂起的更改应用到某个记录时，您可以查看该记录的模样。此 API 响应包含具有新值的记录和具有旧值的更改摘要。此 API 不会返回有关删除的数据的信息。请在 URL 中提供挂起的更改的交互 ID。

请求 URL

预览升级 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID><business entity>/<rowId>?
action=previewPromote&interactionID=<interaction ID>
```

向预览升级 URL 发出以下 HTTP GET 请求：

```
GET http://<host>:<port>/<context>/<database ID><business entity>/<rowId>?
action=previewPromote&interactionID=<interaction ID>
```

查询参数

挂起的更改的交互 ID 是 URL 中的必需参数。

下表列出了查询参数：

参数	说明
contentMetadata	合并预览的元数据。请提供逗号分隔的列表。 可以使用以下值： - BVT。为包含要在合并预览中使用的最可信的值的记录指定行 ID。返回有关交叉引用记录和原始记录 ID 的信息。 - MERGE。指定要合并的记录的行 ID。返回有关后代记录合并方式的信息。
interactionId	挂起的更改的交互 ID。
effectiveDate	可选。要为其预览更改的日期。请使用启用了时间轴的基础对象的参数。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例请求将创建 Person 业务实体中的根记录的预览：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?
action=previewPromote&interactionId=72300000001000
```

示例 API 响应

以下示例响应将返回具有新值的记录的预览和具有旧值的更改摘要：

```
{
    "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": "SYSO          ",
    "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"
    }
}
```

升级

升级 REST API 可基于更改请求的交互 ID 升级对记录所作的所有挂起的更改。

此 API 使用 POST 方法。请提供交互 ID 作为查询参数。

请求 URL

升级 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root record>?
action=promote&interactionId=<interaction ID>
```

向升级 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId of the root record>?
action=promote&interactionId=<interaction ID>
```

查询参数

挂起的更改的交互 ID 是必需参数。此 API 使用交互 ID 查找所有与挂起的更改相关的记录。

示例 API 请求

以下示例请求将基于更改请求的交互 ID 升级所有挂起的更改：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038246?
action=promote&interactionId=69120000294000
```

示例 API 响应

以下示例响应包含升级挂起的更改后记录的行 ID：

```
{
            Person: {
                        rowidObject: "1038246"
                }
}
```

删除挂起

删除挂起 REST API 可基于更改请求的交互 ID，删除您对记录所作的所有挂起的更改。

此 API 使用 DELETE 方法并返回记录的行 ID。

请求 URL

删除挂起 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=deletePending&interactionId=<interaction ID>
```

向删除挂起 URL 发出以下 DELETE 请求：

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=deletePending&interactionId=<interaction ID>
```

查询参数

请提供要删除的挂起的更改的交互 ID。

示例 API 请求

以下示例请求将基于更改请求的交互 ID 删除所有挂起的更改：

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/233?
action=deletePending&interactionId=72300000001000
```

示例 API 响应

以下示例响应包含删除挂起的更改后记录的行 ID：

```
{
    Person: {
        rowidObject: "233"
    }
}
```

预览合并

合并两个或更多根记录时，预览合并 REST API 可返回合并的根记录的预览。

此 API 使用 POST 方法并接受根记录列表和字段级别替代，以返回合并的记录的预览。目标记录的行 ID 是必需参数。

请求 URL

预览合并 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=previewMerge
```

以下预览合并 URL 格式指定了返回的子级数：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge&depth=2
```

向预览合并 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge
```

注意: 在请求主体中，添加键属性并指定要与目标记录合并的根记录。

要替代子记录的匹配项，请添加 contentMetadata 参数：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge&contentMetadata=MERGE/<json/xml>
```

注意: 在请求主体中，添加替代属性并指定合并替代。

要指定想随请求一起发送的数据的媒体类型，请添加 Content-Type 表头：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?
action=previewMerge
Content-Type: application/<json/xml>
```

查询参数

目标记录的行 ID 是必需参数。

您可以使用以下查询参数：

参数	说明
contentMetadata	合并预览的元数据。请提供逗号分隔的列表。 可以使用以下值： - BVT。返回有关入选交叉引用记录和原始记录 ID 的信息。 - MERGE。返回有关后代记录合并方式的信息。
depth	返回的子级数。
effectiveDate	要为其生成预览的日期。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

请求主体

在开始前，请使用读取匹配记录 API 确定可与原始根记录合并的匹配记录。把请求主体中的记录列表发送给预览合并 API。

您可以替代根记录中的字段值。例如，如果所有匹配根记录都不包括名字的正确拼写，您可以在请求主体中指定正确的名字。或者，您可以删除这些匹配记录或指定其他匹配记录。

在请求主体中使用以下属性：

属性/元素	类型	说明
keys	数组	必需。希望合并的匹配根记录的排序列表。可以通过行 ID 或通过源系统和源键的组合来标识记录。
overrides	对象	替代根记录中的字段值以及子记录的匹配项。
MERGE	对象	替代希望合并的子记录的字段值。在 **overrides** 对象内添加子记录类型，然后添加 **MERGE** 对象。

以下 JSON 代码示例标识了要与目标根记录合并的根记录：

```
{
  keys: [
    {
       rowid: "P2"
    }
  ]
}
```

以下代码显示了如何替代 Party 根记录中的一个字段以及如何替代 Telephone 子记录的合并候选项：

```
{
  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
              ]
            }
          }
        ]
      }
    }
  }
}
```

示例 API 请求

以下示例请求将返回合并的记录的预览：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245?action=previewMerge
```

```
{
  keys: [
    {
        rowid: "2478246"
    },
   {
        rowid: "2478230"
    }
  ],
  overrides: {
    Person: {
       firstName: "Charlie"
    }
  }
}
```

示例 API 响应

以下示例响应显示了合并的记录的预览：

```
{
    "Person": {
        "rowidObject": "2478245       ",
        "partyType": "Person",
        "lastName": "Smith",
        "firstName": "Charlie",
```

```
            "displayName": "ALICE SMITH"
        }
    }
```

更新挂起的合并

要保存对属于挂起的合并任务一部分的记录的更改，例如记录值替换，请使用更新挂起的合并 REST API。API 会根据记录的交互 ID 保存更改。

此 API 使用 POST 方法。

请求 URL

请求 URL 的路径组件必须包含目标记录的行 ID。

更新挂起的合并 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>
/<rowid of the root record>?
action=updatePendingMerge&interactionId=<interaction ID>
```

向更新挂起的合并 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>
/<rowid of the root record>?
action=updatePendingMerge&interactionId=<interaction ID>
```

在请求主体中，添加键并指定要与目标记录合并的根记录。还可以指定要替代其匹配项的子记录。

查询参数

下表介绍了可以在 URL 中使用的查询参数：

参数	说明
action	必需。保存对挂起的合并任务的更改（如字段级替代）。设置为 **updatePendingMerge**，然后将此参数与 **interactionId** 参数结合使用。 例如，使用下列查询保存对挂起合并操作的 Person 业务实体记录的更改： **Person?action=updatePendingMerge&interactionId**
interactionId	必需。挂起的合并任务的交互 ID。

请求主体

使用更新挂起的合并 API 之前，使用读取匹配记录 REST API 确定可以与目标根记录合并的匹配记录。把请求主体中的记录列表发送给更新挂起的合并 API。

您可以替代根记录中的字段值。例如，如果所有匹配根记录都不包括名字的正确拼写，您可以在请求主体中指定正确的名字。或者，您可以删除这些匹配记录或指定其他匹配记录。

在请求主体中使用以下属性：

属性/元素	类型	说明
keys	数组	必需。希望合并的匹配根记录的排序列表。可以通过行 ID 或通过源系统和源键的组合来标识记录。
overrides	对象	替代根记录中的字段值以及子记录的匹配项。
MERGE	对象	替代希望合并的子记录的字段值。在 **overrides** 对象内添加子记录类型，然后添加 **MERGE** 对象。

以下 JSON 代码示例标识了要与目标根记录合并的两个根记录：

```
{
  keys: [
    {rowid: "2478246"},
    {rowid: "2478230"}
  ]
}
```

以下示例请求主体显示了如何替代 Party 根记录中的一个字段以及如何替代 Telephone 子记录的匹配记录：

```
{
  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
              ]
            }
          }
        ]
      }
    }
  }
}
```

示例 API 请求

以下示例请求将使用值 Charlie 替换目标记录中的姓名字段：

```
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"
    }
  }
}
```

示例 API 响应

以下示例响应包含挂起合并操作的目标记录的行 ID：

```
{
    "Person": {
        "key": {
            "rowid": "2478245"
        },
        "rowidObject": "2478245"
    }
}
```

挂起的合并

挂起的合并 REST API 可基于更改请求的交互 ID 更新您对记录进行的所有挂起的合并任务。使用“挂起的合并”，您可以将合并操作推迟到所有合并任务都获得工作流进程的批准。

此 API 使用 POST 方法并返回记录的行 ID。

请求 URL

挂起的合并 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=PendingMerge&interactionId=<interaction ID>
```

向挂起的合并 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=Merge
```

查询参数

挂起的合并的交互 ID 是必需参数。

示例 API 请求

以下示例请求将更新与交互 ID 关联的所有挂起的合并任务：

```
POST /Person/123?action=pendingMerge&interactionId=123
```

示例 API 响应

以下示例响应包含受影响的根基础对象的行 ID：

```
{
    keys: [{rowid: "456"}, {rowid: "789"}],
    overrides: {...}
}
```

升级合并

升级合并 REST API 可运行与更改请求的交互 ID 关联的所有挂起的合并任务。

此 API 使用 POST 方法并返回入选的记录的行 ID。

请求 URL

升级合并 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid>?
action=PromoteMerge&interactionId=<interaction ID>
```

向升级合并 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=Merge
```

查询参数

挂起的合并任务的交互 ID 是必需参数。此 API 使用交互 ID 来查找所有挂起的合并任务并运行合并。

示例 API 请求

以下示例请求将升级与交互 ID 关联的所有挂起的合并任务：

```
POST /Person/123?action=promoteMerge&interactionId=123
```

示例 API 响应

以下示例响应包含升级挂起的合并任务后记录的行 ID：

```
POST /Person/123?action=promoteMerge&interactionId=123
```

合并记录

合并记录 REST API 通过合并两个或更多根记录来创建单个合并的记录。该合并的记录的行 ID 是将其他记录合并到的记录的行 ID。

此 API 使用 POST 方法。您可在请求主体中为合并的记录指定字段级别替代。

请求 URL

合并记录 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=Merge
```

向合并记录 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=Merge
```

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowid of the record>?action=Merge
Content-Type: application/<json/xml>
```

查询参数

下表列出了可以在 URL 中使用的参数：

参数	说明
taskComment	向 API 触发的工作流任务添加备注。
taskAttachments	如果启用了任务附件，则向 API 触发的工作流任务附加文件。

请求主体

开始前，请使用预览合并 API 来预览选定根记录的合并结果。如果您对预览结果很满意，则在合并记录 API 的请求主体中使用相同的属性。

您可以替代根记录中的字段值。例如，如果所有匹配根记录都不包括名字的正确拼写，您可以在请求主体中指定正确的名字。或者，您可以删除这些匹配记录或指定其他匹配记录。

在请求主体中使用以下属性：

属性/元素	类型	说明
keys	数组	必需。希望合并的匹配根记录的排序列表。可以通过行 ID 或通过源系统和源键的组合来标识记录。
overrides	对象	替代根记录中的字段值以及子记录的匹配项。
MERGE	对象	替代希望合并的子记录的字段值。在 **overrides** 对象内添加子记录类型，然后添加 **MERGE** 对象。

以下 JSON 代码示例标识了要与目标根记录合并的两个根记录：

```
{
  keys: [
    {rowid: "2478246"},
    {rowid: "2478269"}
  ]
}
```

有关如何将合并记录 API 与 overrides 和 MERGE 属性结合使用的示例，请参阅合并预览 API 的请求主体。

示例 API 请求

以下示例请求将合并记录，形成合并的记录。该请求向 API 触发的工作流任务添加备注和附件。

```
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"
    }
  }
}
```

示例 API 响应

以下示例响应显示了合并的记录：

```
{
    "Person": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478245",
                "rel": "self"
            }
        ],
        "key": {
            "rowid": "2478245"
        },
        "rowidObject": "2478245"
    }
}
```

取消合并记录

取消合并记录 REST API 可将根记录取消合并为合并记录之前存在的各个根记录。

此 API 使用 POST 方法。

请求 URL

取消合并记录 URL 使用以下格式：

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

向取消合并记录 URL 发出以下 HTTP POST 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=unmerge
Content-Type: application/<json/xml>
```

查询参数

下表列出了可以在 URL 中使用的参数：

参数	说明
taskComment	向 API 触发的工作流任务添加备注。
taskAttachments	如果启用了任务附件，则向 API 触发的工作流任务附加文件。

请求主体

请在请求主体中发送要从合并的记录中取消合并的记录的列表。使用 xref 行 ID 或源系统和源键可以指定记录。

您可以使用读取记录 API 获取要取消合并的记录的 xref rowId。以下示例请求检索了记录的 XREF 元数据：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2638243?contentMetadata=XREF
```

示例 API 请求

以下示例请求将从合并的记录中取消合并某个记录。该请求向 API 触发的工作流任务添加备注和附件。

```
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"
}
```

示例 API 响应

以下示例响应显示了从合并的记录中取消合并的记录：

```
{
    "Person": {
        "link": [
            {
                "href": "http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478249",
                "rel": "self"
            }
        ],
        "key": {
            "rowid": "2478249       "
        },
        "rowidObject": "2478249       "
    }
}
```

读取关系

读取关系 REST API 会返回关系记录的详细信息，例如参与方类型、rowID 以及两个记录的显示名称。

此 API 使用 GET 方法。

请求 URL

读取关系 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID of the relationship record>
```

向 URL 发出以下 HTTP GET 请求：

```
GET http://<host>:<port>/<context>/<database ID>/<relationship>/<row ID of the relationship record>
```

查询参数

下表列出了查询参数：

参数	说明
suppressLinks	可选。指示父子链接在 API 响应中是否可见。将此参数设置为 true 可在响应中禁用所有父子链接。将此参数设置为 false 将不会在 API 响应中显示链接。默认值为 false。
depth	可选。返回的子级数。

示例 API 请求

以下示例请求返回了行 ID 为 85、关系类型为 ProductGroupProductGroupIsParentOfProductProducts 的关系记录的详细信息：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85
```

以下示例请求返回了深度为 2 的详细信息：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/
ProductGroupProductGroupIsParentOfProductProducts/85?depth=2
```

示例 API 响应

以下示例显示了行 ID 为 85 、关系类型为 ProductGroupProductGroupIsParentOfProductProducts 的关系记录的详细信息：


```
{
  "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"
    }
  }
}
```


创建关系

创建关系 REST API 会在指定记录之间创建关系。要在记录之间创建关系，记录所属的业务实体之间必须存在关系。例如，如果您想在 Informatica 和 John Smith 之间指定关系，那么组织和人员业务实体之间必须存在关系。您必须在请求主体中发送关系数据。

API 会使用 PUT 和 POST 方法。

请求 URL

创建关系 REST URL 使用以下格式：

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

注意: 源系统的名称是 URL 中的必需参数。

向 URL 发出以下 HTTP POST 或 PUT 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

```
Content-Type: application/<json/xml>
```

URL 参数

源系统的名称是请求 URL 中的必需参数。

请求主体

请在 REST 请求主体中发送关系记录的数据。使用 JSON 格式或 XML 格式发送数据。在请求主体中提供必需参数值。

示例 API 请求

以下示例请求在行 ID 为 101 的“组织”业务实体与行 ID 为 1101 的“人员”业务实体之间创建了 OrganizationEmploysPerson 关系：

```
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"
}
```

OrganizationEmploysPerson 关系定义了从“组织”业务实体到“人员”业务实体的关系。from 元素指定了关系源自哪个记录，而 to 元素指定了关系结束于哪个记录。

示例 API 响应

以下示例响应显示了在行 ID 为 101 的“组织”业务实体与行 ID 为 1101 的“人员”业务实体之间成功创建关系后的响应表头和主体：

```
{
        "OrganizationEmployesPerson": {
        "key": {
        "rowid": "414721"
        "sourceKey": "SVR1.1E7UW"
 }-

"rowidObject": "414721"

    "from": {
                    "key": {
```

```
                                "rowid": "101 "
                                                }-
        "rowidObject": "101 "
        }-

 "to": {
                "key": {
                            "rowid": "1101 "
        }-

"rowidObject": "1101 "
                }-

    }-

}
```

更新关系

更新关系 REST API 会更新两个记录之间的关系。此 API 会更新针对关系定义的其他属性。

此 API 使用 POST 和 PUT 方法。

请求 URL

更新关系 URL 使用以下格式：

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

注意: 源系统的名称是 URL 中的必需参数。

对 URL 进行以下 HTTP POST 或 PUT 调用：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型。

请求主体

在请求主体中发送对关系记录的更新。使用 JSON 格式或 XML 格式发送数据。在请求主体中提供必需参数值。

示例 API 请求

行 ID 为 414721 的关系是行 ID 为 101 的“组织”实体与行 ID 为 1101 的“人员”实体之间的 OrganizationEmploysPerson 关系。

以下示例请求更新了行 ID 为 414721 的关系记录：

```
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"}

}
```

示例 API 响应

以下是在成功更新行 ID 为 414721 的关系后收到的响应示例：

```
{
        "OrganizationEmployesPerson": {
                "key": {
                                "rowid": "414721"
                                "sourceKey": "SVR1.1E7UW"
                                        }-
            "rowidObject": "414721"
        "from": {
            "key": {
                            "rowid": "101"
                                    }-
                "rowidObject": "101"
                }-
        "to": {
                "key": {
                            "rowid": "1101 "
                               }-
        "rowidObject": "1101 "
                       }-
            }-
}
```

删除关系

删除关系 REST API 会删除两个记录之间的关系。

此 API 使用 DELETE 方法。

请求 URL

删除关系 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/<relationship>/<rowID of the relationship record>?
systemName=<name of the source system>
```

注意: 源系统的名称是 URL 中的必需参数。

向删除 URL 发出以下 HTTP DELETE 请求：

```
DELETE http://<host>:<port>/<context>/<database ID>/<relationship>/<rowID of the relationship record>?
systemName=<name of the source system>
```

查询参数

源系统的名称是必需的 URL 参数。请使用 systemName 参数指定源系统。

示例 API 请求

以下示例请求删除了行 ID 为 414721 的关系记录：

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/OrganizationEmploysPerson/414721?systemName=SFA
```

示例 API 响应

以下示例响应显示了成功删除行 ID 为 414721 的关系记录后的响应：

```
{
"OrganizationEmployesPerson": {
        "key": {
                "rowid": "414721"
                }-
"rowidObject": "414721"
            }-
}
```

获取相关记录

获取相关记录 REST API 可基于您配置的关系返回与指定根记录相关的记录的列表。此 API 还会返回关系的详细信息。

此 API 使用 GET 方法。

请求 URL

获取相关记录 URL 使用以下格式：

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

向获取相关记录 URL 发出以下 HTTP GET 请求：

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

查询参数

您可以将查询参数附加到请求 URL。

下表列出了您可以使用的查询参数：

参数	说明
recordsToReturn	要读取的 **many** 子项的记录数量。
searchToken	提取后续的结果集页的搜索标志。
returnTotal	返回结果集中的记录数量。设置为 true 可获取结果集中的记录数量。默认值为 false。

筛选参数

您可以将参数附加到 URL 来筛选相关记录。

下表列出了您可以使用的筛选参数：

参数	说明
recordStates	要检索的记录状态的逗号分隔列表。支持的记录状态为 ACTIVE、PENDING 和 DELETED。默认值为 ACTIVE。 例如，**/Party/123?action=related&recordStates=ACTIVE,PENDING** 查询会返回活动或挂起的记录。
entityLabel	实体的标签。

参数	说明
relationshipLabel	关系的标签。
entityType	实体类型的逗号分隔列表。例如，**entityType=Person,Organization** 列表会返回 **Person** 和 **Organization** 实体类型的相关记录。
relationshipType	关系类型的逗号分隔列表。例如，**relationshipType=Employee,Employer** 列表会返回 **Employee** 和 **Employer** 关系类型的相关记录。

注意: 如果指定多个筛选条件，结果会包含所有满足 AND 条件的记录。

响应主体

响应主体包含相关记录的列表、相关记录与关系的详细信息以及搜索标志。使用搜索标志提取后续的结果页。

示例 API 请求

以下示例请求提取了针对行 ID 为 101 的“组织”业务实体配置的相关记录和关系：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Organization/101?action=related
```

示例 API 响应

以下示例响应显示了行 ID 为 101 的“组织”业务实体类型的相关记录和关系：


```
{
  "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"
      }
    ]
}
```

读取匹配记录

读取匹配记录 REST API 可返回与指定根记录匹配的记录。您可以检查记录列表，确定可与原始根记录合并的记录，还可以使用合并记录 API 合并记录。

此 API 使用 GET 方法。

请求 URL

读取匹配记录 URL 使用以下格式：

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

向读取匹配记录 URL 发出以下 HTTP GET 请求：

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

响应主体

响应主体包含与指定记录匹配的记录的数量、匹配记录的详细信息以及搜索标志。使用搜索标志提取后续的匹配结果页。

示例 API 请求

以下示例请求将搜索业务实体，查找与某个记录匹配的记录：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038245?action=matched
```

示例 API 响应

以下示例响应显示了与指定记录匹配的记录的详细信息：

```
{
    "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"
        }
    ]
}
```

更新匹配记录

更新匹配记录 REST API 可在匹配表中创建或更新记录。对业务实体运行匹配进程后，匹配表包含业务实体中的匹配记录对。使用此 API 可以添加符合与指定记录合并的条件的记录。

此 API 使用 PUT 方法。

请求 URL

更新匹配记录 URL 使用以下格式：

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

向更新匹配记录 URL 发出以下 HTTP PUT 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

```
PUT http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
Content-Type: application/<json/xml>
```

请求主体

请在请求主体中发送与指定记录匹配的记录的列表。使用行 ID 或源系统和源键可以指定记录。

示例 API 请求

以下示例将在匹配表中添加记录：

```
PUT http://localhost:8080/cmx/cs/localhost-ORCL-DS_UI1/Person/1038245?action=matched
{
    keys: [
       {
          rowid: "1038246"
       }
    ]
}
```

示例 API 响应

此 API 将在成功在匹配表中创建记录后返回“200 OK”响应。响应主体为空。

删除匹配记录

删除匹配记录 REST API 可从匹配表中删除与根记录关联的匹配记录。

此 API 使用 DELETE 方法。

请求 URL

删除匹配记录 URL 使用以下格式：

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

向删除匹配记录 URL 发出以下 HTTP DELETE 请求：

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

请添加 Content-Type 表头，指定要随请求一起发送的数据的媒体类型：

```
DELETE http://<host>:<port>/<context>/<database ID>/<business entity>/<rowId>?action=matched
Content-Type: application/<json/xml>
```

请求主体

请在请求主体中发送要从匹配表中删除的记录的列表。

示例 API 请求

以下示例请求将从匹配表中删除与指定根记录匹配的记录：

```
DELETE http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/1038245?action=matched
```

```
{
    keys: [
       {
          rowid: "1038246"
       }
    ]
}
```

示例 API 响应

此 API 将在成功从匹配表中删除记录后返回“200 OK”响应。响应主体为空。

获取记录历史记录事件

获取记录历史记录事件 REST API 会返回与记录关联的历史记录事件或历史记录事件组列表。请在请求主体中发送记录 ID。

此 API 使用 GET 方法为每个历史记录事件组返回以下数据：

- 组的开始和结束日期
- 组中的事件数

此 API 会为每个历史记录事件返回以下数据：

- 历史记录事件 ID
- 更改日期
- 做出更改的用户
- 受更改影响的历史记录表的列表
- 受更改影响的记录节点的列表

请求 URL

获取记录历史记录事件 URL 使用以下格式：

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

向获取记录历史记录事件 URL 发出以下 HTTP GET 请求：

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

查询参数

记录的 ID 是必需参数。此 API 使用记录 ID 来查找所有相关的历史记录事件。

下表列出了查询参数：

参数	说明
startDate 和 endDate	可选。要为其检索数据的日期范围。如果指定了日期范围，响应将仅包含此范围内的事件。
granularity	可选。对历史记录事件进行分组的详细信息级别。如果指定，响应会对历史记录事件进行分组。否则，响应不会对组历史记录事件进行分组。 使用以下值之一： - YEAR - QUARTER - MONTH - WEEK - DAY - HOUR - MINUTE - AUTO
recordStates	可选。历史记录事件列表中返回的记录状态。请提供逗号分隔的列表。 可以使用以下值： - 活动 - 待定 - 已删除
contentMetadata	可选。历史记录事件列表的元数据。请提供逗号分隔的列表。 可以使用以下值： - XREF - PENDING_XREF - DELETED_XREF - HISTORY - 匹配 - BVT - 信任
children	可选。子节点名称的逗号分隔列表。如果指定，响应将包含子节点名称。
order	可选。带有可选前缀 + 或 - 的字段名称的逗号分隔列表。前缀 + 表示按升序对结果排序，前缀 - 表示按降序对结果排序。默认值为 +。如果指定多个参数，则结果集将先按列表中的第一个参数进行排序，接着按第二个参数进行排序，依此类推。
fields	可选。业务实体字段的逗号分隔列表。如果指定，响应将仅包含列出的字段。
filter	可选。筛选器表达式。
depth	可选。返回的子级数。
recordsToReturn	可选。指定返回的行数。

参数	说明
searchToken	可选。指定随上一个请求一起返回的搜索标志。
returnTotal	可选。如果设置为 **true**，则返回结果中的记录数。默认值为 **false**。
firstRecord	可选。指定结果中的第一行。
changeType	可选。指定在结果中返回的更改类型。请提供逗号分隔的列表。 可以使用以下值： - BO - XREF - BVT - MERGE - MERGE_AS_SOURCE - MERGE_AS_TARGET - UNMERGE_AS_SOURCE - UNMERGE_AS_TARGET

相关主题：

- “UTC 格式的日期和时间” 页面上 27

示例 API 请求

以下示例请求返回了一条记录自 2000 年 1 月 1 日开始的所有合并，并按年份分组：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/123?
action=listHistoryEvents&startDate=2010-01-01&granularity=YEAR&depth=2&changeType=MERGE
```

示例 API 响应

以下示例响应列出了指定记录自 2000 年 1 月 1 日开始的合并：

```
{
    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
    ]
}
```

获取事件详细信息

获取事件详细信息 REST API 会返回与记录关联的特定历史记录事件的详细信息。此 API 会返回详细信息，例如所做更改的类型以及更改前后的值。请在请求主体中发送记录 ID 和历史记录事件 ID。

此 API 使用 GET 方法。

请求 URL

获取事件详细信息 URL 使用以下格式：

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

向获取事件详细信息 URL 发出以下 HTTP GET 请求：

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

查询参数

开始之前，请使用获取记录历史记录事件 API 列出与记录关联的历史记录事件。从返回的结果中，将历史记录事件 ID 和记录 ID 用作查询参数。

下表列出了查询参数：

参数	说明
eventID	必需。获取记录历史记录事件 API 会返回历史记录事件的事件 ID。
recordStates	可选。历史记录事件列表中返回的记录状态。请提供逗号分隔的列表。 可以使用以下值： - 活动 - 待定 - 已删除
contentMetadata	可选。历史记录事件列表的元数据。请提供逗号分隔的列表。 可以使用以下值： - XREF - PENDING_XREF - DELETED_XREF - HISTORY - 匹配 - BVT - 信任
children	可选。子节点名称的逗号分隔列表。如果指定，响应将包含子节点名称。
order	可选。带有可选前缀 + 或 - 的字段名称的逗号分隔列表。前缀 + 表示按升序对结果排序，前缀 - 表示按降序对结果排序。默认值为 +。如果指定多个参数，则结果集将先按列表中的第一个参数进行排序，接着按第二个参数进行排序，依此类推。
fields	可选。业务实体字段的逗号分隔列表。如果指定，响应将仅包含列出的字段。
filter	可选。筛选器表达式。
depth	可选。返回的子级数。
recordsToReturn	可选。指定返回的行数。

参数	说明
searchToken	可选。指定随上一个请求一起返回的搜索标志。
returnTotal	可选。如果设置为 **true**，则返回结果中的记录数。默认值为 **false**。
firstRecord	可选。指定结果中的第一行。

示例 API 请求

以下示例请求返回了历史记录事件的信息：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/123?
action=getHistoryEventDetails&eventId=2016-07-14T02%3A01%3A24.529%2B0000
```

示例 API 响应

以下示例响应显示了指定事件的详细信息：

```
{
  "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"
    }
  }
}
```

获取 DaaS 元数据

获取 DaaS 元数据 REST API 返回有关 DaaS 提供程序的信息，例如名称、类型、使用的业务实体以及必填字段列表。

此 API 使用 GET 方法。

请求 URL

获取 DaaS 元数据 URL 的格式如下：

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

向获取 DaaS 元数据 URL 发出以下 HTTP GET 请求：

```
GET http://<host>:<port>/<context>/<database ID>/meta/daas/<providerName>
```

查询参数

providerName 参数是必需参数。此参数是配置的 DaaS 提供程序的名称。

示例 API 请求

以下示例请求返回了 dcp DaaS 提供程序的元数据信息：

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

以下示例请求返回了 ondemand DaaS 提供程序的元数据信息：

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

示例 API 响应

以下示例显示了 JSON 格式的 dcp DaaS 提供程序的元数据信息：

```
{
                    providerName: "dcp",
                    providerType: "READ",
                    businessEntity: "Organization",
                    systemName: "SFA",
                    requiredFields: [
                                    "dunsNumber"
                    ]
}
```

以下示例显示了 JSON 格式的 ondemand DaaS 提供程序的元数据信息：

```
{
                    providerName: "ondemand",
                    providerType: "SEARCH",
                    businessEntity: "Organization",
                    systemName: "SFA",
                    requiredFields: [
                                "displayName"
                    ]
}
```

DaaS 搜索

DaaS 搜索 REST API 会使用业务实体的某些输入字段来调用外部 DaaS 服务并将响应转换为记录列表。

此 API 使用 POST 方法。

请求 URL

DaaS 搜索 URL 的格式如下：

```
http://<host>:<port>/<context>/<database ID>/daas/search/<daas provider>
```

向 DaaS 搜索 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/daas/search/<daas provider>
```

注意: 在请求主体中，提供具有必填字段的业务实体的详细信息。

请求主体

请求主体必须包含 urn:co-ors.informatica.mdm 命名空间的 Organization 或 OrganizationView 类型的 XML 元素或 JSON 元素。

示例 API 请求

以下示例是对 DaaS 提供程序 ondemand 的请求，用于搜索显示名称为 INFA 的 organization 业务实体：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/search/ondemand
{
  "Organization": {
    "displayname": "INFA"
  }
}
```

示例 API 响应

以下示例响应显示了 DaaS 提供程序针对显示名称为 INFA 的“组织”业务实体所返回的搜索结果：

```
{
  "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 读取

DaaS 读取 REST API 会使用业务实体的某些字段来请求外部 DaaS 服务，并将响应转换为完整记录。

此 API 使用 POST 方法。指定请求 DaaS 提供程序时的必填字段。

请求 URL

DaaS 读取 URL 的格式如下：

```
http://<host>:<port>/<context>/<database ID>/daas/read/<daas provider>
```

向该 URL 发出以下 POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/daas/read/<daas provider>
```

注意: 在请求主体中，提供具有必填字段的记录的详细信息。

查询参数

下表列出了您可以使用的查询参数：

参数	说明
logChanges	可选。如果设置为 true，生成的记录将包括传递给写入业务实体服务的服务数据对象 (SDO) 更改摘要。默认值为 false。

请求主体

请求主体必须包含 urn:coors.informatica.mdm 命名空间的 OrganizationView 类型的 XML 元素或 JSON 元素。

示例 API 请求

以下示例是对 DaaS 提供程序 ondemand 的请求，用于搜索显示名称为 INFA 的 organization 业务实体：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/daas/search/ondemand
{
  "Organization": {
    "displayname": "INFA"
  }
}
```

示例 API 响应

以下示例响应显示了 DaaS 提供程序针对 D-U-N-S 数字为 001352574 的组织返回的详细信息：

```
{
  "Organization": {
    "displayName": "Infa Lab Inc",
    "dunsNumber": "001352574",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "phoneNum": "09736250550"
        }
      ]
    },
    "Addresses": {
      "link": [],
      "item": [
        {
          "Address": {
            "cityName": "Rockaway",
            "addressLine1": "11 WALL ST"
          }
        }
      ]
    }
  }
}
```

WriteMerge

WriteMerge REST API 接受从 DaaS 提供程序中检索的记录列表，将它们保存在具有相应源系统的单独 XREF 中，然后将它们合并到单个记录中。所有 XREF 都属于同一个记录。

此 API 使用 POST 方法。

请求 URL

WriteMerge URL 的格式如下：

```
http://<host>:<port>/<context>/<database ID>/daas/write-merge/<business entity name>
```

向 WriteMerge URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/daas/write-merge/<business entity name>
```

请求主体

请求主体应该包含 urn:cobase.informatica.mdm 命名空间的 DaaSEntity.Pager 类型的 XML 或 JSON 元素。

响应表头

响应成功时，此 API 会在响应表头中返回 interactionId 和 processId。

示例 API 请求

以下示例请求将 DaaS 搜索的结果用作输入来创建“组织”业务实体类型的记录：

```
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"
    }
  ]
}
```

示例 API 响应

以下示例响应显示了在将记录列表成功合并到单个记录后的响应表头和主体：

```
{
  "Organization": {
    "key": {
      "rowid": "121921",
      "sourceKey": "140000000000"
    },
    "rowidObject": "121921",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "key": {
            "rowid": "21721",
            "sourceKey": "140000001000"
          },
          "rowidObject": "21721"
        }
      ]
    }
  }
}
```

DaaS 导入

DaaS 导入 REST API 会接受 XML 格式的数据并将其转换为记录。

此 API 使用 POST 方法。

请求 URL

导入 DaaS 数据 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/daas/import/FamilyTreeMemberOrganizationToOrgView
```

向导入 DaaS 数据 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/daas/import/FamilyTreeMemberOrganizationToOrgView
```

查询参数

源系统的名称是必需参数。

下表列出了可以在请求中使用的参数：

参数	说明
systemName	必需。执行数据更改的源系统的名称。
interactionId	可选。要分配给所有更改的交互 ID。通常，Hub 会在因调用而创建挂起的更改时生成该 ID。
effectivePeriod	可选。包含开始日期和结束日期。指定记录保持有效的时间段。请为启用了时间轴的记录提供这些参数。
validateOnly	可选。如果设置为 TRUE，则只会将验证规则应用于经过修改的记录并且更改不会保留。
recordState	可选。已创建或已更新记录的 Hub 状态。支持的记录状态为 ACTIVE 和 PENDING。
processId	可选。包含任务的工作流进程的 ID。当由于服务调用而启动工作流时，参数会包含已启动的工作流进程的标识符。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

请求主体

请求主体必须包含来自 urn:co-ors.informatica.mdm 命名空间的 DaaSChangeFamilyTreeMemberOrganizationToOrgView 类型的 XML 或 JSON 元素。

响应表头

如果响应成功，API 将在响应表头中返回 interactionId 和 processId，并在响应主体中返回记录详细信息。

如果进程生成某个交互 ID 并使用它创建记录，API 将返回该交互 ID。如果进程启动工作流而不直接将记录保存到数据库，API 将返回进程 ID，即工作流进程的 ID。

以下示例显示了带有交互 ID 和进程 ID 的响应表头：

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

示例 API 请求

请求始终是 XML 格式。

以下示例请求显示了关联命名空间中 ChildAssociation 类型的 XML 数据：

```
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>
```

示例 API 响应

以下示例响应显示了成功导入并创建“组织”业务实体后的响应表头和实体：

```
BES-interactionId: 72202100242034
BES-processId: 156048
{
  "Organization": {
    "key": {
      "rowid": "101921",
      "sourceKey": "697557825"
    },
    "rowidObject": "101921"
  }
}
```

DaaS 更新

DaaS 更新 REST API 会接受更改前后的 XML 格式数据。该 API 会将更改应用于记录。

此 API 使用 POST 方法。

请求 URL

DaaS 更新 URL 使用以下格式：

```
http://<host>:<port>/<context>/<database ID>/daas/update/FamilyTreeMemberOrganizationToOrgView
```

向 DaaS 更新 URL 发出以下 HTTP POST 请求：

```
POST http://<host>:<port>/<context>/<database ID>/daas/update/FamilyTreeMemberOrganizationToOrgView
```

查询参数

源系统的名称是必需参数。

下表列出了可以在请求中使用的参数：

参数	说明
systemName	必需。执行数据更改的源系统的名称。
interactionId	可选。要分配给所有更改的交互 ID。通常，Hub 会在因调用而创建挂起的更改时生成该 ID。
effectivePeriod	可选。包含开始日期和结束日期。指定记录保持有效的时间段。请为启用了时间轴的记录提供这些参数。
validateOnly	可选。如果设置为 TRUE，则只会将验证规则应用于经过修改的记录并且更改不会保留。
recordState	可选。已创建或已更新记录的 Hub 状态。支持的记录状态为 ACTIVE 和 PENDING。
processId	可选。包含任务的工作流进程的 ID。如果工作流会作为服务调用的结果而启动，则需在参数中包含所启动工作流进程的标识符。

相关主题：

- “UTC 格式的日期和时间” 页面上 27

请求主体

请求主体必须包含来自 urn:co-ors.informatica.mdm 命名空间的 DaaSChangeFamilyTreeMemberOrganizationToOrgView 类型的 XML 或 JSON 元素。

响应表头

如果响应成功，API 将在响应表头中返回 interactionId 和 processId，并在响应主体中返回记录详细信息。

如果进程生成某个交互 ID 并使用它创建记录，API 将返回该交互 ID。如果进程启动工作流而不直接将记录保存到数据库，API 将返回进程 ID，即工作流进程的 ID。

以下示例显示了带有交互 ID 和进程 ID 的响应表头：

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

示例 API 请求

此 API 接受 XML 格式的两个响应，一个响应是在更改前，另一个响应是在更改后。在以下请求中，组织中添加了新电话号码。before XML 数据没有电话号码，而 after XML 数据有电话号码。

以下请求包含新添加的电话号码：

```
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>
```

示例 API 响应

以下示例显示了为组织新建的电话号码的行 ID：

```
{
  "Organization": {
    "key": {
      "rowid": "101921",
      "sourceKey": "697557825"
    },
    "rowidObject": "101921",
    "TelephoneNumbers": {
      "link": [],
      "item": [
        {
          "key": {
            "rowid": "1722",
            "sourceKey": "09736250550"
          },
          "rowidObject": "1722"
        }
      ]
    }
  }
}
```

第 4 章

简单对象访问协议业务实体服务调用

本章包括以下主题：

业务实体服务的简单对象访问协议调用

简单对象访问协议 (SOAP) 端点调用会使所有业务实体服务可用作 Web 服务。您可以执行 SOAP 调用，在业务实体中创建、删除、更新和搜索记录。您可以执行合并、取消合并和匹配记录等操作。您还可以执行 SOAP 调用，创建、更新、搜索和执行任务。

业务实体服务的 SOAP 端点使用 Web 服务安全 (WS-Security) UsernameToken 对用户进行身份验证。

注意: 使用 SOAP API 调用业务实体服务之前，请验证操作引用存储。

身份验证方法

所有对业务实体服务的 SOAP 调用都需要进行用户身份验证。请在 Web 服务请求的 SOAP 消息表头中提供用户名和密码。

SOAP 表头元素 Security 包含与安全相关的数据。Security 元素包含 UsernameToken 元素，而后者包含以下子元素：

username

与标志关联的用户名。

password

与标志关联的用户名的密码。

请在 UsernameToken 元素中发送用户名和密码。

以下示例显示了 SOAP 消息中的 Security 表头元素：

```
<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>
```

用于从第三方应用程序进行登录的身份验证 Cookie

使用身份验证 Cookie 可以对 MDM Hub 用户进行身份验证，并从第三方应用程序调用业务实体服务。您可以获取基于经过身份验证的用户的凭据的 Cookie。保存该 Cookie 后，可以用它来调用 SOAP API，而不需要对用户名和密码进行硬编码。

发出以下 POST 请求，用您的用户名和密码登录到 Entity 360 视图：

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

登录操作成功后，服务器会在 set-cookie 表头字段中返回身份验证 Cookie。以下示例代码显示了响应表头中的 set-cookie：

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

请存储该哈希值并在 API 调用的请求表头中使用该哈希值。您不需要为 API 调用提供用户名和密码。

以下示例显示了如何在 API 请求表头中使用身份验证 Cookie：

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

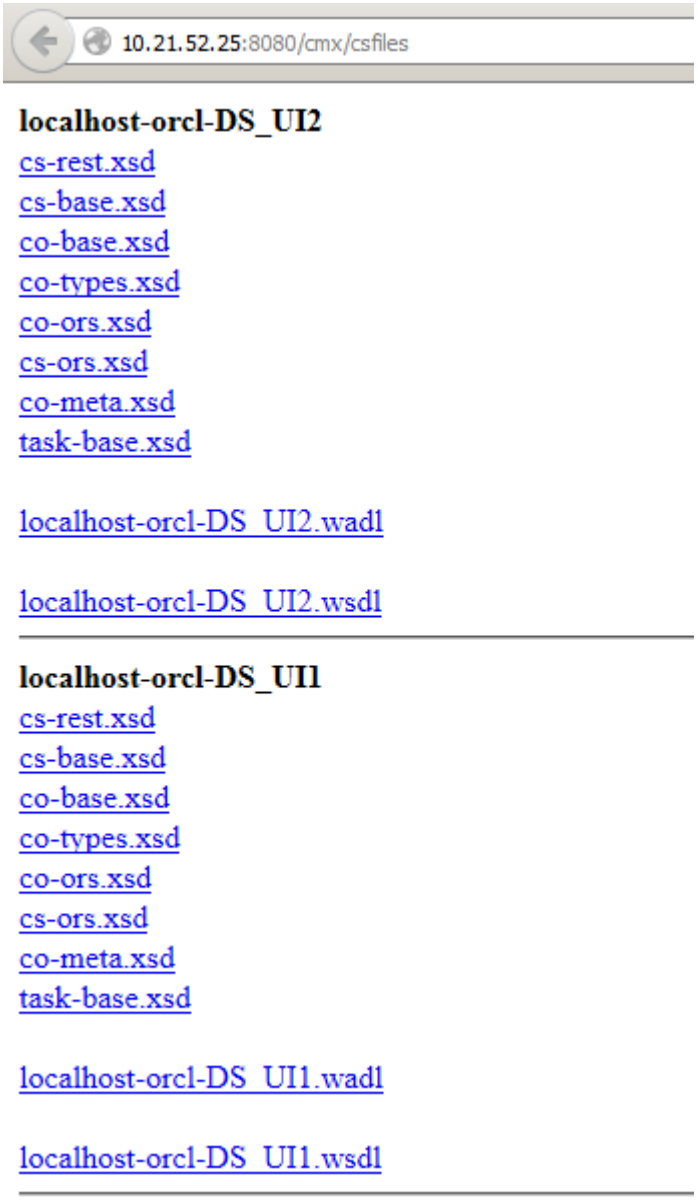
Web 服务描述语言文件

Web 服务描述语言 (WSDL) 文件包含 Web 服务的 XML 描述、SOAP 请求与响应的格式以及所有参数。MDM Hub 将为每个操作引用存储生成 WSDL 文件。

每个操作引用存储的 WSDL 文件位于以下位置：

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

下图显示了每个操作引用存储的 WSDL 文件的下载位置：



单击链接可下载 DS_UI1 或 DS_UI2 操作引用存储的 WSDL 文件。

SOAP URL

SOAP URL 是用于连接到 SOAP 服务器的地址。

SOAP URL 使用以下语法：

```
http://<host>:<端口>/<上下文>/<数据库 ID>
```

URL 包含以下字段：

host

运行数据库的主机。

port

数据库侦听器所使用的端口号。

context

上下文始终为 cmx/services/BEServices。

database ID

在 Hub 控制台的“数据库”工具中注册的 ORS 的 ID。

以下示例显示了一个 SOAP URL：

```
http://localhost:8080/cmx/services/BEServices/localhost-orcl-DS_UI1
```

SOAP 请求和响应

使用 SOAP XML 消息格式可以通过 SOAP 客户端向业务实体服务发送请求，还可以接收业务实体服务对客户端的响应。SOAP 请求和响应的格式相同。

SOAP 消息包含以下元素：

Envelope

定义消息的开头和结尾。

Header

可选。包含附加属性，例如用于处理消息的身份验证详细信息。如果存在 Header 元素，则它必须是 Envelope 元素的第一个子元素。

Body

包含客户端或 Web 服务所处理的 XML 数据。

SOAP 消息使用以下格式：

```
<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Header>
</env:Header>

<env:Body>
</env:Body>

</env:Envelope>
```

SOAP 请求使用以下格式：

```
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>
```

SOAP 响应使用以下格式：

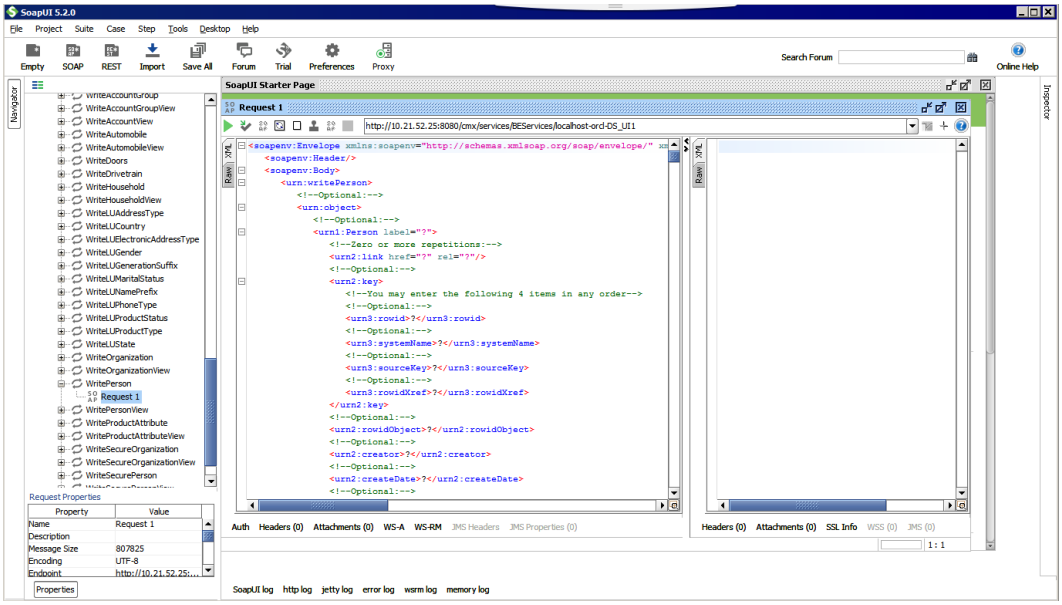
```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Header>
</env:Header>

<env:Body>
</env:Body>

</env:Envelope>
```

查看输入参数和输出参数

您可以使用诸如 SoapUI 等功能测试工具查看 SOAP API 输入参数和输出参数。

请创建一个 SOAP 项目，然后将 WSDL 文件导入到该项目。可以使用业务实体服务执行的操作将作为节点显示在 SoapUI 中。每个操作都具有请求消息和响应消息的格式。导入 WSDL 文件时，SoapUI 将为每个操作创建示例请求。

打开项目并双击某个请求可打开请求编辑器。下图显示了 SoapUI 中 WritePerson SOAP API 的输入参数：



SOAP API 参考

“业务实体服务的 SOAP API 参考”列出了 SOAP API 并提供了每个 API 的说明。有关业务实体服务的说明，也可参阅 WSDL 文件。

可以使用以下 SOAP API 对业务实体执行操作：

获取元数据

返回业务实体的数据结构。

列出记录

返回查找值或外键值的列表。

读取记录

返回业务实体中的根记录的详细信息。

创建记录

在指定的业务实体中创建记录。

更新记录

更新指定的根记录及其子记录。

删除记录

删除业务实体中的根记录。

搜索记录

在可搜索根业务实体中搜索字符串值，并返回与搜索条件匹配的根记录。

预览升级

在您基于更改请求的交互 ID 升级挂起的更改时，返回生成的记录的预览。

升级

基于更改请求的交互 ID，升级对记录所作的所有挂起的更改。

删除挂起

基于更改请求的交互 ID，删除对记录所作的所有挂起的更改。

预览合并

在您合并两个或更多根记录后返回合并的根记录的预览。

合并记录

通过合并两个或更多根记录来创建单个合并的记录。

取消合并记录

将根记录取消合并为合并记录之前存在的各个根记录。

获取相关记录

基于在层次结构管理器中配置的关系返回相关记录的列表。

读取匹配记录

返回与指定的根记录匹配的记录。

更新匹配记录

在匹配表中创建或更新记录。

删除匹配记录

从匹配表中删除匹配记录。

获取 BPM 元数据

返回任务类型和两个指示器（用来指定工作流适配器是否能够执行获取任务沿袭服务和管理服务）。

列出任务

返回工作流任务的列表。

读取任务

返回任务的详细信息。

创建任务

创建任务并启动工作流。

更新任务

更新单个任务。

执行任务操作

执行任务操作，并将任务重新设置为工作流以进行进一步处理。

列出可分配用户

返回可以向其分配任务（属于某个任务类型）的用户的列表。

任务完成

在完成工作流中的所有任务后结束任务工作流。

示例 SOAP 请求和响应

以下示例 SOAP 请求将检索可分配用户的列表：

```
<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>
```

以下示例 SOAP 响应将列出可分配用户：

```
<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>
```

第 5 章

用于交叉引用记录和 BVT 计算的服务

本章包括以下主题：

交叉引用记录和 BVT 计算服务概览

您可以使用交叉引用记录和最佳数据版本 (BVT) 计算服务来了解源数据如何形成主记录。

可以使用这些服务执行以下任务：

- 收集有关源数据的信息
- 确定最佳数据版本是如何确定的
- 覆盖 BVT 计算以确保主记录包含最佳数据版本

获取交叉引用数据并调查 BVT 计算

MDM Hub 中的主记录会维护最佳数据版本 (BVT)。MDM Hub 将多个源系统中信任得分最高的数据整合到每个主记录中，以获得最佳数据版本。MDM Hub 在交叉引用记录中存储源数据。您可以使用业务实体服务来读取交叉引用记录中的数据并确定 BVT 的计算方法。

获取交叉引用记录

您可以使用业务实体服务获取特定主记录的交叉引用记录。

获取交叉引用记录的 REST API URL 具有以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?contentMetadata=XREF
```

以下示例请求检索了行 ID 为 123 的“人员”业务实体记录的交叉引用记录：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=XREF
```

。

获取交叉引用记录响应

以下示例显示了针对行 ID 为 123 的“人员”记录返回的交叉引用记录：

```
GET /Person/123?contentMetadata=XREF

{
  "firstName": "Joe",
  "lastName": "Smith",
  "XREF": {
    "item": [
      {
        "rowidXref": 111,
        "firstName": "Joe",
        "lastName": "Smith",
      },
      {
        "rowidXref": 222,
        "firstName": "John",
        "lastName": "Smith"
      }
    ]
  }
}
```

确定主记录的提供方

您可以使用业务实体服务了解哪些交叉引用记录字段提供了主记录。每个字段的提供记录都由交叉引用记录的行 ID 标识。

用于确定主记录的提供方的 REST API URL 具有以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?contentMetadata=BVT
```

以下示例请求检索了行 ID 为 123 的“人员”记录的 BVT 信息：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=BVT
```

确定主记录响应的提供方

以下示例显示了哪些交叉引用记录提供了主记录中的每个字段：

```
{
  "firstName": "Joe",
  "lastName": "Smith",
  "BVT": {
    "firstName": {
      "rowidXref": 111
    },
    "lastName": {
      "rowidXref": 222
    }
  },
}
```

获取提供交叉引用记录字段的信任得分

您可以使用业务实体服务来获取提供主记录内容的交叉引用记录字段的信任得分。

用于确定提供方并获取信任得分的 REST API URL 具有以下格式：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?contentMetadata=TRUST
```

以下示例请求提供了行 ID 为 123 的“人员”记录的信任得分：

```
GET http://localhost:8080/cmx/cs/ors/Person/123?contentMetadata=TRUST
```

获取提供交叉引用记录字段响应的信任得分

以下响应示例提供了“人员”业务实体记录中的每个字段的信任得分：

```
{
  "firstName": "John",
  "lastName": "Smith",
  "TRUST": {
    "firstName": {
      "score": 75.0,
      "valid": true,
      "trustSetting" :{
          // custom settings
      }
    },
  },
}
```

获取所有交叉引用记录字段的信任得分

使用 REST API 并将 contentMetadata 参数设置为 XREF_TRUST 可获取所有交叉引用记录字段的信任得分和降级百分比。

用于确定提供方并获取信任得分的读取 REST API 的请求 URL：

```
http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?contentMetadata=XREF_TRUST
```

以下示例请求检索了行 ID 为 123 的“人员”记录的交叉引用数据：

```
GET http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?contentMetadata=XREF_TRUST
```

获取所有交叉引用记录字段响应的信任得分

以下示例显示了“人员”业务实体的所有交叉引用记录字段的信任得分和降级百分比：

```
{
    "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
                }
            }
        }]
    }
}
```

获取有关源系统的信息

您可以了解交叉引用数据来自哪个源系统以及源系统为整个记录、每个节点或每个记录提供了多少交叉引用记录。

以下参数可以在请求中指定：

describe

设置为 true 以返回源系统的说明。可以是 true 或 false。默认值为 false。

aggregate

定义了要返回哪个级别的源系统信息。可以是 ENTITY、NODE 和 RECORD。默认值是 ENTITY。

recordStates

指定了要返回的记录的记录状态。可以是 ACTIVE、PENDING 或 DELETED。默认值为 ACTIVE。

compact

设置为 no 时，指定当 aggregate 参数使用 ENTITY 和其他汇总级别时返回实体级别数据。可以是 yes 或 no。仅限 REST API 请求。默认值为 yes。

获取源系统信息的示例

以下示例请求针对行 ID 为 123 的“人员”业务实体获取了实体级别和节点级别的源系统信息：

```
GET http://localhost:8080/cmx/cs/ors/Person/123?action=getSourceSystems&aggregate=ENTITY,NODE&compact=no
```

以下示例请求针对行 ID 为 456 的“人员”业务实体获取了记录级别的源系统信息和源系统说明：

```
GET http://localhost:8080/cmx/cs/ors/Person/123/Address/456?
action=getSourceSystems&aggregate=ENTITY,NODE&compact=no
```

获取有关源系统响应的信息

以下示例显示了“人员”业务实体的实体级别和节点级别的信息：

```
{
      "name": "Admin",
      "xrefCount": 120
    }
  ],
  Person: {
    "rowidObject": "456",
    "sourceSystem":
      {
        "name": "Admin",
        "xrefCount": 30
      }
    ]
```

```
    }
}
```

筛选和分页响应

您可以选择要在响应中返回的字段、按照几个条件筛选结果并对结果进行分页。

筛选请求示例

下表显示了“人员”业务实体的示例请求，其中包含应用的各种筛选器以及在响应中返回的结果的说明：

请求	返回结果的说明
/Person/123	所有用户定义字段
/Person/123?readSystemFields=true	所有用户定义字段和所有系统字段
/Person/123?fields=firstName	一个用户定义字段
/Person/123?fields=updatedBy	一个系统字段
Person/123?fields=firstName,updatedBy	一个用户定义字段和一个系统字段
/Person/123?fields=firstName&readsystemFields=true	一个用户定义字段和所有系统字段

创建最佳数据版本

在数据管理者调查交叉引用记录中的源数据后，他们可以对源数据整合的方式进行调整，以确保主记录代表最佳数据版本。

您可以使用业务实体服务执行以下操作来确定最佳数据版本：

- 更新信任设置
- 移除不匹配的源数据
- 选择正确的提供字段
- 将正确的值写入主记录

选择正确的提供字段

如果具有最高信任得分的字段不包含最佳数据版本，数据管理者可选择包含正确数据的字段以将正确数据贡献给主记录。

根据系统名称和源键选择正确的提供字段的 URL 和请求主体使用以下格式：

```
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>"
        }
    }
}
```

根据交叉引用记录 ID 选择正确的提供字段的 URL 和请求主体使用以下格式：

```
POST http://<host>:<port>/<context>/<database ID>/<business entity>/<row ID>?systemName=<source system
name>
{
    BVT: {
        <field name>: {
            rowidXref: "<row ID>"
        }
    }
}
```

选择正确的提供字段示例

以下 URL 和请求主体从源键为 0001 的销售源系统中选择了交叉引用记录的名字字段，以提供主记录：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=Admin
{
    BVT: {
        firstName: {
            systemName: "Sales",
            sourceKey: "0001"
        }
    }
}
```

以下 URL 和请求主体选择了行 ID 为 789 的交叉引用记录的名字字段，以提供主记录：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-ORS/Person/123?systemName=Admin
{
    BVT: {
        firstName: {
            rowidXref: "789"
        }
    }
}
```

将正确的值写入主记录

在使用业务实体服务调用将正确的值写入主记录时，您还可以为此值建立信任设置。如果没有指定信任设置，MDM Hub 会使用管理员系统设置。

使用管理员信任设置写入正确值的 URL 和请求主体的格式如下：

```
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
            }
        }
    }
}
```

使用定义的信任设置写入正确值的 URL 和请求主体的格式如下：

```
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>"
            }
        }
    }
}
```

信任参数

您可以定义以下信任参数：

minimumTrust

数据值在变旧（衰减期已过后）时采用的信任级别。此值必须小于或等于最大信任值。

注意: 如果最大信任和最小信任相等，则衰减曲线为没有起伏的线，而且衰减期和衰减类型不会生效。

maximumTrust

数据值刚更改时具有的信任级别。 例如，如果源系统 X 将一个电话号码字段从 555-1234 更改为 555-4321，则新值将获取系统 X 中该电话号码字段的最大信任级别。通过设置相对较高的最大信任级别，可以确保源系统中的更改能够应用于基础对象。

timeUnit

指定用于计算衰减期的单位—日、周、月、季度或年。

maximumTimeUnits

指定用于计算衰减期的数值（天数、周数、月数、季度数或年数）。

graphType

在衰减期间，衰减按照信任级别递降模式进行。 图形类型可以是以下衰减类型之一：

图形类型参数	说明
LINEAR	最简单的衰减。衰减按照从最大信任到最小信任的直线模式进行。
RISL	多数下降情况均发生在衰减期的开始处。 衰减呈凹曲线趋势。 如果源系统为此图形类型，则可能会信任该系统中的新值，但此值可能将被替代。
SIRL	多数下降情况均发生在衰减期的结尾处。 衰减呈凸曲线趋势。 如果源系统为此图形类型，则直到由主记录的值靠近其衰减期的结尾处时，其他系统相对才可替代该值。

将正确的值写入主记录示例

示例 1

您想将主记录中的名称从 Sam Brown 更改为 John Smith。更改可归因于销售源系统。信任设置被设置为管理员信任设置。

以下代码显示了示例 1 的 URL 和命令。

```
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
            }
        }
    }
}
```

示例 2

您想将主记录中的名称从 Sam Brown 更改为 John Smith。更改来源于 SFA 源系统。信任设置如下：最小信任值为 60，最大信任值为 90，信任值在三个月的衰减期内呈线性衰减。

以下代码显示了示例 2 的 URL 和命令。

```
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"
            }
        }
    }
}
```

移除不匹配的源数据

如果交叉引用记录与特定的主记录关联不正确，数据管理者可以取消合并交叉引用记录。将根据取消合并的交叉引用记录创建新的主记录。

只有一个交叉引用记录可以在取消合并调用中取消合并。如果需要取消合并几个交叉引用记录，请为每个交叉引用记录执行取消合并调用。

如果触发器针对取消合并事件进行配置，则会创建取消合并任务。否则，交叉引用记录会取消合并。

根据系统名称和源键来取消合并记录的 URL 和命令具有以下格式：

```
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>" }
}
```

根据交叉引用记录 ID 来取消合并记录的 URL 和命令具有以下格式：

```
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>"}
}
```

移除不匹配的源数据示例

REST API 示例

以下代码显示了从地址记录取消合并子级别的交叉引用记录的 URL 和命令：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-MDM_SAMPLE/Person/181921?
action=unmerge&systemName=Admin
{
"name":"Person.Address",
"key":{
"rowid":"41721 ",
"rowidXref":41722
}
}
```

其中：

- 要取消合并的交叉引用记录具有行 ID 41722
- 要从中取消合并交叉引用记录的主记录的行 ID 为 41721
- 根记录的行 ID 为 181921

SOAP/EJB 示例

以下代码显示了从地址记录取消合并子级别的交叉引用记录的 URL 和命令：

```
<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>
```

其中：

- 要取消合并的交叉引用记录具有行 ID 41722
- 要从中取消合并交叉引用记录的主记录的行 ID 为 41721
- 根记录的行 ID 为 181921

取消合并响应

取消合并响应包含从取消合并的交叉引用记录中创建的基础记录的行 ID。

响应示例 1

以下示例显示了从“人员”根节点获取交叉引用记录时的响应。

```
{
    Person: {
        rowidObject: "7777"
    }
}
```

响应示例 2

以下示例显示了从“地址”子节点获取交叉引用记录时的响应。

```
{
    Person: {
        Address: {
            item: [
                rowidObject: "55555"
            ]
        }
    }
}
```

第 6 章

支持企业关联服务

本章包括以下主题：

概览

Duns & Bradstreet (D&B) 提供的企业关联服务会返回请求组织的父级及其所有相关实体。您可以使用 D&B 的关联服务，从中获取组织的所有分支和部门的信息。您可以使用关联服务提供的数据来创建和更新记录。

您可以将企业关联数据导入 MDM Hub。您必须开发自定义应用程序，以便使用实体视图中的 DaaS 提供程序自定义组件提供的关联服务。

您需要通过业务实体服务来导入 D&B 服务提供的数据并使用此数据创建记录。如果外部存储中的数据发生更改，您必须能够在记录中做出相应的更改。D&B 会提供监控服务，向您通知数据更改。您需要通过服务来接受更改前后的数据并将更改应用到相应的记录。

用于 DaaS 导入和更新的业务实体服务

DaaS 导入业务实体服务接受来自关联服务的 XML 格式数据，并将其转换为记录。DaaS 更新业务实体服务接受来自外部服务的两个 XML 文件格式数据。两个 XML 文件对应更改前后的数据。更新服务会将更改应用到相应的记录。

相关主题：

- “DaaS 导入” 页面上 127
- “DaaS 更新” 页面上 130

配置关联支持

要使用 D&B 提供的关联服务来创建并更新记录，您必须在置备工具中添加配置，然后创建自定义应用程序，以拆分来自关联服务的响应。

执行以下任务来为 D&B 的关联服务配置支持：

1. 使用置备工具为关联服务上载 WSDL。
2. 使用置备工具配置 XML 文档到业务实体的转换并将其作为服务呈现。当您将转换呈现为服务时，该进程将创建 DaaS 导入和更新业务实体服务。
3. 创建自定义应用程序，用于请求关联服务的数据并将响应拆分为记录详细信息和关联详细信息。
4. 开发一个将请求发送到自定义应用程序的用户界面。

注意: 有关如何上载 WSDL 以及配置 XML 到业务实体转换的详细信息，请参阅*《Multidomain MDM 置备工具指南》*中的“集成数据即服务”一章。

用于关联数据拆分的自定义应用程序

要使用 D&B 的关联服务，您必须设计一个能将关联信息拆分为记录详细信息和关联详细信息的自定义应用程序。

自定义应用程序必须执行以下功能：

1. 接受实体视图中对关联服务的请求。
2. 将该请求发送到 D&B 并接收响应。
3. 将响应转换为 XML。
4. 将响应拆分为记录详细信息和关联详细信息。
5. 将 XML 信息发送到业务实体服务，以便在数据库中另存为记录。
6. 监控数据更改并调用外部服务的“列表更改通知”功能。

第 7 章

清理、分析和转换数据的外部调用

本章包括以下主题：

概览

外部提供程序提供了 Web 服务来清理、分析和转换记录数据。您可以使用外部 Web 服务进行自定义验证，例如，在添加记录时检查地址字段是否为空。还可以使用外部 Web 服务来自定义转换记录数据的逻辑。例如，在合并两个记录时，可以合并地址，但不能合并电话号码。

外部 Web 服务会呈现业务实体服务可以调用的一个或多个操作。每个操作都有一个请求和响应类型。业务实体服务会将具有所需服务参数的记录数据发送到外部服务。您可以为执行逻辑中的某些步骤配置外部 Web 服务调用。根据您实施的逻辑，请求会从 Data Director 发出，以更新记录数据。如果需要，外部服务也可以修改数据。

在置备工具中，配置业务实体以及您想为其调用外部服务的事件。在置备工具中，上载用于外部服务的 WSDL 文件并注册 SOAP 服务和操作。将服务与特定业务实体和事件绑定起来。

您可以通过资源工具包中的 WSDL 文件来了解服务、操作、方法和服务方法交换的数据类型。外部 Web 服务的 custom-logic-service.wsdl 文件位于资源工具包的以下位置：C:\<infamdm 安装目录>\hub\resourcekit\samples\BESExternalCall\source\resources\webapp\WEB-INF\wsdl\

资源工具包包括实施自定义逻辑和验证的示例代码。安装资源工具包时，用于示例自定义逻辑和验证的 bes-external-call.ear 文件会部署到应用程序服务器上。

支持的事件

一个业务实体服务包含若干服务步骤。您可以针对其中任何一个步骤应用逻辑和验证。

您可以对以下事件进行外部调用：

- 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

如何配置外部调用

一个业务实体服务包含若干服务步骤。传入请求通过每个服务步骤传递。您可以为业务实体服务执行逻辑中的某些步骤配置外部服务调用。

请执行以下步骤来配置外部调用：

1. 构建并部署 bes-external-call.ear 文件。
2. 在置备工具中，请执行以下任务：
   a. 上载外部服务的 WSDL 文件。
   b. 将 Web 服务注册为 SOAP 服务。
   c. 配置外部调用。

有关上载 WSDL 文件、注册 SOAP 服务以及配置外部调用的详细信息，请参阅*《Multidomain MDM 置备工具指南》*。

有关构建和部署 EAR 文件的详细信息，请参阅*《Multidomain MDM 资源工具包指南》*。

示例：业务实体服务的自定义验证和逻辑

添加和合并“人员”记录时可以测试自定义验证和逻辑。自定义验证检查“人员”记录是否具有地址。您无法使用自定义逻辑合并两个电话号码。使用 REST API 来创建并合并“人员”记录。

1. 要在创建“人员”记录时检查验证，请执行以下步骤：
   a. 使用创建 API 来创建不含地址的“人员”记录。这时验证会出错。
   b. 使用创建 API 来创建含地址的“人员”记录。这时操作成功。

2. 要在合并记录时检查自定义逻辑，请执行以下步骤：
   a. 使用创建 API 来创建含地址和电话号码的两个“人员”记录。
   b. 使用预览合并 API 来合并两个“人员”记录。将 overrides 添加到预览合并请求来合并地址和电话号码。响应会显示一个地址，但是有两个电话号码。自定义逻辑会阻止合并电话号码。

先决条件

要检查自定义逻辑和验证，必须在置备工具中上载 WSDL 文件。您必须注册 SOAP 服务和操作。将服务与您想为其使用自定义逻辑和验证的业务实体和事件绑定起来。您可以为指定业务实体和事件测试逻辑和验证。

步骤 1.测试自定义验证

使用创建 API 来创建以下不含地址的“人员”记录：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_Sample/Person?systemName=Admin
{
        firstName: "John"
    }
```

这时验证会出错。

使用创建 API 来创建以下含地址的“人员”记录：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_Sample/Person?systemName=Admin
{
        firstName: "John",
        Addresses: {
            item: [
                {
                    cityName: "Toronto"
                }
            ]
        }
    }
```

此请求会创建“人员”记录。

步骤 2.测试自定义逻辑

执行以下步骤以测试自定义逻辑:

1. 使用创建 API 来创建含地址和电话的两个“人员”记录：

```
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"
                }
            ]
        }
    }
```

响应包含以下行 ID：

- 人员：161923、161924
- 地址：2123、2124
- 电话号码：101723、101724

2. 运行 PreviewMerge API 来合并两个“人员”记录：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-mdm_sample/Person/161923?action=previewMerge&depth=2
{
        keys: [
            {
                rowid: "161924"
            }
        ]
    }
```

响应是具有两个地址和两个电话号码的一个“人员”记录。

```
{
  "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": "SYSO          ",
      "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": "SYSO          ",
      "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": "SYSO          ",
        "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": "SYSO          ",
      "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": "SYSO          ",
        "hubStateInd": 1,
        "label": "Address",
        "cityName": "Ottawa"
      }
```

```
        }
      ]
    }
  }
}
```

3. 运行 PreviewMerge API 来合并两个“人员”记录，并替代合并地址和电话号码：

```
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]
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

响应显示了一个地址，但是有两个电话号码。

```
{
  "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": "SYSO          ",
    "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": "SYSO           ",
          "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": "SYSO           ",
          "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": "SYSO           ",
          "hubStateInd": 1,
          "label": "Addresses"
        }
      ]
    },
    "PersonDetails": {
      "link": [],
      "recordCount": 0,
      "pageSize": 0,
      "item": []
    }
  }
}
```

附录 A

使用 REST API 添加记录

本附录包括以下主题：

使用 REST API 添加记录概览

创建业务实体模型并配置业务实体结构后，可以使用 REST API 添加记录。

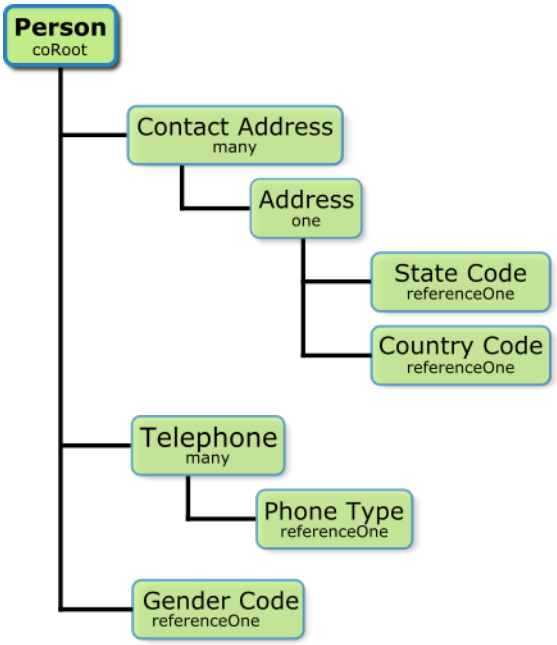
以下各节以 Person 业务实体为例来说明如何使用 REST API 添加记录。Person 业务实体包含您所在公司的员工的数据。

可以使用以下 API 添加员工的详细信息：

1. 获取有关架构的信息。可以使用获取元数据 REST API 获取有关业务实体的数据结构的信息，包括结构、字段列表、字段类型和查找字段的详细信息。或者，您也可以访问 XML 架构定义 (XSD) 文件，其中描述了可以使用的元素和属性。XSD 文件位于 `http://<host>:<port>/cmx/csfiles` 位置。
2. 创建记录。使用创建记录 REST API 创建记录。
3. 从已添加的记录读取数据。使用读取记录 REST API 检索记录中的数据。

Person 业务实体结构

我们将使用 REST API 添加一个 Person 记录。Person 根节点是 Person 业务实体结构中的最顶层节点。Person 根节点下有表示员工详细信息（例如，性别、地址和电话）的节点。



下图显示了 Person 业务实体的结构：

Person 是 Person 业务实体的根节点。节点名称下面列出的节点类型指示父节点和子节点之间的关系。Contact Address 和 Address 之间存在一对一关系，这指示每个联系人地址只能与一个地址关联。Person 和 Telephone 之间存在一对多关系，这指示一个 Person 记录可与多个电话号码记录关联。Person 和 Gender 之间存在一对一关系，这指示一个 Person 记录只能具有一个性别值。性别值驻留在查找表中。同样，州代码和国家/地区代码值也驻留在查找表中。

步骤 1. 获取有关架构的信息

可以使用获取元数据 REST API 获取有关架构的信息。获取元数据 API 可返回业务实体的数据结构。元数据会列出业务实体字段、字段类型以及查找字段的详细信息。

获取元数据 URL 使用以下格式：

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

以下示例请求将检索 Person 业务实体的元数据信息：

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

获取元数据响应

以下示例显示了 Person 业务实体的数据结构的部分摘录：

```
{
    "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
    }
}
```

步骤 2.创建记录

使用创建记录 REST API 创建记录。业务实体名称和源系统名称是必需参数。在请求主体中发送记录的数据。

创建记录 URL 使用以下格式：

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

systemName 参数是必需参数，指定了源系统名称。

“人员”业务实体包含“人员”根节点以及第二层的地址、性别和电话节点。

以下示例请求将创建“人员”记录：

```
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"
                }
            }
        ]
    }
}
```

请求主体指定了“人员”记录的以下详细信息：

- 名。
- 姓。
- 性别。
- 包含州代码和国家/地区代码的地址。
- 电话号码和电话类型（例如，家庭电话和移动电话）。

创建记录响应

以下示例响应显示了成功创建“人员”记录后的响应：


```
{
    "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             "
                    }
                }
            ]
        }
    }
}
```


注意: 响应主体包含记录和生成的 rowId。

如果您将工作流程配置为在创建记录时启动，将发生以下结果：

- 记录创建为挂起状态。
- 工作流程启动。
- 工作流程 ID 在响应表头中返回。

如果未配置工作流程，则默认情况下，记录将创建为活动状态。

如果使用交互 ID 处理请求，API 将在响应表头中返回交互 ID。

步骤 3.读取记录

使用读取记录 REST API 检索已添加的根记录的详细信息。您可以使用此 API 检索根记录的子记录的详细信息。

读取记录 URL 使用以下格式：

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

可以使用 depth 参数指定返回的子级数。指定 2 将返回根节点及其直接子代；指定 3 将返回根节点及其直接子代和孙代。使用以下 URL 可返回子记录的详细信息：

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

以下示例请求将返回根节点、直接子代和孙代的详细信息：

```
GET http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248?depth=3
```

该请求将返回活动记录的详细信息。

注意: 如果在创建记录时启动了工作流，则创建的记录将处于挂起状态。默认情况下，读取记录请求会读取活动记录。可以使用 recordStates 参数指定记录的挂起状态。

以下示例请求将读取挂起的记录的详细信息：

```
GET http://localhost:8080/cmx/cs/localhost-hub101-ds_ui1/Person/658248?depth=3&recordStates=PENDING
```

读取记录响应

以下示例响应显示了已添加的记录的详细信息：


```
{
    "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"
                }
            }
        ]
    }
}
```

附录 B

使用 REST API 上载文件

本附录包括以下主题：

使用 REST API 上载文件概览

可以使用 REST API 将文件上载到某个存储类型。上载文件后，可以将文件附加到记录或任务，或使用该文件来本地化 Data Director 用户界面。

根据您希望如何使用这些文件，使用的 REST API、文件组件和存储类型的组合可能有所不同。例如，要将文件附加到记录或任务，请创建文件的元数据，然后将文件上载到临时存储。上载文件后，可以将文件附加到数据库中的记录，或 BPM 存储中的任务。要本地化 Data Director 用户界面，请下载 ZIP 文件，修改压缩文件，然后将修改后的 ZIP 资源包上载到资源包存储。

文件的 REST API

可以使用一组通用 REST API 来上载和管理要附加或本地化的文件。

下表列出了文件的 REST API：

REST API	说明	受支持的存储类型
列出文件元数据	返回存储中的文件元数据列表。	BPM 或 TEMP
创建文件元数据	为存储中的文件创建元数据。	DB 或 TEMP

REST API	说明	受支持的存储类型
获取文件元数据	返回文件的元数据。	BPM、BUNDLE、DB 或 TEMP
更新文件元数据	更新文件的元数据。	DB 或 TEMP
上载文件内容	向存储中上载文件的内容。	BUNDLE、DB 或 TEMP
获取文件内容	下载文件的内容。	BPM、BUNDLE、DB 或 TEMP
删除文件	删除存储中的文件，包括与文件关联的组件，例如文件元数据或内容。	BUNDLE、DB 或 TEMP

文件组件

要向记录或任务附加文件，请创建文件的元数据，然后上载文件内容。要本地化 Data Director 用户界面，请下载资源包文件，然后上载修改后的资源包文件。

文件元数据

有关文件的信息，例如文件名、文件类型和内容类型。根据存储类型，您可能需要包含其他参数，例如创建者、创建时间和上载日期。

文件内容

文件的内容。例如，文本、图像、文档或资源包。

存储类型

在受支持的存储实施中上载并存储文件。使用的存储类型取决于是要本地化 Data Director 用户界面还是将文件附加到记录或任务。

下表介绍了受支持的存储类型：

BPM

将附加到任务的文件与任务数据存储在一起。向任务附加文件时，进程会将该文件从 TEMP 存储空间存储到 BPM 存储空间。

存储在 BPM 存储中的文件使用以下文件 ID 格式：`taskId::filename`。

注意: 要将文件附加到触发的任务或现有任务，请在置备工具中为任务触发器、任务类型和任务操作启用附件。有关详细信息，请参阅*《Multidomain MDM 置备工具指南》*。

BUNDLE

存储本地化 Data Director 用户界面的资源包文件。

存储在 BUNDLE 存储空间中的文件使用以下文件 ID 格式：`besMetadata`。

DB

将记录的文件附件存储在 C_REPOS_ATTACHMENTS 表中。向记录附加文件时，进程会将该文件从 TEMP 存储空间存储到 DB 存储空间。

存储在 DB 存储空间中的文件使用以下文件 ID 格式：DB_<RowID>。

注意: 要向记录附加文件，请在置备工具中配置数据类型为 FileAttachment 的字段。有关配置数据类型的详细信息，请参阅*《Multidomain MDM 置备工具指南》*。

TEMP

将文件临时存储在 C_REPOS_ATTACHMENTS 表中，并将文件标记为 TEMP。当文件成功上载到 BPM 或 DB 存储空间，或在预配置的到期时间之后，将从 TEMP 存储空间中删除文件。

存储在 TEMP 存储空间中的文件使用以下文件 ID 格式：TEMP_<ROWID_ATTACHMENT>。

有关配置到期时间的详细信息，请参阅*《Multidomain MDM 配置指南》*。

向记录附加文件

在向记录附加文件之前，请创建文件的元数据，然后再将文件上载到临时存储。

1. 要创建文件的元数据，请使用创建文件元数据 REST API，将存储类型设置为 TEMP。
   例如，以下请求可创建 Document_3.pdf 文件的元数据：

```
POST http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP
Content-Type: application/json
{
    "fileName": "Document_3.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

   注意: 始终在 TEMP 存储中创建文件元数据。

   创建文件元数据 REST API 返回文件 ID。文件 ID 格式如下：<存储类型>_<RowID>。其中，RowID 指的是您向存储上载的文件的行 ID。

   在示例中，API 调用为 Document_3.pdf 文件返回以下 ID：TEMP_SVR1.0JU3

   可以使用文件 ID 来上载、附加、更新、下载和删除文件。

2. 要上载文件，请使用上载文件内容 REST API，将存储类型设置为 TEMP。
   例如，以下请求可将文件上载到 TEMP 存储：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.0JU3/content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

   注意: 上载文件后，TEMP 存储会将文件存储预配置的 60 分钟时间。您必须在预配置的时间到期之前将文件附加到记录。

3. 要创建记录并将文件附加到新记录，请使用创建记录 REST API。

   例如，以下请求可创建记录并附加文件 ID 为 TEMP_SVR1.0JU3 的文件：

```
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_SVR1.0JU3"
      }
    ]
  }
}
```

注意: 向记录附加文件时，进程会将该文件存储到数据库。文件 ID 会更改为 DB_<RowID>，其中 DB 指示文件存储在数据库中。

4. 要替换向记录附加的文件，请使用上载文件内容 REST API，将存储类型设置为 DB。
   例如，以下请求会替换数据库中向记录附加的文件 ID 为 DB_SVR1.0JU3 的文件：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU3/content
Content-Type: application/octet-stream
<file object (upload using REST client)>
```

   注意: 请求 URL 中的存储类型为 DB。

5. 要在向记录附加文件后编辑文件元数据，请使用上载文件元数据 REST API，将存储类型设置为 DB。
   例如，以下请求会更新 DB 存储中与文件 ID DB_SVR1.0JU3 关联的文件的文件元数据：

```
PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU3
Content-Type: application/json
{
    "fileName": "Document_4.pdf",
    "fileType": "pdf",
    "fileContentType": "application/pdf"
}
```

6. 要下载向记录附加的文件，请使用获取文件内容 REST API，将存储类型设置为 DB。
   例如，以下请求会从 DB 存储中下载与文件 ID DB_SVR1.0JU3 关联的文件：

```
GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU3/content
```

7. 要删除向记录附加的文件，请使用删除文件内容 REST API，将存储类型设置为 DB。
   例如，以下请求会从 DB 存储中删除与文件 ID DB_SVR1.0JU3 关联的文件：

```
DELETE http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/DB/DB_SVR1.0JU3
```

向任务附加文件

创建文件的元数据，然后将文件内容上载到临时存储。上载文件后，将文件附加到触发的任务或现有任务。

注意: 要将文件附加到触发的任务或现有任务，请在置备工具中为任务触发器、任务类型和任务操作启用附件。有关详细信息，请参阅*《Multidomain MDM 置备工具指南》*。

1. 要创建文件的元数据，请使用创建文件元数据 REST API，将存储类型设置为 TEMP。
   例如，以下请求可创建 file1.txt 文件的元数据：

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

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

   注意: 始终在 TEMP 存储中创建文件元数据。

   创建文件元数据 REST API 返回文件 ID。文件 ID 格式如下：<存储类型>_<RowID>。其中，RowID 指的是您向存储上载的文件的行 ID。
   在示例中，API 调用为 file1.txt 返回以下 ID：TEMP_SVR1.1VDVS

   可以使用文件 ID 来上载、附加、更新和删除文件。

2. 要上载文件，请使用上载文件内容 REST API，将存储类型设置为 TEMP。
   例如，以下请求可将文件上载到 TEMP 存储：

   ```
   PUT http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/TEMP/TEMP_SVR1.1VDVS/content

   Test attachment content: file 1
   ```

   注意: 上载文件后，TEMP 存储会将文件存储预配置的 60 分钟时间。您必须在预配置的时间到期之前将文件附加到任务。

3. 将文件附加到管理记录时触发的任务。

   - 要将文件附加到创建记录时触发的任务，请使用带 taskattachments 参数的创建业务实体 REST API。
     例如，以下请求可创建记录并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

     ```
     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"
                    }
               ]
          }
     }
     ```

   - 要将文件附加到更新记录时触发的任务，请使用带 taskattachments 参数的更新业务实体 REST API。
     例如，以下请求可更新记录并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

     ```
     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"
         }
     }
     ```

   - 要将文件附加到合并记录时触发的任务，请使用带 taskattachments 参数的合并业务实体 REST API。
     例如，以下请求可合并记录并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

     ```
     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"
    }
  }
}
```

- 要将文件附加到取消合并记录时触发的任务，请使用带 taskattachments 参数的取消合并业务实体 REST API。
  例如，以下请求可取消合并记录并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

```
POST http://localhost:8080/cmx/cs/localhost-orcl-DS_UI1/Person/2478248?
action=unmerge&taskAttachments=TEMP_SVR1.1VDVS
{
       rowid: "4880369"
}
```

4. 向现有任务附加文件。

- 要在更新任务时附加文件，请使用请求主体中含 attachments 的更新任务 REST API。
  例如，以下请求可更新任务并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

```
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"
          }
     ],
     ...
}
```

- 要在执行任务操作时附加文件，请使用请求主体中含 attachments 的执行任务操作 REST API。
  例如，以下请求可执行任务操作并附加文件 ID 为 TEMP_SVR1.1VDVS 的文件：

```
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"
          }
     ],
     ...
}
```

将文件附加到任务后，进程会移动 TEMP 存储中的文件，将该文件与任务数据一起存储在 BPM 存储中。文件 ID 将更改为 taskId::filename。

上载资源包文件

要本地化 Data Director 用户界面，请下载资源包 ZIP 文件，修改 ZIP 文件中的文件，然后将修改后的 ZIP 文件上载到资源包存储。

1. 要下载资源包 ZIP 文件，请使用获取文件内容 REST API，并将存储类型设置为 BUNDLE。
   例如，以下请求将下载资源包 ZIP 文件：

   ```
   GET http://localhost:8080/cmx/file/localhost-orcl-MDM_SAMPLE/BUNDLE/besMetadata/content
   ```

2. 通过添加特定于语言的资源包文件修改 ZIP 文件。
   例如，要将字段名称、标签和表名称本地化为俄语，请添加 besMetadata_ru.properties 文件。

3. 要上载修改后的资源包 ZIP 文件，请使用上载文件内容 REST API，并将存储类型设置为 BUNDLE。
   例如，以下请求将上载资源包 ZIP 文件：

   ```
   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
   ```

索引

S

U

W

X

Y

Z