Informatica®

Informatica® Multidomain MDM
10.4

Services Integration Framework Guide

Informatica Multidomain MDM Services Integration Framework Guide
10.4
March 2020

Publication Date: 2023-01-04

Table of Contents

Preface 8

Informatica Resources 8

- Informatica Network 8
- Informatica Knowledge Base 8
- Informatica Documentation 8
- Informatica Product Availability Matrices 9
- Informatica Velocity 9
- Informatica Marketplace 9
- Informatica Global Customer Support 9

Chapter 1: Introduction to Services Integration Framework 10

Services Integration Framework (SIF) 10

SIF SDK 11

Use Cases for SIF 11

SiperianClient Library Classes 12

Access Protocols 12

- Web Services 13
- XML Over HTTP 14

Chapter 2: Setting Up the SIF SDK 15

Before You Begin 15

Installing the SIF SDK 15

SIF API Reference Documentation 16

Setting Up a Sample Eclipse Client 16

- Importing the Sample Project File 16
- Identifying the Missing Library JAR Files 16
- Adding the Missing Library JAR Files 17
- Customizing the Properties in the Library Files 18

Sample Code to Retrieve Records 18

- Running the Sample Code to Retrieve Records 19

Chapter 3: Request and Response Objects 20

Request and Response Objects Overview 20

Request Objects 20

- SiperianRequest Class 21

Response Objects 21

- SiperianResponse Class 22

Sample Java Class Diagram 22

Chapter 4: Transactions and Exception Handling. 23

Transactions Overview. 23
Business Entity Services. 23
Example EJB Transaction for WebLogic and WebSphere. 24
Example EJB Transaction for JBoss. 24
Exception Handling. 25

Chapter 5: ORS-Specific SIF API. 27

ORS-Specific SIF API Overview. 27
Supported Repository Objects. 28
ORS-Specific SIF API Properties. 28
Repository Objects Statuses. 29
Generating and Deploying an ORS-Specific SIF API. 29
Renaming an ORS-specific SIF API. 30
Downloading an ORS-Specific Client JAR File. 30
Removing an ORS-Specific SIF API. 30
Using ORS-Specific Client JAR Files with SIF SDK. 30
Archive Table. 31
ORS-Specific SIF Classes. 31
Cleanse<Resource Name>. 31
CleansePut<Resource Name>. 32
Get<Resource Name>. 33
Put<Resource Name>. 34
SearchMatchColumn<Resource Name>. 35
SearchMatchRecord<Resource Name>. 35
SearchQuery<Resource Name>. 36
ORS-Specific SIF API Field Parameters. 37

Chapter 6: Asynchronous SIF Requests. 40

Asynchronous SIF Requests Overview. 40
Architecture of JMS Message Queue for SIF. 40
Processing Asynchronous SIF Requests. 41

Chapter 7: ORS-Specific JMS Event Messages. 42

ORS-Specific JMS Event Messages Overview. 42
Elements in a Response XML Message. 43
Sample Response XML Message for an Update Event. 44

Chapter 8: Using Security Access Manager. 45

Security Access Manager Workbench Overview. 45
Using the Security Access Manager Workbench. 46
Permissions for SIF Requests. 46

Chapter 9: Using Dynamic Data Masking. 50

Dynamic Data Masking Overview. 50
Rules. 50
Supported SIF Requests for Dynamic Data Masking. 51

Chapter 10: SIF API Reference. 52

Functional SIF API Listing. 52
Reference SIF API Listing. 56
AcceptUnmatchedRecordsAsUnique. 56
AddRelationship. 57
ApplyChangeList. 57
AssignUnmergedRecords. 58
Audit 59
Authenticate. 60
CanUnmergeRecords. 60
CleanTable. 60
Cleanse 61
CleansePut. 62
ClearAssignedUnmergedRecords. 66
CreateChangeList. 66
CreateTask. 67
Delete. 69
DeleteRelationship. 71
DescribeSiperianObject. 71
ExecuteBatchAutoMatchAndMerge. 72
ExecuteBatchAutomerge. 73
ExecuteBatchBVTSnapshot. 74
ExecuteBatchDelete. 74
ExecuteBatchExternalMatch. 75
ExecuteBatchGenerateMatchTokens. 76
ExecuteBatchGroup. 77
ExecuteBatchKeyMatch. 77
ExecuteBatchLoad. 78
ExecuteBatchMatch. 79
ExecuteBatchMatchAnalyze. 79
ExecuteBatchPromote. 80
ExecuteBatchRecalculateBo. 81
ExecuteBatchRecalculateBvt. 82
ExecuteBatchResetMatchTable. 82
ExecuteBatchRevalidate. 83
ExecuteBatchStage. 84
ExecuteBatchSynchronize. 84

ExecuteBatchUnmerge . . . 85
ExecuteBatchValidateFKRelationships . . . 86
FlagForAutomerge . . . 86
GenerateConstraints . . . 87
Get . . . 88
GetAggregatePeriod . . . 90
GetAssignableUsersForTasks . . . 92
GetAssignedRecords . . . 93
GetBatchGroupStatus . . . 93
GetBvt . . . 94
GetEffectivePeriods . . . 95
GetEntityGraph . . . 96
GetLookupValue . . . 97
GetLookupValues . . . 98
GetMatchedRecords . . . 98
GetMergeHistory . . . 99
GetOneHop . . . 99
GetOrsList . . . 100
GetOrsMetadata . . . 101
GetSearchResults . . . 101
GetSiperianObjectCompatibility . . . 103
GetSystemTrustSettings . . . 103
GetTaskLineage . . . 104
GetTasks . . . 107
GetTrustGraphData . . . 109
GetTrustScore . . . 109
GetUnmergedRecordCount . . . 110
GetXrefForEffectiveDate . . . 110
Link . . . 112
ListSiperianObjects . . . 112
Merge . . . 114
MultiMerge . . . 114
PreviewBVT . . . 115
PromotePendingXrefs . . . 116
Put . . . 118
ReassignRecords . . . 123
RegisterCustomIndex . . . 123
RegisterCustomTableObject . . . 124
RegisterUsers . . . 125
RemoveMatchedRecords . . . 126
ResetBatchGroup . . . 126
Restore . . . 127

SearchHmQuery . . . 128
SearchLookupValues . . . 128
SearchMatch . . . 128
SearchQuery . . . 133
SearchRequestBase . . . 135
SearchResponseBase . . . 136
SetPassword . . . 136
SetRecordState . . . 136
Tokenize . . . 138
Unlink . . . 139
Unmerge . . . 139
UnregisterUsers . . . 141
UpdateMatchRecord . . . 141
UpdateRelationship . . . 142
UpdateTask . . . 143
ValidateChangeList . . . 145
ValidateMetadata . . . 146
ValidateTasks . . . 147

Chapter 11: Troubleshooting . . . 149

Appendix A: Identifiers . . . 150

List of Identifiers . . . 150
SiperianObjectUID . . . 150
RecordKey . . . 154

Appendix B: Frequently Asked Questions . . . 155

Index . . . 157

Preface

Use the Services Integration Framework (SIF) to integrate the Informatica® MDM Hub functionality with your applications. The *Multidomain MDM Services Integration Framework Guide* explains how to access the data that the MDM Hub provides by using APIs.

This guide assumes that you have a working knowledge of MDM Hub and are familiar with Java and APIs.

Informatica Resources

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

Informatica Network

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

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

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

Informatica Knowledge Base

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

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

Informatica Documentation

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

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

Informatica Product Availability Matrices

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

Informatica Velocity

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

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

Informatica Marketplace

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

Informatica Global Customer Support

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

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

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

CHAPTER 1

Introduction to Services Integration Framework

This chapter includes the following topics:

- Services Integration Framework (SIF), 10
- SIF SDK, 11
- Use Cases for SIF, 11
- SiperianClient Library Classes, 12
- Access Protocols, 12

Services Integration Framework (SIF)

Use Services Integration Framework (SIF) to invoke the MDM Hub operations from external applications in real time. SIF uses service-oriented architecture that provides application functionality as services to other applications.

You can configure SIF for the MDM Hub to interface with the client programs. SIF serves as the middle tier in the client-server model. You can use the SIF access protocols to implement the request and response interactions.

Note: Only admin users can access private resources through the SIF requests.

The Services Integration Framework offers the following services and events:

Process Services

Process services are integration processes and people-to-system processes. For example, in a customer domain, the process services include activities such as verifying customers and approving customers.

Business Events

Business events are rule-driven events and actions that are based on business logic. For example, you might evaluate the impact of a new relationship.

Business Services

Business services act on base objects in the data model. For example, in a customer domain, you can create a Customer base object or retrieve a Customer base object.

Data Services

Data services act on records in the tables. You can insert a record, retrieve a record, or update a record.

Data Events

Data events are events that affect master data or source data. For example, a customer last name changes in a source record or an address changes in a master record.

The SIF requests can directly interact with each other. Data services can interact with data events, process services with data services, data services with business services, and data events with process services.

You can use external applications or the MDM Hub to generate the data events. Use the event-driven architecture (EDA) capabilities of the MDM Hub that include event capture, event processing, event filtering, and event generation to handle these events. The EDA components and external applications for query and data synchronization operations can use the services provided in the MDM Hub. You can use the existing infrastructure, such as an enterprise service bus (ESB) and enterprise application integration (EAI) technologies, such as TIBCO, webMethods, and message-oriented middleware with SIF.

SIF SDK

Use the SIF SDK to develop web services and Java applications that interact with the MDM Hub. The SIF SDK includes utilities to build and deploy SIF applications, a set of Java classes to create services, and sample codes to build web services. You can use the SIF SDK to create data objects, client services, business services, and GUI controls for creation and deployment of web-based and rich-client applications.

The SIF SDK is packaged with the MDM Hub Resource Kit installer. You can find the directory structures, libraries, online documentation for SIF, and build files in the following directory: `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk`.

You can copy the SIF SDK to any client system on which you want to develop and run programs to interact with the MDM Hub. If you can run a Java virtual machine (JVM) on the client system, you can use the Java classes included in the SIF SDK. You can configure the SIF SDK to use any access protocol. If you cannot run a JVM, you must explicitly use web services, Java Message Service (JMS), or XML over HTTP.

You can use the SIF SDK for the following tasks:

- Automatic generation and deployment of data objects and data services for the web services-based interactions.
- Generation of a client .jar file that includes data objects. You can use the data objects in external applications.
- Creation and management of complex integration scenarios by combining data objects from different MDM Hub schemas.

Use Cases for SIF

You can use the SIF requests to develop any of the following sample applications:

- Web services to perform operations in the MDM Hub.
- Business process modeling (BPM) and workflow integration by using the Java API or SOAP directly.
- A Java Swing UI to query, view, and edit data in the MDM Hub.
- Server components in a J2EE application server to get and update data in the MDM Hub. Enterprise JavaBeans (EJB) can seamlessly integrate with the MDM Hub transactions.

- An application based on JavaServer Pages (JSP) or servlet to present a portal view of the MDM Hub data.

SiperianClient Library Classes

Use the SiperianClient library classes to build custom web services that interact with the MDM Hub. You can use the sample codes located in the following directory to build a custom web service: `<Resource Kit Installation Directory>\samples`.

Use the `process` method of the `com.siperian.sif.client.SiperianClient` class to implement the request and response interactions between the client program and the MDM Hub. The `process` method accepts any subclass of the `com.siperian.sif.message.SiperianRequest` class as an argument. If the `process` method successfully processes the request, it returns the `com.siperian.sif.message.SiperianResponse` object as the response. Otherwise, it returns the `com.siperian.sif.client.SiperianServerException` object.

The `com.siperian.sif.message.mrm` and `com.siperian.sif.message.hm` packages include the request and response objects to perform operations on the MDM Hub. The `com.siperian.sif.client` package manages the details of the client communication with the MDM Hub. You can use the access protocols, such as Enterprise JavaBeans (EJB), XML over HTTP, or SOAP, to communicate with the MDM Hub.

The `siperian-api.jar` file located in the following directories contains all the SiperianClient library classes:

- `<Resource Kit Installation Directory>\sdk\sifsdk\lib`
- `<MDM Hub Installation Directory>\hub\server\lib`

When you use the SiperianClient library classes, ensure you also have all the dependent files. The `siperian-api.jar` file is dependent on the following JAR files:

siperian-common.jar

- `<Resource Kit Installation Directory>\sdk\sifsdk\lib`
- `<MDM Hub Installation Directory>\hub\server\lib`

informatica-bpm-adapter.jar

- `<Resource Kit Installation Directory>\sdk\sifsdk\lib`
- `<MDM Hub Installation Directory>\hub\server\lib`

siperian-server-pkiutil.jar

- `<Resource Kit Installation Directory>\sdk\sifsdk\lib`
- `<MDM Hub Installation Directory>\hub\server\lib\pkiutils`

If you run an external SIF client, you might also require other JAR files in the following directory:

`<Resource Kit Installation Directory>\sdk\sifsdk\lib`

Access Protocols

You can use one of the following access protocols for the SIF request and response interactions:

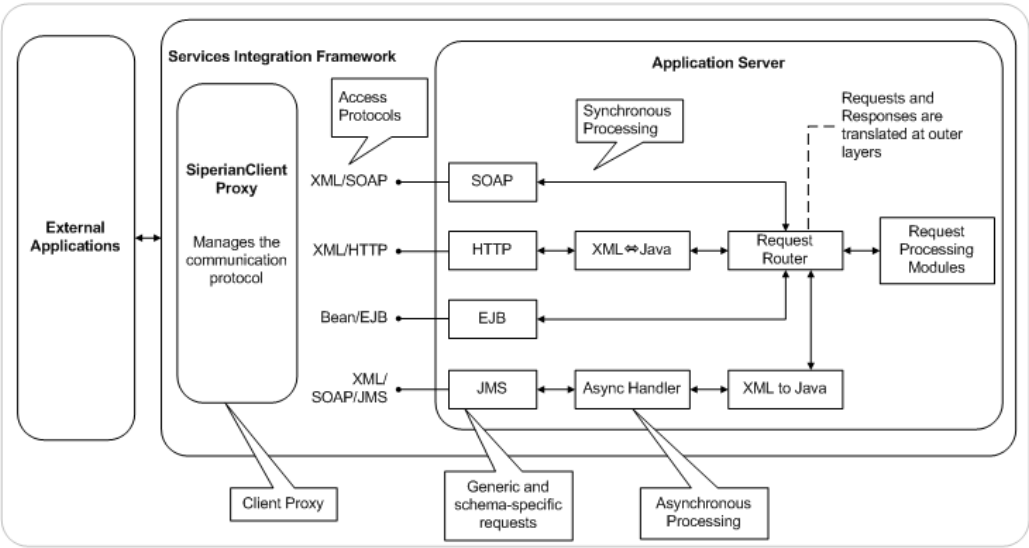
- Tightly coupled Java remote procedure calls based on Enterprise JavaBeans (EJB) in a Java development environment.

- Loosely coupled web services that use SOAP protocol. Use Web Services Description Language (WSDL) to define the request and response XML. The development environment can be Eclipse, Microsoft Visual Studio, or other web service client tools.
- XML over HTTP protocol, which is similar to web services but without the SOAP envelope.
- Asynchronous JMS-based messages that use the XML over HTTP protocol.

The access protocol runs on top of the native MDM Hub protocol, which accepts request in the XML or EJB format and returns responses in the same format.

You can use a SiperianClient proxy in a Java development environment to manage the communication protocol for the SIF requests.

The following image shows how SIF processes the request and response interactions:



When you cannot or do not want to use the SiperianClient Java classes, you can use other access protocols to directly interact with SIF. This guide does not include information about how to directly use EJB or JMS protocols. You can use these protocols through SiperianClient if you perform any of the following tasks:

- Use an appropriately configured `AsynchronousOptions` object with a SIF request.
- Place the request directly onto the `siperian.sif.jms.queue` message queue.

Web Services

After you install the MDM Hub on an application server, you can use the SiperianClient library classes to create web services on that application server. You can interrogate the web service to get the Web Services Description Language (WSDL) descriptions of the web service's operations and arguments. The operations and arguments are equivalent to the methods and arguments of the SiperianClient Java classes that the web service exposes.

WSDL

WSDL is an XML-based interface definition language that describes web services and how to access them. If you use a web service to interface with the MDM Hub, use the classes and methods described in the SIF API reference documentation through proxies that wrap the interactions in SOAP messages. You can use a tool based on your development environment to interpret WSDL.

The development environment can be Eclipse, Microsoft Visual Studio, or other web service client tools. For example, .NET has tools to read WSDL and create proxies that you can call from the programming language you use. The Eclipse integrated development environment has a web services browser that reads the WSDL and presents the information in a user-friendly way.

Use the following URL to access the WSDL, where `host` is the name of the computer that runs the application server and `port` is the port on which the host computer accepts the MDM Hub requests:

```
http://<host>:<port>/cmx/request/wsdl
```

The SiperianClient proxy uses the SOAP protocol to communicate with the web service and receive requests from your application program. The SiperianClient proxy translates the requests into SOAP messages and send them to the web service. The web service decodes the SOAP messages and translates them to Java calls for the SiperianClient running on the application server. The web service receives responses from the SiperianClient, encodes them into SOAP messages, and sends them back to the SiperianClient proxy. The SiperianClient proxy returns the responses to your application program.

The MDM Hub uses Axis version 1.3, which is an XML-based web service framework. Use Axis to configure SIF as web services and access the web services through a URL. For example, if you use SoapUI to view a list of web services, the tool presents the list of web services that you configured in Axis. When you deploy the Hub Server on the application server, Axis is automatically deployed.

You can also create and deploy a web service to process ORS-specific requests.

XML Over HTTP

Use the HTTP protocol to send requests to the MDM Hub and receive responses in the XML format. You can use the following schemas to describe and manage the requests and responses:

- `siperian-core.xsd`. Contains the schema elements.
- `siperian-types.xsd`. Contains the type definitions.
- `siperian-metadata.xsd`. Describes the objects used in the `ListSiperianObjects` and `DescribeSiperianObjects` classes.

You can use the following URLs to access the schemas, where `host` is the name of the computer that runs the application server and `port` is the port on which the host computer accepts the MDM Hub requests:

```
http://<host>:<port>/cmx/request/xsd/siperian-core.xsd
http://<host>:<port>/cmx/request/xsd/siperian-types.xsd
http://<host>:<port>/cmx/request/xsd/siperian-metadata.xsd
```

Use the schema to construct an XML request message, and use the HTTP POST method to send the request to the following address:

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

The body of the HTTP response is the SIF response and is encoded in XML according to the schema definitions.

CHAPTER 2

Setting Up the SIF SDK

This chapter includes the following topics:

- Before You Begin, 15
- Installing the SIF SDK, 15
- SIF API Reference Documentation, 16
- Setting Up a Sample Eclipse Client, 16
- Sample Code to Retrieve Records, 18

Before You Begin

Before you use the SIF SDK, you must install the following applications:

- MDM Hub
- Application server

For more information about product requirements and supported platforms, see the Product Availability Matrix on the Informatica My Support Portal:
https://mysupport.informatica.com/community/my-support/product-availability-matrices

Note: Ensure that the date format of the application server host computer is dd-mmm-yyyy.

Installing the SIF SDK

The SIF SDK is packaged with the MDM Hub Resource Kit installer. Use the MDM Hub Resource Kit installer to install the SIF SDK and SIF API reference documentation. After you install the Resource Kit, you can find the directory structures, libraries, SIF API reference documentation, and build files in the following directory:
`<Resource Kit Installation Directory>\hub\resourekit\sdk\sifsdk`.

SIF API Reference Documentation

The SIF SDK includes SIF API reference documentation that describes all the SiperianClient library classes, methods, and attributes. You can find the SIF API reference documentation in the following directory: `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\javadoc`.

To open the SIF API reference documentation, double-click the `index.html` file.

Setting Up a Sample Eclipse Client

Use the sample `.project` file that is part of the SIF SDK to create a sample Eclipse client.

1. In the Eclipse IDE, import the sample `.project` file located in the `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk` directory into your Eclipse workspace.
2. Identify the missing library JAR files that are specific to the application server and add them to the build class path. The sample `.project` file requires the library JAR files to set up the proper Java build path.
3. Customize the following files according to your environment:
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\build.xml`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\my.properties`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\source\properties\log4j.xml`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\source\properties\siperian-client.properties`

Importing the Sample Project File

1. From the Eclipse IDE, click **File** > **Import**.

   The **Import** dialog box appears.
2. Select **Existing Projects into Workspace** and click **Next**.

   The **Import Projects** page appears.
3. Select **Select root directory**, and browse to the `.project` file located in the following directory:`<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk`

   The project name appears under **Projects**. The default project name is SIF-SDK-[ORS NAME].
4. Under **Projects**, select the project, and click **Finish**.

   The sample project appears in the package explorer.

Identifying the Missing Library JAR Files

After you import the sample `.project` file, verify whether all the required library JAR files for the project are added to the build class path. The library JAR files are specific to the application server.

1. In the Eclipse IDE, right-click the sample project, and select **Properties**.

   The **Properties** dialog box appears.
2. Click **Java Build Path**.
3. On the **Libraries** tab, identify the missing JAR files that are marked with a red cross mark.

Required JAR Files for JBoss

The following library JAR files are required to set up an Eclipse client for a JBoss application server.:

- `jboss-client.jar`
- `picketbox-4.0.16.Final-redhat-1.jar`

The name of the JAR files might vary according to the version of JBoss, so use the equivalent JAR files based on your environment. You can find the library JAR files in the following directory:`<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\lib\jboss`

Required JAR Files for WebLogic

The `wlthint3client.jar` library JAR file is required to set up an Eclipse client for a WebLogic application server. The name of the JAR file might vary according to the version of WebLogic, so use the equivalent JAR file based on your environment.

You can find the library JAR file in the following directory:`<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\lib\weblogic`

Required JAR Files for WebSphere

The following library JAR files are required to set up an Eclipse client for a WebSphere application server:

- `admin.jar`
- `jmxc.jar`
- `rsadapterspi.jar`
- `wasjmx.jar`
- `wasx.jar`

The name of the JAR files might vary according to the version of WebSphere, so use the equivalent JAR files based on your environment. You can find the library JAR files in the following directory:`<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\lib\websphere`

Adding the Missing Library JAR Files

After you identify the missing library JAR files, add them to the build class path.

1. In the Eclipse IDE, right-click the sample project, and select **Properties**.
2. Click **Java Build Path**.
3. On the **Libraries** tab, perform one of the following tasks:
   - To add a JAR file that is inside your workspace, click **Add JARs**.
   - To add an external JAR file, click **Add External JARs**.

   The **JAR Selection** dialog box appears.
4. Browse to the location of the JAR file that you want to add, and select the JAR file.

   The JAR file is added to the build class path.
5. Similarly, add the other missing JAR files.

Customizing the Properties in the Library Files

The SIF SDK includes some library files that contain properties related to your development environment. You must customize the properties before you use the SIF SDK.

1. Open the following files in a text editor:
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\build.xml`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\my.properties`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\source\properties\log4j.xml`
   - `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\source\properties\siperian-client.properties`

   Note: If you do not want to use the `build.xml` file, use the library JAR files located in the following directory to manually compile the code:`<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk\lib`
2. Customize the properties according to your development environment, and comment the properties that are not relevant to your development environment.

Sample Code to Retrieve Records

After you set up an Eclipse client, you can write code to interact with the MDM Hub. For example, create a class and add the following sample code that uses the `SearchQueryRequest` and `SearchQueryResponse` classes to retrieve records:

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import com.siperian.sif.client.SiperianClient;
import com.siperian.sif.client.SoapSiperianClient;
import com.siperian.sif.message.Parameter;
import com.siperian.sif.message.Record;
import com.siperian.sif.message.mrm.SearchQueryRequest;
import com.siperian.sif.message.mrm.SearchQueryResponse;


public class SearchQuery {

    public static void main(String[] args) {
        File file = new File("E:\\siperian-client.properties");
        System.out.println("Reading File:" + file.getAbsolutePath());
        if (!file.exists()) {
            System.out.println("***ERROR -> Properties File does not exist in location -
");
            return;
        }
        SoapSiperianClient sipClient = (SoapSiperianClient)
SiperianClient.newSiperianClient(file);
        SearchQueryRequest request = new SearchQueryRequest();
        request.setRecordsToReturn(5);
        request.setSiperianObjectUid("BASE_OBJECT.C_PARTY");
        request.setFilterCriteria("C_PARTY.FIRST_NAME =?");
        ArrayList params = new ArrayList(2);
        params.add(new Parameter("3333"));
        request.setFilterParameters(params);
        SearchQueryResponse response = (SearchQueryResponse) sipClient.process(request);
        List <Record> records = response.getRecords();
        for (Record record : records) {
```

```
        System.out.println("Period Start Date: " +
record.getField("PERIOD_START_DATE").getDateValue().toString());
        System.out.println("Period End Date: " +
record.getField("PERIOD_END_DATE").getDateValue().toString());
    }
}

}
```

Running the Sample Code to Retrieve Records

After you create a class, run the sample class as Java application.

To run the sample class as Java application, in the Eclipse IDE, right-click the sample class, and select **Run As** > **Java Application**. The Console View displays the output.

CHAPTER 3

Request and Response Objects

This chapter includes the following topics:

- Request and Response Objects Overview, 20
- Request Objects, 20
- Response Objects, 21
- Sample Java Class Diagram, 22

Request and Response Objects Overview

Every operation that you perform by using SIF requires a set of request and response objects. A request object includes methods that indicate the action that you want to perform on the MDM Hub, and a response object includes methods that return the result of that action.

A SIF class can represent a request object or a response object. A SIF class that represents a request object has a suffix of 'Request,' and a SIF class that represents a response object has a suffix of 'Response.' For example, the `PutRequest` class represents a request object, and the `PutResponse` class represents the response to the `PutRequest` object.

The request and response objects can have Java or XML representations. To represent request and response objects in Java, you must internally convert an XML representation to Java.

Use EJB protocol to invoke the SIF classes if you want multiple requests to participate in a single transaction.

Request Objects

A request object includes methods that indicate the action that you want to perform on the MDM Hub. A SIF class that represents a request object is a subclass of the `SiperianRequest` class and extends the `SiperianRequest` class.

For example, the following sample uses the `SearchQueryRequest` object:

```
SearchQueryRequest request = new SearchQueryRequest();
request.setRecordsToReturn(5); //Required
request.setSiperianObjectUID("PACKAGE.PARTY_ADDRESS_READ_PKG");//Required
request.setFilterCriteria("PARTY_FULL_NAME LIKE ?");
```

The request runs the `PARTY_ADDRESS_READ_PKG` package, uses the `PARTY_FULL_NAME LIKE` filter criteria, and returns no more than five records.

SiperianRequest Class

The `SiperianRequest` class is the base class for the all the SIF classes that represent request objects.

The `SiperianRequest` class includes methods that set the following information:

User Name and Password

User credentials to run the request. If the user does not have access permission to perform the operation, the SIF request fails.

ORS ID

ID of the ORS to which the request is directed. If you do not specify the ORS, the request is directed to the default ORS.

Interaction ID

Interaction ID to group multiple requests into a single interaction.

Asynchronous Options

Indicates whether to process the request asynchronously or synchronously. If the value of the `asynchronousOptions` parameter is null, SIF processes the request synchronously. If the value is not null, SIF processes the request synchronously or asynchronously according to the value that you set.

When you process a request asynchronously, SIF returns a dummy response with a message that the request is asynchronously processed. The actual response goes to the JMS queue that you specify. If you do not specify a queue, SIF discards the actual response.

The `SiperianRequest` class includes methods that get the following information:

Transaction Attribute Type

Specifies whether the request object can participate in the transactions. You can get any of the following transaction attribute types:

- NOT_SUPPORTED if the request cannot participate in the transactions.
- SUPPORTS if the request can participate in the transactions.
- REQUIRED if the request requires a transaction.
- REQUIRES_NEW if the request requires a new transaction.

Name of the Request

Name of the class of which the request object is an instance. For example, if x is an instance of the `AuditRequest` class, `x.getRequestName()` returns `AuditRequest` as the name of the request.

Response Objects

A response object represents the response to the corresponding request object and includes methods that return the result of the action that you perform on the MDM Hub. A SIF class that represents a response object is a subclass of the `SiperianResponse` class and extends the `SiperianResponse` class into it.

For example, the following sample uses the `GetOrsMetadataResponse` object:

```
GetOrsMetadataResponse getOrsMetadataResponse = (GetOrsMetadataResponse)
sifClient.process(getOrsMetadataRequest );
System.out.println("ORS Metadata (first line only): " +
getOrsMetadataResponse.getRepositoryXml().substring(0, 80));;
```

SiperianResponse Class

The `SiperianResponse` class is the base class for the all the SIF classes that represent response objects.

The `SiperianResponse` class includes methods that get the following information:

Interaction ID

Interaction ID for the request.

Message

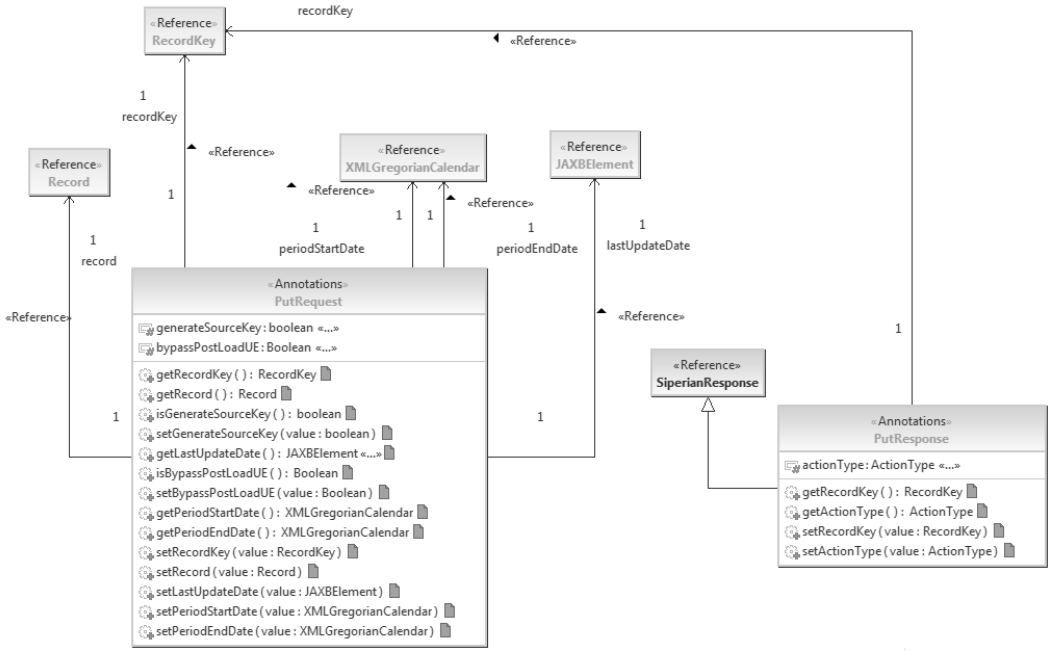
Brief message that indicates the status of the request.

Sample Java Class Diagram

The sample class diagram represents the relationship and dependencies between the `PutRequest` and `PutResponse` classes, methods, and their attributes. The class is represented by a rectangular box that has the following components:

- Name of the class.
- Attributes that the methods use.
- List of methods.

The following image shows the relationship between the `PutRequest` and `PutResponse` classes and their dependencies with other objects:



The `PutResponse` class has a dependency on the `SiperianResponse` class, which is the base class for all the response objects.

CHAPTER 4

Transactions and Exception Handling

This chapter includes the following topics:

- Transactions Overview, 23
- Business Entity Services, 23
- Example EJB Transaction for WebLogic and WebSphere, 24
- Example EJB Transaction for JBoss, 24
- Exception Handling, 25

Transactions Overview

A transaction is a set of procedures or operations that you can perform on the MDM Hub.

A transaction must complete all the operations or procedures without generating any error to be a successful transaction. If any procedure or operation in a transaction fails and generates an error, the entire transaction is rolled back. You can use SIF requests and MDM Hub requests within a transaction. You can use transactions in a business entity service to create a response object.

Note: You cannot use transactions in a web service.

Business Entity Services

A business entity service uses multiple requests to create an appropriate response object. For example, a service that returns a complete client profile can use multiple requests to return a profile, one or more addresses, emails, and phone numbers.

Use EJB access protocol to enable transactions for a business entity service. A business entity service uses service calls to trigger server services.

Note: The MDM Hub does not control the transactions from an external business entity service, and your service must manage these transactions.

Example EJB Transaction for WebLogic and WebSphere

Use a `sifClient` object as an `EjbSiperianClient` instance for EJB transactions.

The following sample code specifies the WebLogic properties for the MDM Hub to create an `EjbSiperianClient` instance:

```
java.naming.provider.url=t3://10.1.48.123:7001/
java.naming.security.principal=weblogic
java.naming.security.credentials=weblogic
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.security.authentication=strong
weblogic.security.SSL.ignoreHostnameVerification=true
```

The following sample code uses the `createTX(int)` method to create a transaction:

```
UserTransaction tx = ((EjbSiperianClient)sifClient).createTX(30)
```

The following sample code commits a transaction:

```
tx.begin();
// sif api calls
tx.commit();
```

The following sample code rolls back a transaction:

```
tx.rollback()
```

Example EJB Transaction for JBoss

Create an instance of `SiperianClient` with `ejb` as the `SiperianClient.SIPERIANCLIENT_PROTOCOL` property value, and start JBoss with the `-Djboss.node.name=<Node Name>` parameter for user transactions.

Ensure that you add the following JAR files to the CLASSPATH environment variable:

- `<MDM Hub Installation Directory>\hub\server\lib\siperian-api.jar` or `<Resource Kit Installation Directory>\sdk\sifsdk\lib\siperian-api.jar`
- `<MDM Hub Installation Directory>\hub\server\lib\siperian-common.jar` or `<Resource Kit Installation Directory>\sdk\sifsdk\lib\siperian-common.jar`
- `<JBoss Installation Directory>\bin\jboss-client.jar`
- `<JBoss Installation Directory>\modules\system\layers\base\org\picketbox\main \picketbox-4.<x>.<x>.Final-redhat-1.jar`

The following sample code specifies the JBoss properties for the MDM Hub to create a `SiperianClient` instance:

```
Properties properties = new Properties();
properties.put(SiperianClient.SIPERIANCLIENT_PROTOCOL, "ejb");
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
properties.put( "java.naming.factory.url.pkgs", "org.jboss.ejb.client.naming" );
properties.put( Context.PROVIDER_URL, "remote://<JBoss host name>:<JBoss port>");
properties.put("jboss.naming.client.ejb.context", true);
properties.put(Context.SECURITY_PRINCIPAL, "<User name of the MDM Hub user registered in
JBoss>");
properties.put(Context.SECURITY_CREDENTIALS, "<Password of the MDM Hub user registered
in JBoss>");
properties.put("siperian-client.orsId", "<Operational Reference Store ID>");
```

```
properties.put("siperian-client.username", "<User name of the MDM Hub user registered in
JBoss>");
properties.put("siperian-client.password", "<Password of the MDM Hub user registered in
JBoss>");
System.setProperty("jboss.node.name", "localhost"); //Value of this property should be
the same as it was provided while starting JBoss with the -Djboss.node.name option.
client = SiperianClient.newSiperianClient(properties);
```

The following sample code uses the `createTX(int)` method to create a transaction:

```
UserTransaction tx = client.createTX(30);
```

The following sample code commits a transaction:

```
tx.begin();
// sif api calls
tx.commit();
```

The following sample code rolls back a transaction:

```
tx.rollback()
```

Exception Handling

When a SIF request fails, the SIF SDK handles exception through the `com.siperian.common.SipRuntimeException` object. The `SipRuntimeException` object returns the same error codes and messages that the MDM Hub returns.

The `SipRuntimeException` object includes methods that return the following information:

Error Code and Message

Error codes and messages that the MDM Hub returns.

Request Name

Name of the request that failed.

The following sample code uses the `SipRuntimeException` object to handle a SIF request failure:

```
public static void main(String args[]) {
        UserTransaction tx = null;
        try {

            File file = new File("E:\\siperian-client.properties");
            System.out.println("Reading File:" + file.getAbsolutePath());
            if (!file.exists()) {
                System.out.println("***ERROR -> Properties File does not exist in
location - ");
                return;
            }
            EncryptionManager en = EncryptionManager.getInstance();
            try {
                en.init();
                //System.out.println("check :" +
en.isAPIFieldEncrypted("orcl.informatica.com-MDM_SAMPLE.BASE_OBJECT", "C_PARTY",
"FIRST_NAME"));
            } catch (IOException e) {
                // TODO Auto-generated catch block
                System.out.print("Exception \n" + e);
            }
            EjbSiperianClient sipClient = (EjbSiperianClient)
SiperianClient.newSiperianClient(file);

            tx.begin();
            System.out.println("TXN BEGAN");
```

```
            PutRequest putRequest1 = new PutRequest();


            System.out.println("Put 1");

            RecordKey recordKey1 = new RecordKey();
            recordKey1.setSourceKey("1000022");
            //recordKey1.setRowid("1970");
            recordKey1.setSystemName("Admin");

            Record record1 = new Record();
            record1.setSiperianObjectUid("BASE_OBJECT.C_PARTY");

            Field field1 = new Field();
            Field field2 = new Field();
            field1.setName("FIRST_NAME");
            field1.setStringValue("kk4");
            field2.setName("LAST_NAME");
            field2.setStringValue("kk4");
            Field field3 = new Field();
            field3.setName("PARTY_TYPE");
            field3.setStringValue("Person");
            record1.setField(field1);
            record1.setField(field2);
            record1.setField(field3);
            putRequest1.setRecord(record1);
            putRequest1.setRecordKey(recordKey1);
            System.out.println("Put 2");
            PutResponse putResponse1 = (PutResponse) sipClient.process(putRequest1);
            System.out.println("Record1 created for PUT - " + putResponse1);

            tx.commit();
            System.out.println("TXN COMMITTED");
        } catch (Exception name) {
            throw new SipRunTimeException("ERROR_CODE");
        } finally {
            System.out.println("Finally");
        }
    }
}
```

CHAPTER 5

ORS-Specific SIF API

This chapter includes the following topics:

- ORS-Specific SIF API Overview, 27
- Supported Repository Objects, 28
- ORS-Specific SIF API Properties, 28
- Repository Objects Statuses, 29
- Generating and Deploying an ORS-Specific SIF API, 29
- Renaming an ORS-specific SIF API, 30
- Downloading an ORS-Specific Client JAR File, 30
- Removing an ORS-Specific SIF API, 30
- Using ORS-Specific Client JAR Files with SIF SDK, 30
- Archive Table, 31
- ORS-Specific SIF Classes, 31
- ORS-Specific SIF API Field Parameters, 37

ORS-Specific SIF API Overview

You can generate SIF API for specific repository objects in an Operational Reference Store (ORS), such as base objects, packages, and cleanse functions. Use the SIF Manager utility in the Hub Console to generate an ORS-specific SIF API.

An ORS-specific SIF API acts on specific ORS objects. For example, an ORS-specific SIF API identifies data as a name and an email address and adds the data into a customer record, as defined in the ORS. A SIF API might place the same data in the database record that you specify.

You can use the SIF SDK to access the ORS-specific SIF API through the client JAR file or use the ORS-specific SIF API as SOAP web services. If you use the SIF SDK, install the Java Development Kit and the Apache Ant build system.

When you generate an ORS-specific SIF API, the performance of SIF API generation depends on the number of objects that you select, so select only the objects for which you want to generate the SIF API. The MDM Hub creates a unique version ID for an ORS-specific SIF API.

Note: The MDM Hub generates an ORS-specific SIF API only for the repository objects that are secure.

Supported Repository Objects

You can generate SIF API for specific repository objects that are secure. To secure an object, use the Secure Resources tool in the Hub Console.

You can generate ORS-specific SIF API for the following repository objects:

- Base objects
- Packages
- Mappings
- Cleanse functions
- Match columns
- Match rule sets

Note: When you generate APIs for match columns and match rule sets, ensure that you select the associated packages. If you do not select the associated packages, the MDM Hub does not generate the ORS-specific SIF API for the match columns and match rule sets.

ORS-Specific SIF API Properties

Use the SIF Manager utility in the Hub Console to configure the properties of an ORS-specific SIF API.

You can configure the following properties of an ORS-specific SIF API:

Logical Name

Logical name of the ORS.

You can edit the logical name and ensure that the edited name is unique. After you edit the logical name of an ORS-specific SIF API, regenerate and deploy the ORS-specific SIF API.

Java Name

Java name of the ORS.

The client JAR file name of the ORS-specific SIF API includes the Java name. Edit the logical name to change the Java name. Ensure that the edited logical name is unique.

WSDL URL

URL of the WSDL file that the MDM Hub generates when it deploys the ORS-specific SIF API.

API Generation Time

The date and time when you generate the ORS-specific SIF API. The format is `mm/dd/yy hh:mm tt`.

Version ID

Unique ID of the ORS-specific SIF API that the MDM Hub generates and deploys.

The MDM Hub uses the version ID in the following elements:

- Properties on the **Environment Report** and **ORS databases** tabs of the Enterprise Manager tool.
- Name of the client JAR file.
- The `MANIFEST.MF` file in the client JAR.

Repository Objects Statuses

The status of a repository object determines whether the MDM Hub can generate and deploy an ORS-specific SIF API for the repository object.

In the SIF Manager utility of the Hub Console, the **Selected and Out of Sync Objects** table displays the status of a repository object in the **Status** column. After you update an object, you can refresh the status of the object. To refresh the status of an object, on the **SIF API Manager** tab in the SIF Manager utility, click **Refresh Objects Status**.

A repository object can have one of the following statuses:

New

Indicates that the object is new and has no SIF API generated and deployed for it. If you generate and deploy an ORS-specific SIF API for the object, the status changes to `Up to date`.

Up to date

Indicates that the object has not changed after the SIF API generation and the object is up to date.

Out of sync

Indicates that the object has changed after the SIF API generation and the object is out of sync. Regenerate the SIF API to change the status to `Up to date`.

Not secure

Indicates that the object is not secure and you cannot generate SIF API for it. In the Secure Resources tool of the Hub Console, objects with the `Not secure` status appear as a private resource.

Deleted

Indicates that the object is deleted and you cannot generate API for it. In the Hub Console, if you use any tool in the Model workbench to delete an object, the status of the object becomes `Deleted`. When you generate and deploy an ORS-specific SIF API, the objects with the `Deleted` status are removed.

Generating and Deploying an ORS-Specific SIF API

Use the SIF Manager utility in the Hub Console to generate and deploy ORS-specific SIF API for a secure repository object. You can select specific repository objects to generate ORS-specific SIF API.

To generate and deploy ORS-specific SIF API, the MDM Hub requires access to a Java compiler on the computer that has application server installed. Ensure that you configure the base objects and packages of the ORS before you generate and deploy an ORS-specific SIF API.

1. Start the Hub Console, and connect to an ORS.
2. Expand the Utilities workbench, and click **SIF Manager**.

   The SIF Manager utility appears.
3. To acquire a write lock, on the **Write Lock** menu, click **Acquire Lock**.
4. To update the status of the repository objects, on the **SIF API Manager** tab, click **Refresh Objects Status**.

   The status of the repository objects is updated in the **Selected and Out of Sync Objects** table.
5. Select the repository objects for which you want to generate and deploy SIF API.
6. Click **Generate and Deploy ORS-Specific SIF APIs**.

   The ORS-specific client JAR file and a WSDL file are generated.

Renaming an ORS-specific SIF API

Use the SIF Manager utility in the Hub Console to rename an ORS-specific SIF API.

1. In the SIF Manager utility, on the **Write Lock** menu, click **Acquire Lock**.
2. On the **SIF API Manager** tab, in the **Logical Name** box, click the **Edit** button, and edit the logical name.
3. Click the **Accept** button.

   The logical name is saved, and the Java name is updated to match the logical name.
4. Click **Generate and Deploy ORS-Specific SIF APIs**.

   The ORS-specific client JAR file and the WSDL file are regenerated.

Downloading an ORS-Specific Client JAR File

After you generate an ORS-specific SIF API for the specific repository objects, download the client JAR file that contains the SiperianClient classes and SIF API reference documentation.

1. In the SIF Manager utility, on the **SIF API Manager** tab, click **Download Client JAR File**.

   The **Select the directory in which to save the client JAR file** dialog box appears.
2. Select the directory in which you want to save the client JAR file, and click **Save**.

   The `<Java Name>Client_<Version ID>.jar` file is downloaded and saved in the selected directory.

Removing an ORS-Specific SIF API

Use the SIF Manager utility in the Hub Console to remove an ORS-specific SIF API.

1. In the SIF Manager utility, on the **Write Lock** menu, click **Acquire Lock**.
2. On the **SIF API Manager** tab, click **Remove ORS-Specific APIs**.

   The ORS-specific client JAR file and the WSDL file are removed.

Using ORS-Specific Client JAR Files with SIF SDK

You can use the ORS-specific client JAR file with SIF SDK.

1. If you use an integrated development environment (IDE) and have a project file to build web services, add the downloaded client JAR file to the build class path.
2. Open the `build.xml` file in the following directory:
   - On Windows. `<Resource Kit Installation Directory>\hub\resourcekit\sdk\sifsdk`
   - On UNIX. `<Resource Kit Installation Directory>/hub/resourcekit/sdk/sifsdk`

3. Customize the `build.xml` file so that the `build_war` macro includes the downloaded client JAR file.
4. Save and close the `build.xml` file.

Archive Table

You can archive all the ORS-specific SIF APIs that you generate in the C_REPAR_SIF_ORS_CONFIG table stored in the CMX_DATA tablespace. Use the version ID of an ORS-specific SIF API to identify the archives.

The records in the archive table contain blob data that can be larger than the character-based records and can build up over time. The database administrator can archive or purge the archive table periodically to clean the database.

ORS-Specific SIF Classes

An ORS-specific SIF API can include the following classes:

- `Cleanse<Resource Name>`
- `CleansePut<Resource Name>`
- `Get<Resource Name>`
- `Put<Resource Name>`
- `SearchMatchColumn<Resource Name>`
- `SearchMatchRecord<Resource Name>`
- `SearchQuery<Resource Name>`

The classes depend on the type of the repository object that you select to create the ORS-specific SIF API. The classes are created for each repository object, and `Resource Name` indicates the name of the repository object that you select to create the ORS-specific SIF API.

Cleanse<Resource Name>

The `Cleanse<Resource Name>` request invokes the cleanse function defined in the MDM Hub to cleanse the input record. The response contains a record with the cleansed data.

Usage Examples

The following sample code uses the SOAP protocol to call the `Cleanse<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:mdmsample.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:cleanseFormatDate>
         <urn1:username>siftester</urn1:username>
         <urn1:password>
            <urn1:password>password</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <urn1:orsId>orcl-MDM_SAMPLE</urn1:orsId>

<urn:com.siperian.sif.dataobject.mdmsample.cleansefunctions.dataconversion.formatDateInpu
```

```
t>
            <urn:date>2014-03-17T00:00:00</urn:date>
            <urn:format>MMM dd, yyyy</urn:format>
         </
urn:com.siperian.sif.dataobject.mdmsample.cleansefunctions.dataconversion.formatDateInput
>
      </urn:cleanseFormatDate>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `Cleanse<Resource Name>` request:

```
<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>
      <cleanseFormatDateReturn xmlns="urn:mdmsample.siperian.api">
         <ns1:message xmlns:ns1="urn:siperian.api">The CLEANSE was processed
successfully.</ns1:message>

<com.siperian.sif.dataobject.mdmsample.cleansefunctions.dataconversion.formatDateOutput>
            <string>Mar 17, 2014</string>
         </
com.siperian.sif.dataobject.mdmsample.cleansefunctions.dataconversion.formatDateOutput>
      </cleanseFormatDateReturn>
   </soapenv:Body>
</soapenv:Envelope>
```

CleansePut<Resource Name>

The `CleansePut<Resource Name>` request cleanses a specified record and updates or inserts it into a base object in a single request.

Usage Example

The following sample code uses the SOAP protocol to call the `CleansePut<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:mdmsample.siperian.api" xmlns:urn1="urn:siperian.api">
  <soapenv:Header/>
   <soapenv:Body>
      <urn:cleansePutSfaAddress>
         <urn1:username>siftester</urn1:username>
         <urn1:password>
            <urn1:password>password</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <urn1:orsId>orcl-MDM_SAMPLE</urn1:orsId>
         <urn:com.siperian.sif.dataobject.mdmsample.mappings.sfaAddressInput>
            <urn:lastUpdateDate>2014-03-14T00:00:00</urn:lastUpdateDate>
            <urn:addressLine>2100 Seaport Blvd</urn:addressLine>
            <urn:cityName>Redwood City</urn:cityName>
            <urn:stateCode>CA</urn:stateCode>
            <urn:zip>94063</urn:zip>
         </urn:com.siperian.sif.dataobject.mdmsample.mappings.sfaAddressInput>
         <urn:generateSourceKey>true</urn:generateSourceKey>
      </urn:cleansePutSfaAddress>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `CleansePut<Resource Name>` request:

```
<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>
      <cleansePutSfaAddressReturn xmlns="urn:mdmsample.siperian.api">
         <ns1:message xmlns:ns1="urn:siperian.api">The CLEANSE PUT was processed
successfully</ns1:message>
```

```
            <recordKey>
               <ns2:systemName xmlns:ns2="urn:siperian.api">SFA</ns2:systemName>
               <ns3:rowid xmlns:ns3="urn:siperian.api">180161</ns3:rowid>
               <ns4:sourceKey xmlns:ns4="urn:siperian.api">415</ns4:sourceKey>
               <ns5:rowidXref xmlns:ns5="urn:siperian.api">180161</ns5:rowidXref>
            </recordKey>
            <actionType>Insert</actionType>
         </cleansePutSfaAddressReturn>
      </soapenv:Body>
   </soapenv:Envelope>
```

Get<Resource Name>

The `Get<Resource Name>` request retrieves a single row of data from the specified package. The row of data can include data from a base object and its associated cross-reference (XREF) records.

Usage Example

The following sample code uses the SOAP protocol to call the `Get<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:mdmsample.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:getPkgPerson>
         <urn1:username>siftester</urn1:username>
         <urn1:password>
            <urn1:password>password</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <urn1:orsId>orcl-MDM_SAMPLE</urn1:orsId>
         <urn:recordKey>
            <urn1:systemName>SFA</urn1:systemName>
            <urn1:sourceKey>300001035000</urn1:sourceKey>
         </urn:recordKey>
      </urn:getPkgPerson>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `Get<Resource Name>` request:

```
<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>
      <getPkgPersonReturn xmlns="urn:mdmsample.siperian.api">
         <ns1:message xmlns:ns1="urn:siperian.api">The GET was executed successfully -
retrieved 1 records</ns1:message>
         <recordKey>
            <ns2:rowid xmlns:ns2="urn:siperian.api">301952</ns2:rowid>
         </recordKey>
         <pkgPerson>
            <rowidObject>301952</rowidObject>
            <lastUpdateDate>2014-02-14T17:08:31.491-08:00</lastUpdateDate>
            <firstName>John</firstName>
            <lastName>Doe</lastName>
            <partyType>Person</partyType>
            <hubStateInd>1</hubStateInd>
         </pkgPerson>
      </getPkgPersonReturn>
   </soapenv:Body>
</soapenv:Envelope>
```

Put<Resource Name>

The `Put<Resource Name>` request updates or inserts a single record identified by a key into a base object.

Note: Use the `AddRelationshipRequest` and `UpdateRelationshipRequest` objects to add or update relationship records. If you use a `PutRequest` object or an ORS-specific `Put<Resource Name>` request to update the relationship records, the updates might result in incorrect relationship records.

Usage Example

The following sample code uses the SOAP protocol to call the `Put<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:mdmsample.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:putPkgPerson>
         <urn1:username>siftester</urn1:username>
         <urn1:password>
            <urn1:password>password</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <urn1:orsId>orcl-MDM_SAMPLE</urn1:orsId>
         <urn:recordKey>
            <urn1:systemName>SFA</urn1:systemName>
            <urn1:sourceKey>u12492746345</urn1:sourceKey>
         </urn:recordKey>
         <urn:pkgPerson>
            <urn:firstName>John</urn:firstName>
            <urn:lastName>Smith</urn:lastName>
            <urn:genderCd>M</urn:genderCd>
            <urn:displayName>Mr John Smith</urn:displayName>
            <urn:partyType>Person</urn:partyType>
            <urn:hubStateInd>1</urn:hubStateInd>
         </urn:pkgPerson>
         <urn:generateSourceKey>false</urn:generateSourceKey>
      </urn:putPkgPerson>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `Put<Resource Name>` request:

```
<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>
      <putPkgPersonReturn xmlns="urn:mdmsample.siperian.api">
         <ns1:message xmlns:ns1="urn:siperian.api">The PUT was processed successfully</
ns1:message>
         <recordKey>
            <ns2:systemName xmlns:ns2="urn:siperian.api">SFA</ns2:systemName>
            <ns3:rowid xmlns:ns3="urn:siperian.api">361930</ns3:rowid>
            <ns4:sourceKey xmlns:ns4="urn:siperian.api">u12492746345</ns4:sourceKey>
            <ns5:rowidXref xmlns:ns5="urn:siperian.api">361950</ns5:rowidXref>
         </recordKey>
         <actionType>Insert</actionType>
      </putPkgPersonReturn>
   </soapenv:Body>
</soapenv:Envelope>
```

SearchMatchColumn<Resource Name>

The `SearchMatchColumn<Resource Name>` request searches for records in a package based on the match column values. The response contains the matching records.

Usage Example

The following sample code uses the SOAP protocol to call the `SearchMatchColumn<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:datastewarddemo.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:searchMatchColumnPkgParty>
         <!--Optional:-->
         <urn1:username>admin</urn1:username>
         <!--Optional:-->
         <urn1:password>
            <urn1:password>admin</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <!--Optional:-->
      <urn1:securityPayload>cid:1224793701596</urn1:securityPayload>
         <!--Optional:-->
         <urn1:orsId>localhost-orcl-DS_UI1</urn1:orsId>
         <urn:sortCriteria></urn:sortCriteria>
         <urn:recordsToReturn>1</urn:recordsToReturn>
         <urn:returnTotal>true</urn:returnTotal>
         <urn:matchType>NONE</urn:matchType>
         <!--Zero or more repetitions:-->
         <urn:organizationName>?</urn:organizationName>
         <!--Zero or more repetitions:-->
         <urn:personName>John Doe</urn:personName>
         <!--Zero or more repetitions:-->
         <urn:addressPart1>?</urn:addressPart1>
         <!--Optional:-->
         <urn:matchRuleSetUid>?</urn:matchRuleSetUid>
         <!--Optional:-->
         <urn:disablePaging>?</urn:disablePaging>
      </urn:searchMatchColumnPkgParty>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `SearchMatchColumn<Resource Name>` request:

```
<q0:searchMatchColumnPkgParty>
…
<q0:personName>John Doe</q0:personName>
…
</q0:searchMatchColumnPkgParty>
```

SearchMatchRecord<Resource Name>

The `SearchMatchRecord<Resource Name>` request searches for match column values in a package based on the specified record. The response contains the match column values.

Usage Example

The following sample code uses the SOAP protocol to call the `SearchMatchRecord<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:datastewarddemo.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:searchMatchRecordPkgParty>
         <!--Optional:-->
         <urn1:username>admin</urn1:username>
         <!--Optional:-->
```

```
            <urn1:password>
               <urn1:password>admin</urn1:password>
               <urn1:encrypted>false</urn1:encrypted>
            </urn1:password>
            <!--Optional:-->
            <urn1:orsId>localhost-orcl-DS_UI1</urn1:orsId>
            <urn:sortCriteria>?</urn:sortCriteria>
            <urn:recordsToReturn>1</urn:recordsToReturn>
            <urn:returnTotal>true</urn:returnTotal>
            <urn:matchType>NONE</urn:matchType>
            <!--Zero or more repetitions:-->
            <urn:pkgOrganization>
            </urn:pkgOrganization>
            <!--Zero or more repetitions:-->
            <urn:pkgParty>
               <urn:partyType>?</urn:partyType>
            </urn:pkgParty>
            <!--Zero or more repetitions:-->
            <urn:pkgPerson>
               <!--Optional:-->
               <urn:firstName>John</urn:firstName>
               <!--Optional:-->
               <urn:lastName>Doe</urn:lastName>
               <urn:partyType>?</urn:partyType>
            </urn:pkgPerson>
            <!--Zero or more repetitions:-->
            <urn:dnbPartyInput>
            </urn:dnbPartyInput>
            <!--Zero or more repetitions:-->
            <urn:lgcPartyInput>
            </urn:lgcPartyInput>
         </urn:searchMatchRecordPkgParty>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `SearchMatchRecord<Resource Name>` request:

```
<q0:searchMatchRecordPkgParty>
…
<q0:contactPkg>
<q0:firstName>John</q0:firstName>
<q0:lastName>Doe</q0:lastName>
</q0:contactPkg>
</q0:searchMatchRecordPkgParty>
```

SearchQuery<Resource Name>

The `SearchQuery<Resource Name>` request searches for records in a package based on an SQL condition clause. The condition clause can reference any columns in the package and can use operators that the target database supports. The response contains matching records.

Note: To perform a search that is not case sensitive, set the `case.insensitive.search` property in the `cmxserver.properties.xml` file to true, and use the `SearchQuery<Resource Name>` request to specify a search criterion, such as lower(name)=lower('Jim').

Usage Example

The following sample code uses the SOAP protocol to call the `SearchQuery<Resource Name>` request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:datastewarddemo.siperian.api" xmlns:urn1="urn:siperian.api">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:searchQueryPkgParty>
         <!--Optional:-->
         <urn1:username>admin</urn1:username>
         <!--Optional:-->
         <urn1:password>
```

```
            <urn1:password>admin</urn1:password>
            <urn1:encrypted>false</urn1:encrypted>
         </urn1:password>
         <urn1:orsId>localhost-orcl-DS_UI1</urn1:orsId>
         <urn:recordsToReturn>10</urn:recordsToReturn>
         <urn:returnTotal>true</urn:returnTotal>
         <!--Zero or more repetitions:-->
         <urn:pkgParty>
            <!--Optional:-->
            <urn:displayName>ELIEZER MENDEZ</urn:displayName>
         </urn:pkgParty>
      </urn:searchQueryPkgParty>
   </soapenv:Body>
</soapenv:Envelope>
```

The following sample code displays the response to the `SearchQuery<Resource Name>` request:

```
<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>
      <searchQueryPkgPartyReturn xmlns="urn:datastewarddemo.siperian.api">
         <ns1:message xmlns:ns1="urn:siperian.api">The SEARCH QUERY REQUEST was
processed successfully</ns1:message>
         <pkgParty>
            <rowidObject>173</rowidObject>
            <displayName>ELIEZER MENDEZ</displayName>
            <partyType>Person</partyType>
         </pkgParty>
         <recordCount>1</recordCount>
      </searchQueryPkgPartyReturn>
   </soapenv:Body>
</soapenv:Envelope>
```

ORS-Specific SIF API Field Parameters

The following table lists the field parameters that the ORS-specific SIF classes use:

Note: The list of field parameters might vary based on your ORS.

Field Name	Type	Description
username	String	Optional. Name of the user who executes the request.
password	String	Optional. Password for the user name.
encrypted	Boolean	Optional. Indicates whether the password is encrypted. Specify true if the password is encrypted, or specify false if the password is not encrypted.
securityPayload	Byte	A security token or binary data used with a third-party authentication provider.
orsID	String	Optional. Identifier for the ORS.
interactionId	String	Optional. A unique identifier for the request.

Field Name	Type	Description
isAsynchronous	Boolean	Optional. Indicates whether to place the request on a JMS queue. Specify true to place the request on a JMS queue, or specify false to disable asynchronous processing.
jmsReplyTo	Boolean	Optional. Indicates whether to post the response on the specified JMS queue. Specify true to post the response on the specified JMS queue, or specify false if you perform synchronous processing.
jmsCorrelationId	Boolean	Optional. Indicates whether to set the JMS correlation ID to the specified value. Specify true to set the JMS correlation ID to the specified value, or specify false if you do not want to set the JMS correlation ID.
sortCriteria	String	List of column names, separated by commas, to order the results. The sortCriteria field is equivalent to the ORDER BY clause of an SQL query.
recordsToReturn	Int	Maximum number of relationship records to return.
returnTotal	Boolean	Indicates whether to return the total number of records that satisfy the search criteria. Specify true to return the total number of records that satisfy the search criteria, or specify false if you do not want to return the total number of records that satisfy the search criteria.
removeDuplicates	Boolean	Indicates whether to remove the duplicate records from the results. Specify true to remove the duplicate records, or specify false to retain the records.
matchType	MatchType object	Indicates the type of match rules that you want to apply from the match rule set. You can specify one of the following match types: - `BOTH`. Applies the automatic and the manual merge match rules. - `AUTO`. Applies the automatic merge match rules. - `NONE`. Applies the automatic and the manual merge match rules. If you selected exact-match columns in the **Selected Match Columns** list, and a selected record contains a value for that exact-match column, the match process identifies duplicate records based on the exact-match column plus the match rules.
matchRuleSetUid	String	Optional. ID of the match rule set that you want to use. Use null if you want to use the default match rule set.
disablePaging	Boolean	Indicates whether to disable paging. Specify true to disable paging to increase performance, and specify false to return a search token that is valid for 15 minutes.
systemName	String	Optional. Name of the source system.
sourceKey	String	Optional. The value of the PKEY_SRC_OBJECT column of the XREF record.
columnUid	String	UID of the Global Business Identifier (GBID) column.

Field Name	Type	Description
package	Boolean	Optional.
xref	Boolean	Optional.
pendingXref	Boolean	Optional. Indicates whether to return the pending XREF records. Specify true to return the pending XREF records, or specify false if you do not want to return the pending XREF records.
deletedXref	Boolean	Optional. Indicates whether to return the deleted XREF records. Specify true to return the deleted XREF records, or specify false if you do not want to return the deleted XREF records.
history	Boolean	Optional.
xrefHistory	Boolean	Optional. Indicates whether to return XREF history records. Specify true to return the XREF history records, or specify false if you do not want to return the XREF history records.
raw	Boolean	Optional.
returnTrustScores	Boolean	Optional. Indicates whether to return trust scores for the trust-enabled columns in the package and XREF records. Specify true to return the trust scores, or specify false if you do not want to return the trust scores.
returnLineage	Boolean	Optional. Indicates whether to return the lineage of the records. Specify true to return the lineage of the records, or specify false if you do not want to return the lineage of the records.
generateSourceKey	Boolean	Indicates whether to generate a source key if you do not specify the key in the record key. Specify true to generate a source key, or specify false if you specify the key in the record key.
lastUpdateDate	Date time	Optional. The date when you last updated the relationship record.

CHAPTER 6

Asynchronous SIF Requests

This chapter includes the following topics:

- Asynchronous SIF Requests Overview, 40
- Architecture of JMS Message Queue for SIF, 40

Asynchronous SIF Requests Overview

SIF uses a message-driven bean (MDB) on a Java Message Service (JMS) queue to process the asynchronous SIF requests from an external application. JMS queues are embedded message queues that use the JMS providers of the application servers. An embedded message queue uses the Java Naming and Directory Interface (JNDI) name of the connection factory and the queue name to connect with the JMS queue. The application server sets up the JNDI name of the connection factory.

The MDM Hub installer automatically sets up the following components for the specific application server that you use in your environment:

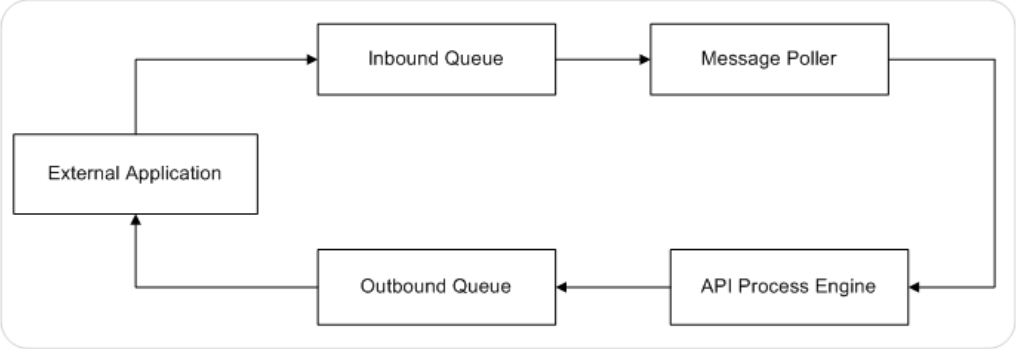
- Inbound message queue. `siperian.sif.jms.queue`
- Connection factory. `siperian.mrm.jms.xaconnectionfactory`

You must create an outbound message queue in the application server that has the MDM Hub installed. The MDM Hub processes the asynchronous SIF request and posts the response to the outbound message queue. The MDM Hub does not manage the outbound message queue. The external application retrieves the message from the outbound queue and processes it.

Note: The `siperian.sif.jms.queue` queue name is reserved for the MDM Hub. You cannot use reserved names when you create message queues.

Architecture of JMS Message Queue for SIF

The following image shows the architecture of JMS message queue and explains how an asynchronous SIF request is processed:



An external application sends a message containing a service invocation request to the `siperian.sif.jms.queue` inbound queue. The application server polls the queue for messages. The MDB of the Hub Server forwards the service request to the MDM Hub for processing. The MDM Hub processes the request and posts a response to the specified JMS outbound message queue. The external application retrieves the message from the specified message queue and processes it.

Processing Asynchronous SIF Requests

Use one of the following methods to process a SIF request asynchronously:

- Use an appropriately configured `AsynchronousOptions` object with a SIF request. When you run the SIF request, the SIF request is placed in an inbound queue for processing.
- Place the SIF request directly on the inbound message queue.

If you set the `jmsReplyTo` field, the response is posted to the specified JMS outbound queue. If you set the `jmsCorrelationId` field, the response includes the correlation ID, and you can identify the response based on the correlation ID that you set.

CHAPTER 7

ORS-Specific JMS Event Messages

This chapter includes the following topic:

- ORS-Specific JMS Event Messages Overview, 42

ORS-Specific JMS Event Messages Overview

Use the JMS Event Schema Manager tool that is part of the SIF Manager utility in the MDM Hub to generate and deploy ORS-specific JMS event messages for the current ORS. You can use the generated URL or download the XSD file to access the schema file for the JMS event messages. After you generate the schema file, you can configure the message triggers to identify the MDM Hub actions that you want to communicate to external applications and the message queue to publish the XML messages.
Note: You must have at least one secure package to generate an ORS-specific JMS event message schema.

The ORS-specific XSD file uses elements from the common XSD file, `siperian-mrm-events.xsd`. The format of the ORS-specific XSD file name is `<ORS Name>-siperian-mrm-event.xsd`.

The ORS-specific JMS event message schema file contains the following objects for each package:

Object Name	Description
`<Package Name>Event`	A complex type element that contains the `EventMetadata` and `<Package Name>` element types.
`<Package Name>Record`	A complex type element that represents a package and its fields and includes the `SipMetadata` element type. The complex type element resembles the package record structures defined in SIF.

If you have two databases that have the same schema such as `CMX_ORS`, use the unique database display name as the initial logical name instead of the database logical name. Otherwise, the logical name can be duplicated for the JMS events when you initially save the configuration.

Note: If you want to use the legacy XML event message objects, you do not require the ORS-specific JMS event message schema.

For more information about the JMS event messages, see the *Multidomain MDM Configuration Guide*.

Elements in a Response XML Message

The following table describes the elements in a response XML message:

Field	Description
siperianEvent	Root node in the response XML message.
eventMetadata	Root node for the event metadata.
messageId	Unique ID for the `siperianEvent` message.
eventType	Type of event. Use one of the following values: - Insert - Update - Update XREF - Accept as Unique - Merge - Merge Update - Unmerge
baseObjectUid	Unique ID of the base object that the event affects.
packageUid	Unique ID of the package associated with the event.
messageDate	Date and time when the message was generated.
orsId	ID of the ORS associated with the event.
triggerUid	Unique ID of the rule that triggered the event.
<Event Type>Event	Root node for the event details.
sourceSystemName	Name of the source system associated with the event.
sourceKey	Value of the PKEY_SRC_OBJECT column associated with the event.
eventDate	Date and time when the event was generated.
rowid	Row ID of the base object record that the event affects.
xrefKey	Root node of a cross-reference record that the event affects.
systemName	System name of the cross-reference record that the event affects.
sourceKey	Value of the PKEY_SRC_OBJECT column of the cross-reference record that the event affects.
<Package Name>	Name of the secure package associated with the event.
<Column Name>	Each column in the package is represented as an element in the XML file. For example, `rowidObject` and `consolidationInd`. The field is defined in the ORS-specific XSD file that you generate.
mergedRowid	List of ROWID_OBJECT values of the records that are merged. Applicable only for the merge events.

Sample Response XML Message for an Update Event

An update event generates the following sample response XML message:

```
<?xml version="1.0" encoding="UTF-8"?>
<siperianEvent>
  <eventMetadata>
    <eventType>Update</eventType>
    <baseObjectUid>BASE_OBJECT.CUSTOMER</baseObjectUid>
    <packageUid>PACKAGE.CUSTOMER_PKG</packageUid>
    <messageDate>2008-04-24T15:35:51.000-07:00</messageDate>
    <orsId>localhost-mrm-CMX_ORS</orsId>
    <triggerUid>MESSAGE_QUEUE_RULE.UpdateTrigger</triggerUid>
  </eventMetadata>
  <updateEvent>
     <sourceSystemName>TestSystem123</sourceSystemName>
     <sourceKey>123-1</sourceKey>
     <eventDate>2008-04-24T15:35:51.000-07:00</eventDate>
     <rowid>1</rowid>
     <xrefKey>
       <systemName>Admin</systemName>
       <sourceKey>SVR1.161</sourceKey>
     </xrefKey>
     <xrefKey>
       <systemName>System1</systemName>
       <sourceKey>2-1</sourceKey>
     </xrefKey>
     <customerPkg>
       <rowidObject>1</rowidObject>
       <creator>admin</creator>
       <createDate>2008-04-22T15:47:04.000-07:00</createDate>
       <updatedBy>admin</updatedBy>
       <lastUpdateDate>2008-04-24T15:35:50.000-07:00</lastUpdateDate>
       <lastRowidSystem>TESTSYSTEM</lastRowidSystem>
       <firstName>John</firstName>
       <lastName>Doe</lastName>
     </customerPkg>
 </updateEvent>
</siperianEvent>
```

CHAPTER 8

Using Security Access Manager

This chapter includes the following topics:

- Security Access Manager Workbench Overview, 45
- Using the Security Access Manager Workbench, 46
- Permissions for SIF Requests, 46

Security Access Manager Workbench Overview

Use the Security Access Manager workbench to configure a security framework for protecting the MDM Hub resources from unauthorized access. The security framework enforces the security policy decisions of your organization for your MDM Hub implementation and handles user authentication and access authorization at run time.

The security framework applies to the users of third-party applications who want to access the MDM Hub resources. The Hub Console has its own security mechanisms to authenticate users and authorize access to the Hub Console tools and resources.

If your application uses SIF requests to perform a task on your MDM Hub implementation that has the Security Access Manager workbench configured, ensure that you have appropriate permissions to access the MDM Hub objects.

You can also apply privileges at the column level. For example, the columns of the `P_CUST` package has the following privileges:

- READ privilege on column 1
- READ and CREATE privileges on column 2
- No privileges on column 3

A `GetRequest` object can access data only from column 1 and column 2 because the column 3 does not have the READ privilege. A `PutRequest` object can insert a record only in column 2 because column 1 and column 3 do not have the CREATE privilege. You cannot update any column because none of the columns has the UPDATE privilege.

Note: Only admin users can access private resources through the SIF requests.

Using the Security Access Manager Workbench

Use the Security Access Manager workbench to set permissions for a role and assign roles to a user.

1. Use the Secure Resources tool in the Security Access Manager workbench to configure a resource as secure. An external application can access only secure resources.
2. Use the Roles tool in the Security Access Manager workbench to define a role that can access the MDM Hub resources. A role represents a set of privileges to access secure MDM Hub resources.
3. Use the Users and Groups tool in the Security Access Manager workbench to associate the role with a specific user.

For more information about the Security Access Manager workbench, see the *Multidomain MDM Configuration Guide*.

Permissions for SIF Requests

The following table shows the permissions that the SIF requests require to access the MDM Hub resources:

SIF Request	Object	Permission
AcceptUnmatchedRecordsAsUnique	Package and base object	Update
AddRelationship	Hierarchy Manager profile	Create
ApplyChangeList	Repository Manager	Update
AssignUnmergedRecords	Package. Must grant permission to the ultimate parent base object column.	Update
Audit	Audit table	Update
Authenticate	-	None
CanUnmergeRecords	Package, base object, and cross-reference tables	Read
CleanTable	Repository Manager	-
Cleanse	Function and mapping	Execute
CleansePut	Base object, column, and mapping	Update and Create
ClearAssignedUnmergedRecords	Package and base object	Update
CreateChangeList	Repository Manager	Read
CreateTask	-	-
Delete	Base object if deleting a base object, cross-reference table if deleting a cross-reference table, and base object if deleting a base object and cross-reference record	Delete

SIF Request	Object	Permission
DeleteRelationship	Hierarchy Manager profile	Update
DescribeSiperianObject	Package, Repository Manager, base object, column, function, mapping, match rule set, and Hierarchy Manager profile	Read
ExecuteBatchAutoMatchAndMerge	-	Execute
ExecuteBatchAutomerge	-	Execute
ExecuteBatchBVTSnapshot	-	Execute
ExecuteBatchExternalMatch	-	Execute
ExecuteBatchGenerateMatchTokens	-	Execute
ExecuteBatchGroup	Batch group	Execute
ExecuteBatchKeyMatch	-	Execute
ExecuteBatchLoad	-	Execute
ExecuteBatchMatch	-	Execute
ExecuteBatchMatchAnalyze	-	Execute
ExecuteBatchPromote	-	Execute
ExecuteBatchRecalculateBO	-	Execute
ExecuteBatchRecalculateBVT	-	Execute
ExecuteBatchResetMatchTable	-	Execute
ExecuteBatchRevalidate	-	Execute
ExecuteBatchStage	-	Execute
ExecuteBatchSynchronize	-	Execute
ExecuteBatchValidateFKRelationships	-	Execute
GenerateConstraints	-	Merge
Get	Package, column, history, raw, cross-reference table, and cross-reference table history	Read
GetAssignableUsersForTasks	Record	Update
GetAssignedRecords	Package and column	Read
GetAssignedUnmergedRecords	Package and column	Read
GetBatchGroupStatus	Batch group	Read

SIF Request	Object	Permission
GetBVT	Package and column	Read
GetEntityGraph	Hierarchy Manager profile	Read
GetLookupValue	Column	Read
GetMatchedRecords	Package and column	Read
GetMergedHistory	Package and base object	Read
GetOneHop	Hierarchy Manager profile	Read
GetORSList	-	None
GetORSMetadata	Repository Manager	Read
GetSearchResults	Depends on the primary processor	Read
GetSiperianObjectCompatibility	Metadata	Read
GetSystemTrustSettings	Metadata	Read
GetTaskLineage	-	-
GetTrustGraphData	Metadata	Read
GetTrustScore	Column	Read
GetUnmergedRecordCount	Package and base object	Read
Link	Package	Merge
ListSiperianObject	Metadata	Read
Merge	Package	Merge
Multimerge	Package	Merge
PromotePendingXrefs	Package and column	Update
Put	Package and column	Update and Create
ReassignRecords	Package and base object	Update
RegisterCustomIndex	Repository Manager	-
RegisterCustomTableObject	Repository Manager	-
RegisterUsers	Users	Create
RemoveMatchedRecords	Repository Manager	-
ResetBatchGroup	Batch group	Update

SIF Request	Object	Permission
Restore	Base object and cross-reference table	Update
SearchHMQuery	Package, column, and Hierarchy Manager profile	Read
SearchLookupValues	Base object and column	Read
SearchMatch	Package and match rule set	Read
SearchQuery	Package and column	Read
SetPassword	-	None
SetRecordState	Package	Update
UpdateRelationship	Hierarchy Manager profile	Update
UpdateTask	-	-
Unlink	Package	Merge
Unmerge	Package	Merge
UnregisterUsers	Users	Update
Tokenize	Package	Update
ValidateChangeList	Repository Manager	Read
ValidateMetadata	Repository Manager	Read
ValidateTask	-	-

CHAPTER 9

Using Dynamic Data Masking

This chapter includes the following topics:

- Dynamic Data Masking Overview, 50
- Rules, 50
- Supported SIF Requests for Dynamic Data Masking, 51

Dynamic Data Masking Overview

Informatica Dynamic Data Masking is a data security product that operates between a client and a database to prevent unauthorized access to sensitive information. Dynamic Data Masking intercepts requests sent to the database and applies data masking rules to the request to mask the data before it is sent back to the client.

You can use Dynamic Data Masking to mask or prevent access to sensitive data stored in production and non-production databases. You set up the rules to specify the database requests to intercept and the masking actions to apply. Dynamic Data Masking monitors incoming database requests from the MDM Hub. Dynamic Data Masking applies the data masking rules to the database request before it sends it to the database. The database processes the modified request as normal and returns masked results to Dynamic Data Masking. Dynamic Data Masking then sends the results to the MDM Hub.

You can use Dynamic Data Masking to mask data for specific types of database requests or you can restrict access to data from certain groups within an organization. For example, you can create a rule to apply a masking function to credit card numbers when the database request comes from a support team member. When Dynamic Data Masking sends the data back to the MDM Hub, the support team member sees the masked numbers instead of the real credit card numbers.

Rules

A rule contains conditions and actions that Dynamic Data Masking uses to process a request.

A connection rule defines the connection criteria to identify a connection and the target database. A security rule defines the criteria to parse and rewrite an SQL request. Use security rules to mask data in a specific row or to mask an entire column. For example, you can create a security rule that rewrites SQL requests that reference the Social Security column from the Employee table.

Supported SIF Requests for Dynamic Data Masking

After you add a database that the MDM Hub implementation uses to Dynamic Data Masking, you can configure security rules to mask data in a specific row or column. If you use SIF requests to retrieve data from a security rule-enabled column or row, the responses might include masked data based on the security rule that you configure.

The following SIF requests support the data masking rules:

- Get
- GetAggregatePeriod
- GetEffectivePeriods
- GetEntityGraph
- GetLookupValue
- GetMatchedRecords
- GetOneHop
- GetSearchResults
- GetXrefForEffectiveDate
- SearchHmQuery
- SearchLookupValues
- SearchMatch
- SearchQuery
- PreviewBvt

For more information about configuring Dynamic Data Masking for the MDM Hub, see the *Multidomain MDM Security Guide*.

CHAPTER 10

SIF API Reference

This chapter includes the following topics:

- Functional SIF API Listing, 52
- Reference SIF API Listing, 56

Functional SIF API Listing

The following table describes SIF API requests organized by function:

SIF Functional Group/Class	Description
Batch Group APIs	Batch Group API requests enable developers to run batch groups directly without using the MDM Hub Console or stored procedures.
"ExecuteBatchGroup" on page 77	Runs a set of batch jobs, some sequentially, and some in parallel according to the configuration.
"GetBatchGroupStatus" on page 93	Get status of most recent execution; polls for status after executing asynchronously.
"ResetBatchGroup" on page 126	Finds the status of the last run of a batch group, and if the status is failed, sets it to incomplete.
Data Steward APIs	Data Steward API requests facilitates developers to write applications with a custom user interface. You can use any SIF API request that your application requires.
"GetLookupValue" on page 97	Retrieves the lookup display name, lookup code description, for the specific lookup values, lookup codes, for the specified lookup columns.
"GetLookupValues" on page 98	Retrieves the list of valid lookup values (lookup codes) and lookup display names (lookup code descriptions) for the specified lookup columns.
"GetMatchedRecords" on page 98	Retrieves the match candidates for the specified record.
"GetMergeHistory" on page 99	Retrieves a tree that represents the history of merges for a specified base object record.

SIF Functional Group/Class	Description
"GetSystemTrustSettings" on page 103	Retrieves the system-specific trust settings for the specified columns.
"GetTrustGraphData" on page 109	Retrieves the data to plot the trust decay curve for the specified trust setting.
"GetTrustScore" on page 109	Retrieves the current trust score for the specified column in a base object record.
"GetXrefForEffectiveDate" on page 110	Retrieves multiple XREF records for the specified effective date.
" PreviewBVT" on page 115	Provides a preview for a base object record if a specified set of records are merged or pending updates are applied.
"SearchLookupValues" on page 128	Searches for lookup values that match a lookup display name, lookup code description.
"SetRecordState" on page 136	Sets the record state of base object records identified by the specified keys.
Data APIs	Data API requests enable developers to run the MDM Hub Cleanse, Link, MultiMerge, and Unlink base object requests.
"Cleanse " on page 61	Uses cleanse functions to transform an input record provided in the request to the output format specified by the cleanse function selected.
"Link" on page 112	Links two or more base object records using the specified groupRecordKey as the group ID.
"MultiMerge" on page 114	Merges multiple base object records that are identified as representing the same object and you can specify the field level overrides for the merged record.
"Unlink" on page 139	Unlinks two or more base object records with the group ID specified in the groupRecordKey field.
Data Update / Insert APIs	Data update or insert API requests enable developers to run data updates and inserts on base object records.
"AddRelationship" on page 57	Adds a relationship between two entities.
"CleansePut" on page 62	Inserts or updates a single record identified by a key into a base object.
"DeleteRelationship" on page 71	Deletes a relationship between two entities by changing the Hub state to deleted. DeleteRelationship does not remove the record from the relationship table. If the relationship is a foreign key relationship, the request sets the foreign key value to null.
"Merge " on page 114	Merges two base object records that are identified as representing the same object.
"Put " on page 118	Inserts or updates a single record identified by a key into a base object.

SIF Functional Group/Class	Description
"Tokenize " on page 138	Generates match tokens for a base object record that is updated or inserted.
"Unmerge " on page 139	Unmerges base object (BO) records.
"UpdateRelationship" on page 142	Hierarchy Manager request for changing some characteristics of an existing relationship.
Data Retrieval APIs	Data Retrieval API requests enable developers to retrieve data, including BVT, a single record or sets of records, as well as to perform searches based on match columns.
"GetBvt" on page 94	Retrieves the best version of truth (BVT) from the specified package using a known key.
"Get" on page 88	Retrieves a single record from the specified package using a known key.
"GetEntityGraph" on page 96	Hierarchy Manager request for fetching a graph of entities and relationships related to a specified set of entities.
"GetOneHop" on page 99	Hierarchy Manager request for fetching information about the entities directly related to a specified group of entities in a specified HM configuration.
"GetSearchResults " on page 101	Retrieves additional data when the number of records found by the SIF API search queries, SearchMatch and SearchQuery, exceeds the number of records to return specified in the search API request.
"SearchHmQuery" on page 128	Provides search capabilities for Hierarchy Manager.
"SearchMatch " on page 128	Searches for records in a package based on match columns and rule definitions.
"SearchQuery " on page 133	Retrieves a set of records from an MDM package satisfying the specified criteria.
Merge Workflow APIs	Merge Workflow API requests enable developers to run post-match batch processes, such as search for unmatched or unmerged records.
"AcceptUnmatchedRecordsAsUnique" on page 56	Once the match batch process is run and records are placed into match groups, there are often records that did not match any other records in the Hub. Sets the unmatched records to unique (that is, sets CONSOLIDATION_IND=1)
"AssignUnmergedRecords" on page 58	After the match batch process has been run and records are placed into match groups, the records that were processed and not automatically merged are placed into the UNMERGED state. This is to assign the unmerged records to specified user.
"CanUnmergeRecords" on page 60	Determines whether the specified cross reference (XREF) record can be unmerged from the consolidated base object.
"ClearAssignedUnmergedRecords" on page 66	Clears the list of unmerged records that are currently assigned to this user.

SIF Functional Group/Class	Description
"GetAssignedRecords" on page 93	Get a set of records requiring manual merge decisions that are assigned to the user.
"GetUnmergedRecordCount" on page 110	Get the number of unmerged records.
"ReassignRecords" on page 123	Reassigns the specified records assigned for manual merge evaluation to another user.
Metadata APIs	Metadata API requests enable developers to return metadata for specified objects.
"DeleteRelationship" on page 71	Request to describe Informatica objects by fetching their metadata.
"GetOrsList" on page 100	Retrieves a list of operational record stores (ORS) registered in the master database.
"ListSiperianObjects " on page 112	Returns metadata of Informatica MDM Hub objects.
"DescribeSiperianObject" on page 71	Returns metadata of Informatica MDM Hub objects.
Metadata Management APIs	Metadata Management API requests enable developers to manage ORS change lists.
"ApplyChangeList" on page 57	Applies a change list to the current repository.
"CreateChangeList" on page 66	Creates a change list in XML format for the current repository.
"ValidateChangeList" on page 145	Validates a change list against the current repository.
"ValidateMetadata" on page 146	Validates the metadata for the current repository.
Repository Manager APIs	Repository Manager API requests enable developers to export metadata.
"GetOrsMetadata" on page 101	Export metadata to a change list XML file.
State Management APIs	State Management API requests enable developers to delete and restore state-enabled records with the state set to DELETE, additionally promotes pending cross-reference records.
"Delete" on page 69	Deletes the specified records from the MDM Hub.
"PromotePendingXrefs" on page 116	Promotes or flags for promotion the XREF records specified in the request.
"Restore" on page 127	Restores the specified XREF records in the MDM Hub.
Task APIs	Task APIs are used for task administration.
"CreateTask" on page 67	Creates a task.
"GetTasks " on page 107	Retrieves lists of tasks and task details.
"GetTaskLineage" on page 104	Retrieves the lineage of the specified task.

SIF Functional Group/Class	Description
"GetAssignableUsersForTasks" on page 92	Retrieves a list of users whom you can assign a list of specified tasks.
"UpdateTask" on page 143	Updates a task.
"ValidateTasks" on page 147	Checks each merge task specified in the request to verify there is a match table record
User Management APIs	User Management API requests enable developers to manage user security.
"Authenticate" on page 60	Authenticates a user against the specified ORS.
"SetPassword " on page 136	Changes a user password to a new password.
Miscellaneous APIs	These miscellaneous API requests enable developers to run audit requests, register and unregister users, and perform other compatibility requests.
"Audit " on page 59	Add a custom entry to the Hub Audit trail.
"GetSiperianObjectCompatibility" on page 103	Request to get a checksum that represents the definition of the specified object in Informatica MDM Hub.
"RegisterUsers" on page 125	Allows for automated provisioning of users that are authenticated externally using one of the registered JAAS login modules.
"UnregisterUsers" on page 141	Allows for previously provisioned users (see RegisterUsers) to be unregistered.

Reference SIF API Listing

This section is an alphabetical listing for the SIF API. It provides a description and use case examples for the various SIF API requests. Refer to the SIF Javadocs for details of how to use these API requests with the interfaces that the Informatica Java client provides. If you are using a Web service interface to the requests, refer to the Web Services Description Language (WSDL) descriptions of the Informatica Web service.

Note: Only admin users can access private resources through SIF requests.

AcceptUnmatchedRecordsAsUnique

AcceptUnmatchedRecordsAsUnique changes the state of records that have no match candidates from Unmerged to Consolidated (unique). Once a record is in the Consolidated state, it will no longer appear in the list of records that needs to be reviewed and it will not be merged by the merge batch process. These records can still be merged manually in the Console or by using the Merge API.

The request specifies the base object table or a package on that table. It also supplies a boolean value indicating whether or not to change only those records assigned to the user.

The response contains the number of records accepted as unique.

Note: You can configure AcceptUnmatchedRecordsAsUnique requests only for "no system" when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the AcceptUnmatchedRecordsAsUnique request:

- **Set unmatched records as unique** — You can use the AcceptUnmatchedRecordsAsUnique request in an application with a custom UI for the data stewards. In the screen that manages the status of records, you might create a button that uses this request to accept all the unmatched records as unique.

AddRelationship

AddRelationship enables you to add a relationship between two entities.

The request identifies the HM configuration and hierarchy, the relationship type, the records, and a number of optional parameters. Note that this request cannot be used to add a new Relationship with Foreign Key Relationship Type because adding a FK Relationship really involves updating an existing record in the FK Relationship Base Object. For more information, see "UpdateRelationship" on page 142.

The response contains the record key for the added relationship. Informatica MDM Hub infers the types of the entities being related (and thus the base objects containing those entities) from the relationship type.

Use Case

This is the common scenario for using the addRelationship request:

- **Add a relationship between two HM entities** — If you have Hierarchy Manager and have populated it with entities, you can use the addRelationship request to create a relationship between two entities.

Related SIF Requests

- "UpdateRelationship" on page 142
- "DeleteRelationship" on page 71

ApplyChangeList

The **ApplyChangeList** request applies all the changes in the specified change list to the current repository (ORS).

Required Parameters

The following table describes the required parameters:

Parameter	Description
ChangeListXml	Contains the XML string representing the change list to apply.

Optional Parameters

The following table describes the optional parameters:

Parameter	Description
RollBackStrategy	If set to `FULL_ROLLBACK`: No changes are applied if an error occurs during the change list process. The default is `FULL_ROLLBACK`. If set to `ROLLBACK_TO_LAST_CHANGE`: Only the change list item that failed is rolled back. All other changes are applied.
OwnerPassword	Contains the owner password. The default is `""`.
ValidateDataIntegrity	If set to `true`, data integrity validation is required. If set to `false`, data integrity validation is not required. The default is `false`.

Response Field

The following table describes the response fields:

Field	Description
Messages	Contains an array of error messages.
Success	If `true`, the change list executed without errors. If `false`, the change list executed with errors.
DataLost	If `true`, data was lost because of a rollback. If `false`, no data was lost because of a rollback.

AssignUnmergedRecords

AssignUnmergedRecords assigns records in the unmerged state to the specified user. It assigns no more than the requested number of records. Optionally, you can specify a WHERE clause to select Unmerged records from the package. The Unmerged state is equivalent to setting the consolidation indicator to 2 and can also be referred to as the “ready to merge” state. Records are placed into the Unmerged state regardless of whether they matched other records or not. This request is used to assign the records that are in the Unmerged state to a specified user for review and processing. However, any records that are already assigned to a user will not be reassigned by this API.

The response contains the number of records assigned.

Note: Hub Implementers can setup user exits that control how records are assigned. These user exits are invoked when this API is run and will override the standard logic for assignment of records. For information regarding user exits, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the AssignUnmergedRecords request:

- **Assigning unmerged records to a user**—You can use the AssignUnmergedRecords request in an application with a custom UI for the data stewards. In the screen that manages the data steward’s queues, you might create a button that uses this request to assign unmerged records.

Audit

Audit adds an entry to the C_REPOS_AUDIT table to record information about some activity involving a record stored in Informatica MDM Hub. You can log similar information about information in your own application programs.

Set the attributes of the new entry (for example, component, action, status, context). Then process the request to add the entry to the audit table. The process method returns an AuditResponse, which contains the rowid of the resulting audit record.

To use this facility, store the name of a project or similar large entity in component, and let action be an element of the component. For example, component might be “SIF API” and action might be AuditRequest.

You can set the audit rowid of the last previous related audit entry. In this way you can build a chain of audit entries. You obtain the rowid of an audit entry from the AuditResponse that comes back when you process an AuditRequest.

Use the status field to convey information useful for determining what to do with the audit record. For example, status values might be debug, info, warn, error, and fatal.

Use the contextXML and dataXML to add XML-formatted additional information to the audit entry.

Note: You can not configure Audit API requests to audit requests made by external applications. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the Audit request:

- **Adding auditing information to the log**—You can use the audit request in an application to record auditing information in the log for reporting or compliance purposes.

Usage Example

```
// For example, if this is in a Servlet that receives an XML
// to update multiple Hub packages.

AuditRequest request = new AuditRequest();
request.setComponent("mycompany.customerServlet");
request.setAction("POST");
request.setStatus("info");
// from: the same system to be used in other SIF calls
request.setFromSystem("CRM");
request.setToSystem("Admin"); // to: Siperian Hub

// context: any metadata to help understand the entry
request.setContext( dataId ); // example: pkeySource
// context xml: complex metadata, for debug, may impact performance
request.setContextXML("<metadata>"
+ "<url>" + httpServletRequest. getRequestURI() + "</url>"
+ "</metadata>");

// It may be helpful to identify the root package
request.setSiperianObjectUid(
SiperianObjectType.PACKAGE.makeUid("CUSTOMER_UPDATE") );

// data xml: usually for debug only, may impact performance
request.setDataXML( requestXmlAsString );

// If there was a related audit before this one:
request.setRowidAuditPrevious(prevAuditResponse.getRowidAudit());

// If the rowid_object is known:
request.setRowidObject("");

AuditResponse response = (AuditResponse)
sipClient.process(request);
```

```
// Now decompose the request data and call other SIF API's ...
```

Authenticate

Authenticate allows you to determine a user's rights to access an ORS. If the user has the right to access the ORS, the message in the response object is STATUS_GRANTED. Otherwise it is STATUS_DENIED. The response contains a list of the roles assigned to the user and information about the user's password—if and when it expires and whether it is externally authenticated using a service such as LDAP.

Use Case

This is the common scenario for using the authenticate request:

- **Determine a user's access rights to an ORS**—Before using a request that requires specific access privileges, you can use authenticate to determine if the user possesses the required rights.

CanUnmergeRecords

CanUnmergeRecords determines whether or not specified records can be unmerged from the consolidated base object. The request contains a package and a key identifying the XREF to unmerge. The response contains a boolean value that is true if the records can be unmerged, false if they cannot.

Cross reference records can be added to a base object record either by consolidating two base object records or by adding them directly using the ROWID_OBJECT of a base object record. If a cross reference is added using the ROWID_OBJECT and no PKEY_SOURCE_OBJECT, and there is not already a cross reference for that base object record for the specified system, a new cross reference record is added that is considered an "edit" cross reference.

An unmerge is not allowed if the specified cross reference is not an edit cross reference and all the other cross references for that base object are edit cross references. If there are at least two cross references that are not edit cross references, the cross reference can be unmerged.

Note: You can configure CanUnmergeRecords requests according to a specific system when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the CanUnmergeRecords request:

- **Determining whether a given record can be unmerged**—You can use the CanUnmergeRecords request in an application with a custom UI for the data stewards to determine whether two records can be unmerged before attempting to do so.

CleanTable

A CleanTable request removes data from an Operational Reference Store table and all its companion tables.

Request Parameters

A CleanTable request contains the following parameters:

SiperianObjectUid

Identifier of a base object or a table from which you want to remove data.

CleanStaging

Optional. Indicates whether to remove rows in the staging tables and the dependent tables such as `<Staging Table>_CL`, `<Staging Table>_DLT`, `<Staging Table>_RAW`, `<Staging Table>_REJ`, `<Staging Table>_OPL`, and `<Staging Table>_PRL`.

The CleanStaging parameter uses the following values:

- True. Removes the rows in the staging tables and the dependent tables.
- False. Retains the rows in the staging tables and the dependent tables.

Default is false.

UseTruncate

Optional. Indicates whether to use the TRUNCATE or DELETE statement to remove the records.

The UseTruncate parameter uses the following values:

- True. Uses the TRUNCATE statement, which functions faster, to remove the records. In the Oracle-distributed transaction environments, the TRUNCATE statement commits transactions that might cause errors.
- False. Uses the DELETE statement, which functions slower, to remove the records.

Default is false.

Response Fields

The CleanTable request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

Usage Example

The following code sample uses the TRUNCATE statement to remove data from the base object C_PARTY and the associated staging table:

```
CleanTableRequest request = new CleanTableRequest();
request.setSiperianObjectUid("BASE_OBJECT.C_PARTY");
request.setCleanStaging(true);
request.setUseTruncate(true);
CleanTableResponse response = (CleanTableResponse)sipClient.process(request);
```

Cleanse

Cleanse invokes a cleanse function defined in Informatica MDM Hub. The request specifies the record and the cleanse function. The response contains a record containing the cleansed data.

Available cleanse functions can be viewed in the Hub Console in the Cleanse Function Manager. Additionally, you can use the ListSiperianObject request to retrieve the list of available cleanse functions. Cleanse function details, including parameters, can be retrieved using the DescribeSiperianObject request.

You can specify the name of the cleanse function to use in a Cleanse request in two ways:

- a cleanse function UID: “CLEANSE_FUNCTION.[Cleanse Library Name]|[Cleanse Function Name]”
- or “[Cleanse Library Name]|[Cleanse Function Name]”.

For example, in order to use the Concatenate cleanse function that resides in the String Functions cleanse library, the cleanse function would be identified as either “CLEANSE_FUNCTION.String Functions|Concatenate” or “String Functions|Concatenate”.

Mappings defined in the Hub may also be accessed and used as cleanse functions. Mappings are automatically placed in the “Mappings” library and can be accessed using the UID “CLEANSE_FUNCTION.Mappings|[mapping name]”.

Related Topics:

- “CleansePut” on page 62

Use Cases

These are the common scenarios for using the cleanse request:

- **Data cleansing for external applications** — An external application can use the cleanse request independently of the Informatica MDM Hub master record functionality. External applications can invoke cleanse to interface with data quality facilities provided by Informatica to process input data.
- **Address verification for external applications** — Informatica MDM Hub provides the functionality to validate and standardize addresses. These facilities can be used by external applications to improve the quality of the address data that is entered into them.
- **Cleanse used in combination with put** — The most common use of the cleanse request is to cleanse an individual field before the record is passed to the put request.
- **Cleanse used in combination with match** — The match request provides access to the matching rules and allows you to search Informatica MDM Hub for records that contain values that are similar, but not necessarily identical to the search criteria. To improve the quality of matches returned, you can cleanse the search criteria before passing them to the match request.

Related SIF Requests

“CleansePut” on page 62

CleansePut

CleansePut cleanses the specified record and updates or inserts the record into the specified table in a single request. **CleansePut** replicates the Stage and Load batch processes that move data from the landing table, through the cleansing process, into the staging table, and into the base object. **CleansePut** can also perform the lookups required to translate source system foreign keys into Hub foreign keys. The physical landing and staging tables are not used by **CleansePut**.

The record is put into the base object based on a mapping, which defines the transformation of data from a landing table structure to a staging table structure. The staging table associated with the mapping determines which base object the resulting data is inserted or updated in.

You can configure **CleansePut** requests in all systems when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, the MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Filtered Requests

A **CleansePut** request can be filtered so that no changes are made in the ORS. If **CleansePut** is filtered, an ActionType of `No Action` is returned in the **CleansePut** response. **CleansePut** can be filtered in two ways:

- Filtered by the mapping. The mapping can include a condition that must be true before allowing **CleansePut** to process a record.
- Filtered by delta detection. In the Hub, you can enable delta detection on the staging table. With delta detection enabled, **CleansePut** requests are filtered if the data does not differ from the data in the previous request.

State Management

If a package has state management enabled, you can specify a record's initial state when you insert a record by setting the value of HUB_STATE_IND. When you insert a new record and do not specify a HUB_STATE_IND value, the HUB_STATE_IND is set to 1 (ACTIVE). You cannot use **CleansePut** to change the state of a record by updating the HUB_STATE_IND value. State management is enabled in the Hub Console.

The possible values for HUB_STATE_IND and the state these values represent are outlined in the following table:

HUB_STATE_IND Value	State
1	ACTIVE
0	PENDING
-1	DELETED

Transaction Support

When executed within an EJB context, this request can be part of a transaction with other requests. If there is a failure in any of the requests within a transaction, the entire transaction is rolled back.

Restrictions

Consider the following restrictions when using the **CleansePut** API.

- Special characters do not need to be escaped before making the **CleansePut** API call. However, if you have custom code that used escaped special characters in the past, you must update your custom code to remove the escaped special characters.
- Both **Put** and **CleansePut** requests process null values. For example, when no value is specified for a field, the field is set to null. However, **CleansePut** does not process records that contain a reference not found in a lookup table.
- You cannot insert a null value into a nonnullable column, such as a unique key column. You must provide a value for nonnullable columns because empty fields are set to null.
- You cannot use **CleansePut** to insert or update a read-only column.
- You cannot use **CleansePut** to insert or update a system column unless it is enabled in the Hub to be Putable. See the Column Properties in the *Multidomain MDM Configuration Guide* for information about which system columns can be putable.
- You can specify a value for HUB_STATE_IND when inserting a new record, but you cannot change the state of an existing record by changing the HUB_STATE_IND value using the **Put** API. If you provide a value for the HUB_STATE_IND column when updating a record, the **Put** API throws an exception. To

change the state of a record, refer to the following classes: "Delete" on page 69, "Restore" on page 127, and "PromotePendingXrefs" on page 116.

- If you use special characters like ' and ~ in CleansePut calls, you must escape them with a backslash character.
- If the foreign key column for a child base object is not specified or is specified as NULL in the **CleansePut** request, the lookup is on the parent key of the foreign key column instead of the lookup column defined on the staging table.
- If you use a CleansePut call to insert a lookup into a base object, the lookup will be case-sensitive. For example, the Gender column in the lookup C_LU_GENDER|GENDER_CD only accepts values of M, F, and UNK. Lowercase values (m, f, or unk) in the Gender column are not accepted.
- When an omitted field is mapped from the landing table to the staging table through a cleanse function, the MDM Hub sets the field to NULL.

Required Parameters

The following table lists and describes the parameters that are required by the **CleansePut** API:

Parameter	Description
Record	This parameter contains the data to be cleansed and inserted.
SiperianObjectUid	Name and type of mapping to use in the **CleansePut** request. The mapping defines the structure of the record.

Optional Parameters

The following table lists and describes the optional parameters that are used by the CleansePut API:

Parameter	Description
SystemName	The system name of the record to be cleansed. A staging table is associated with a source system. If the system name is not specified by this parameter, the staging table's source system is used.
GenerateSourceKey	Useful for keyless systems (for example, an application that does not persist source data). When set to `true`, a source key is generated if one is not already specified.
PeriodStartDate	Specifies the period start date for timeline-enabled base objects.
PeriodEndDate	Specifies the period end date for timeline-enabled base objects.
PeriodReferenceTime	Applicable for base objects for which you track data change events. Specifies a reference date within an effective period to identify a record version that you want to update. Default is null.

Parameter	Description
timelineAction	Applicable for base objects for which you track data change events. Specifies the action to perform on a record version during the load process. Use one of the following values: - 0. Adds a record version for a new effective period without maintaining contiguity between the record versions. - 1. Updates data in an existing record version. The effective period of the record does not change. - 2. Updates the effective period of a record version. An update to an effective period of a record version is through an increase or decrease of the effective start or end date. - 4. Adds a record version for a new effective period while maintaining contiguity between the record versions. Default is 0.
isFillOnGap	Applicable for base objects for which you track data change events. Ensures that contiguity between the effective dates of record versions is maintained when you add new record versions. If set to `true`, when you can add a new record version to the base object, the MDM Hub maintains the contiguity between effective periods of record versions. If set to `false`, the MDM Hub rejects any addition of record version that breaks the contiguity between effective periods of record versions. Default is false.

Response Fields

The **Put** response can contain the information described in the following table:

Field	Description
RecordKey	Contains the ROWID_OBJECT of the base object affected by **CleansePut**. When performing a **CleansePut** request using a ROWID_OBJECT for a base object record that has been merged into another base object record, **CleansePut** response returns the ROWID_OBJECT of the surviving base object record. RecordKey also contains a new primary key created by the key generator if GenerateSourceKey in the request was set to `true`.

ActionType	Indicates the action that the **Put** performed. The possible values are:
	`- Insert`
	`- Update`
	`- Update XREF`
	`- No Action`
	Tokenize requires the value of ActionType. `Insert` indicates that a record has not yet been tokenized and new tokens need to be created. `Update` and `Update XREF` indicate that a record has already been tokenized and the existing tokens need to be regenerated.

Use Case

The following is a typical scenario for using the CleansePut request:

- Cleanse a record and update or insert it in the specified table. You can cleanse a specified record and update or insert it in the specified table in a single request. This increases performance, when compared to doing the a **Cleanse** and then a **Put**, by reducing round trips between the client and the MDM Hub.

- **CleansePut** used in combination with **Tokenize**. **CleansePut**, followed by **Tokenize**, cleanses the new row of data, inserts or updates it in the base object, and then encodes the base object so it is ready for matching. The **CleansePut** response contains an ActionType value used as an input to the **Tokenize** request. **CleansePut** and **Tokenize** can occur in the same transaction.

Usage Example

The following example shows how a record with ROWID_OBJECT key 782 is updated by using the mapping, Stage CRM Address:

```
CleansePutRequest request = new CleansePutRequest();Record record = new Record();
record.setSiperianObjectUid("MAPPING.Stage CRM Address");
record.setField( new Field("ADDRESS_ID", "782") );
record.setField( new Field("ADDRESS_LINE", "123 Main St.") );
record.setField( new Field("CITY_NAME", "Anytown") );
record.setField( new Field("LAST_UPDATE_DATE", new Date()) );
request.setRecord( record );
CleansePutResponse response = (CleansePutResponse) sipClient.process(request)
```

Related SIF Requests

"Cleanse " on page 61, "Put " on page 118, "Tokenize " on page 138

ClearAssignedUnmergedRecords

ClearAssignedUnmergedRecords clears a user's assigned unmerged records for the specified base object, making those records available for assignment to another user.

Note: There are no parameters for this request. All the unmerged records assigned to this user making the request will now be available to be assigned to another user. If there is a specific user that the records should be assigned to the ReassignRecordsRequest should be used.

Use Case

This is the common scenario for using the ClearAssignedUnmergedRecords request:

- **Clearing the queue of unmerged records to a given user**—You can use the ClearAssignedUnmergedRecords request in an application with a custom UI for the data stewards. In the screen that manages the data steward's queues, you might create a button that uses this request to remove unmerged records from a user's queue.

CreateChangeList

CreateChangeList creates a change list in XML format for the current ORS. The change list contains a list of actions and a list of any messages.

CreateChangeList Request Parameters

The following table describes the **CreateChangeList** parameters:

Parameter	Description
SourceRepositoryId	Specifies the database ID of the source repository to use for comparison.
SourceRepositoryXml	Specifies the XML string representing the source repository. Contains NULL if the source repository is a physical database.
TransactionAttributeType	If set to `NOT_SUPPORTED`, the request does not support a transactional context. If set to `REQUIRED`, the request does requires a transactional context. If set to `REQUIRES_NEW`, the request requires a new transactional context. If set to `SUPPORTS`, the request supports but does not require a transactional context.

Response Fields

The **CreateChangeList** response contains the information described in the following table:

Field	Description
ChangeListXml	Contains the XML string representing the change list.

CreateTask

The CreateTask request creates a new task in the C_REPOS_TASK_ASSIGNMENT table and initializes the task data and task properties. Once a task is created, use the UpdateTask request to modify the task.

TaskData

The `TaskData` object contains information about a task.

The following table lists the TaskData fields that you can configure:

Field	Description
TaskRecord	A link to a data record associated with a task.
Comment	An optional task comment.
TaskType	The task type.
SubjectAreaUID	The UID of the task subject area.
Title	The task title.
TaskID	The ROWID of the task. Cannot be set by user.
DueDate	The date when the task is due.

Field	Description
Priority	The priority of the task. `1`: High priority. `0`: Normal priority. The default is `0`. `-1`: Low priority.
StatusEnum	The workflow status. The default is `TaskStatusEnum.OPEN`.
OwnerUID	The user or role ID to whom the task is assigned.
InteractionID	The Interaction ID.
WorkflowProcessID	The ID of the workflow process that contains the task. Cannot be set by user.
CreateDate	The date when the task was created. Cannot be set by user.
Creator	The name of the user who created the task. Cannot be set by user.
LastUpdateDate	The date when the task was updated. Cannot be set by user.
LastUpdatedBy	The name of the user who updated the task. Cannot be set by user.
PreviousOwner	The name of the user or role to whom the task was previously assigned. The value is Null if the task is new or has not been assigned. Cannot be set by user.

TaskRecord

The `TaskRecord` object contains information about a record.

The following table describes the TaskRecord fields:

Field	Description
SiperianObjectUID	An identifier for an object in Informatica MDM Hub.
RecordKey	An identifier for a record in Informatica MDM Hub.
MatchRuleUID	An identifier for a match rule in Informatica MDM Hub. Only merge tasks require a MatchRuleUID.

Required Request Parameters

The following table describes the required parameters for a CreateTask request:

Parameter	Description
TaskData	Specifies the task to create. If an owner is not specified in the TaskData parameter, the task assignment engine will attempt to assign the task at its next scheduled execution time.

Optional Request Parameters

The **CreateTask** API does not have any optional request parameters.

Response Fields

The **CreateTask** response contains the information described in the following table:

Parameter	Description
TaskID	Contains the ROWID_OBJECT of the task that was created.
interactionID	Contains the interactionID that is used to protect any pending records associated with the task. Only SIF requests having this same interactionID can update the task. The interactionID can be set either at the request level or in the TaskData object. If an interactionID is set in both places and the IDs do not match, an SiperianServerException will be thrown.

Use Cases

The following scenario is a common use case for using the **CreateTask** request:

- Create a new task and assign it to a user.

Usage Example

The code in the following example creates a new task:

```
CreateTaskRequest request = new CreateTaskRequest();
TaskData newTask = new TaskData();
request.setTaskData(newTask);
newTask.setTitle("Research and resolve item");
newTask.setComment("This is a new task.");
newTask.setDueDate(new Date());
newTask.setSubjectAreaUid("SUBJECT_AREA.test|Person");
newTask.setTaskType("ReviewNoApprove");
CreateTaskResponse response = (CreateTaskResponse) sipClient.process(request);
```

Related SIF Requests

"UpdateTask" on page 143

Delete

The Delete request sets the record state to DELETED on the specified records from the Hub. If you specify the deleteBORecord flag, then the BO record is deleted even if only a sourceKey and systemName are specified.

State Management

When an XREF record is deleted, the state of the BO record will be calculated as the greatest of the states of its XREFs. The order of precedence for state is ACTIVE, PENDING, DELETED. The following list describes the behavior of this request based on various XREF states:

- Active records will be transitioned to the DELETED state.
- Pending records will be hard deleted.
- Deleted records will remain unchanged.

Required Parameters

The following table lists and describes the parameters that are required by the Delete API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be deleted.
SystemName	Name of the system for which the record must be deleted.
RecordKey	Key to uniquely identify the record to be deleted.
SourceKey	Source key of the record that must be deleted.

Optional Parameters

The following table lists and describes the optional parameters that are used by the Delete API:

Parameter	Description
deleteBORecord	If `true`, the record is deleted at the base object level. The Delete API deletes the base object record and all its cross-reference records. You must have delete privileges for the base object or the parent base object. If `false`, the Delete API deletes the record specified by the RecordKey and SourceKey parameters.
deleteAsSMOS	If `true`, the hub state of the record is set to deleted by the state management override system, and this takes precedence over active records from other source systems. Default is false.

Use Case

Record A has two XREFs that are ACTIVE. If one of the XREFs is deleted, then record A will have one ACTIVE xref and one DELETED XREF. Since the ACTIVE state has higher precedence than the DELETED state, the state of BO record A after the delete operation is ACTIVE. If the remaining ACTIVE XREF is then deleted, record A will have two deleted XREFs and the state of BO record A will be DELETED.

Usage Example

The following example deletes the XREF record with sourceKey=1234 and system=CRM from the package CUSTOMER_UPDATE. If the XREF record is PENDING, it will be hard deleted. If the XREF record is ACTIVE, it will be soft deleted. If the record is already in the DELETED state, the record will remain as is.

Note: Delete throws an exception if you attempt to delete a record that is in the DELETED state.

```
DeleteRequest request = new DeleteRequest();
RecordKey recordKey = new RecordKey();
recordKey.setSourceKey("1234");
recordKey.setSystemName("CRM");
ArrayList recordKeys = new ArrayList();
recordKeys.add(recordKey);
request.setRecordKeys(recordKeys); // Required
request.setSiperianObjectUID("PACKAGE.CUSTOMER_UPDATE"); //Required
DeleteResponse response = (DeleteResponse) sipClient.process(request);
```

Related SIF Requests

"PromotePendingXrefs" on page 116, "Restore" on page 127

DeleteRelationship

DeleteRelationship deletes a relationship between two entities. This request does not remove the record from the relationship table. If the relationship is a foreign key relationship rather than a record in a relationship table, the request sets the foreign key value to null.

This request behaves differently when used with Foreign Key Relationship Types. Since all Relationship records of a Foreign Key Relationship Type use the same End Date, instead of setting the End Date this request sets the foreign key value in the FK Relationship Base Object to null.

The request provides the Hierarchy Manager configuration, the record key, and the relationship type of the relationship to be removed.

Required Parameters

The following table lists and describes the parameters that are required by the DeleteRelationship API:

Parameter	Description
HmConfigurationUid	Unique ID of the Hierarchy Manager configuration.
RelTypeUid	Unique ID of the relationship type.
RecordKey	Key to uniquely identify the relationship record to be deleted.

Optional Parameters

The following table lists and describes the optional parameter that is used by the DeleteRelationship API:

Parameter	Description
deleteAsSMOS	If `true`, the hub state of the relationship record is set to deleted by the state management override system, and this takes precedence over active records from other source systems. Default is false.

Use Case

This is the common scenario for using the DeleteRelationship request:

- **Delete a relationship between two HM entities** — If you have Hierarchy Manager and have populated it with data, you can use the DeleteRelationship request to delete an existing relationship between two entities.

Related SIF Requests

"UpdateRelationship" on page 142, "AddRelationship" on page 57

DescribeSiperianObject

The **DescribeSiperianObject** API returns the metadata for the Informatica MDM Hub objects specified in the request.

Required Parameter

The following table describes the parameter required for the **DescribeSiperianObject** request.

Parameter	Description
objectUid	Each object defined in Informatica MDM Hub has a unique identifier of the form `<objectType>.<objectName>`, for example, `PACKAGE.CUSTOMER_READ`. ObjectUid specifies which object's metadata is returned in the **DescribeSiperianObjects** response. Use the objectUid MATCH_KEY.*<base_object_name>* to request the match key of a base object. For a list of valid object types, see "SiperianObjectUID" on page 150.

Response Parameters

The following table describes the information contained in the **DescribeSiperianObject** response.

Field	Type	Description
objects	List	A list of returned metadata for the objects specified in the request.

If you request a match key for a base object, **DescribeSiperianObject** returns the match column that represents the base object, in addition to the following fields:

Field	Type	Description
fuzzyColumn	Boolean	The value is `true` if the match column is a fuzzy column.
columns	List	The list of UIDs for the columns that make up this match column. UIDs are formed as COLUMN.*<base_object_name>*|*<column_name>*.

Use Case

The following scenario is a common use for the **DescribeSiperianObject** request:

- Obtaining metadata about an object before manipulating it. You can use the **DescribeSiperianObject** request to gather information about an object before you perform any operations on it.

Usage Example

The following example shows how metadata is retrieved for the base objects C_PARTY and C_ADDRESS:

```
DescribeSiperianObjectRequest request =  new DescribeSiperianObjectRequest();
request.setOrsId("orcl-VER_IDD");
ArrayList objectUids = new ArrayList();
objectUids.add(SiperianObjectType.BASE_OBJECT.makeUid("C_PARTY"));
objectUids.add(SiperianObjectType.BASE_OBJECT.makeUid("C_ADDRESS"));
request.setUids(objectUids);
DescribeSiperianObjectResponse response =
(DescribeSiperianObjectResponse)sipClient.process(request);
```

ExecuteBatchAutoMatchAndMerge

ExecuteBatchAutoMatchAndMerge calls the Auto Match and Merge batch job.

Request Parameters

The ExecuteBatchAutoMatchAndMerge request contains the following parameters:

TableName

Specifies the base object table name. Required.

MatchSetName

Specifies the name of the match rule set for the batch job. Default is null. Optional.

Response Fields

The ExecuteBatchAutoMatchAndMerge API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Auto Match and Merge batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchAutoMatchAndMergeRequest req = new ExecuteBatchAutoMatchAndMergeRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 req.setMatchSetName(jobContext.getMatchSetName());
 ExecuteBatchAutoMatchAndMergeResponse executed =
(ExecuteBatchAutoMatchAndMergeResponse) sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchAutomerge

ExecuteBatchAutomerge calls the Automerge batch job.

Request Parameters

The ExecuteBatchAutomerge request contains the following parameters:

TableName

Specifies the base object table name. Required.

Response Fields

The ExecuteBatchAutomerge API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Automerge batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchAutomergeRequest req = new ExecuteBatchAutomergeRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchAutomergeResponse executed = (ExecuteBatchAutomergeResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchBVTSnapshot

ExecuteBatchBVTSnapshot calls the BVT Snapshot batch job.

Request Parameters

The ExecuteBatchBVTSnapshot request contains the following parameters:

TableName

Specifies the base object table name. Required.

Response Fields

The ExecuteBatchBVTSnapshot API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the BVT Snapshot batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchBVTSnapshotRequest req = new ExecuteBatchBVTSnapshotRequest();
 req.setTableName(jobContext.getTableName());        // BO table name
 ExecuteBatchBVTSnapshotResponse executed = (ExecuteBatchBVTSnapshotResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchDelete

The ExecuteBatchDelete API calls the Delete batch job.

Request Parameters

The ExecuteBatchDelete request contains the following parameters:

TableName

Specifies the base object table name. Required.

SourceTableName

Specifies the name of the table that contains the list of cross-reference records to delete. Required.

Cascading

Determines if the batch delete is cascading. Set to `true` to run a cascading batch delete. Optional.

RecalculateBVT

Determines if the BVT is recalculated. Set to `true` to recalculate BVT after a batch delete. Optional.

OverrideHistory

Determines if the MDM Hub records the activity performed by the batch delete in the history tables. Set to `true` to record the history of deleted records in the history table. Set to `false` to ignore the value of PurgeHistory and write the last state of the data into the history tables when the record is deleted. Optional.

PurgeHistory

Determines if the MDM Hub deletes all non-merge history records related to deleted cross-reference record. You cannot retrieve the deleted history records. Set to `true` to delete the history records. Set to `false` to retain the history records. Optional.

Note: If you set OverrideHistory to `true` and PurgeHistory to `true`, the batch delete removes all traces of the deleted records from the history tables.

Response Fields

The ExecuteBatchDelete API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Delete batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchDeleteRequest req = new ExecuteBatchDeleteRequest();
 req.setTableName(jobContext.getTableName());                    // Base object table
name
 req.setSourceTableName(jobContext.getSourceTableName());        // The table that
contains a  list of cross-reference records to delete
 ExecuteBatchDeleteResponse executed = (ExecuteBatchDeleteResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchExternalMatch

ExecuteBatchExternalMatch calls the External Match batch job.

Request Parameters

The ExecuteBatchExternalMatch request contains the following parameters:

TableName

Specifies the base object table name. Required.

MatchSetName

Specifies the name of the match rule set for the batch job. Default is null. Optional.

Response Fields

The ExecuteBatchExternalMatch API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the External Match batch job on a base object table:

```
 SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchExternalMatchRequest req = new ExecuteBatchExternalMatchRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 req.setMatchSetName(jobContext.getMatchSetName());
 ExecuteBatchExternalMatchResponse executed = (ExecuteBatchExternalMatchResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchGenerateMatchTokens

ExecuteBatchGenerateMatchTokens calls the Generate Match Tokens batch job.

Request Parameters

The ExecuteBatchGenerateMatchTokens request contains the following parameters:

TableName

Specifies the base object table name. Required.

FullRestripInd

If the value is 1, the batch job tokenizes all records in the base object.

If the value is 0, the batch job tokenizes the records that have their ROWID_OBJECT values stored in the dirty table.

Default is 0. Optional.

Response Fields

The ExecuteBatchGenerateMatchTokens API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Generate Match Tokens batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchGenerateMatchTokensRequest req = new
ExecuteBatchGenerateMatchTokensRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchGenerateMatchTokensResponse executed =
(ExecuteBatchGenerateMatchTokensResponse) sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchGroup

ExecuteBatchGroup executes a batch group. A batch group is a set of batch jobs executed together, some sequentially and some in parallel according to the configuration. When one job has an error, the group will stop; that is, no more jobs will be started, but running jobs will run to completion. There are two other related services in this request:

- "ResetBatchGroup" on page 126
- "GetBatchGroupStatus" on page 93

Use Case

This is the common scenario for using the executeBatchGroup request:

- **ExecuteBatchGroup with getBatchGroupStatus** — After calling ExecuteBatchGroup, wait and then use "GetBatchGroupStatus" on page 93 to see if the batch group executed successfully.

Related SIF Requests

"GetBatchGroupStatus" on page 93, "ResetBatchGroup" on page 126

ExecuteBatchKeyMatch

ExecuteBatchKeyMatch calls the Key Match batch job.

Request Parameters

The ExecuteBatchKeyMatch request contains the following parameter:

TableName

Specifies the base object table name. Required.

Response Fields

The ExecuteBatchKeyMatch API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Key Match batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchKeyMatchRequest req = new ExecuteBatchKeyMatchRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchKeyMatchResponse executed = (ExecuteBatchKeyMatchResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchLoad

ExecuteBatchLoad calls the Load batch job.

Request Parameters

The ExecuteBatchLoad request contains the following parameters:

TableName

Specifies the staging table name. Required.

ForceUpdateInd

If the value is 0, MDM Hub only loads data that has a more recent late updated date than the data in the hub.

If the value is 1, MDM Hub loads the data regardless of the last updated date.

Default is 0. Optional.

Response Fields

The ExecuteBatchLoad API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Load batch job on a staging table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchLoadRequest req = new ExecuteBatchLoadRequest();
 req.setTableName(jobContext.getTableName());    // STG table name
 ExecuteBatchLoadResponse executed = (ExecuteBatchLoadResponse) sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchMatch

An ExecuteBatchMatch request calls a match batch job, which runs the match process.

Request Parameters

The ExecuteBatchMatch request contains the following parameters:

TableName

Name of the base object table.

MatchSetName

Optional. Name of the match rule set for the batch job. Default is null.

validateTableName

Optional. Name of the table that contains the row ID values to validate. Default is null.

Response Fields

The ExecuteBatchMatch request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

A return code for the interaction. The RetCode field uses the following return codes:

- 0. Indicates a successful transaction.
- -21014. Indicates that the base object is empty or no records match the request.

Usage Example

The following example runs a match batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchMatchRequest req = new ExecuteBatchMatchRequest();
 req.setTableName(jobContext.getTableName());
 req.setMatchSetName(jobContext.getMatchSetName());
 while(rc==0) {
     ExecuteBatchMatchResponse executed = (ExecuteBatchMatchResponse)
sipClient.process( req );
     errMessage = executed.getMessage();
     rc = executed.getRetCode();
     completeStep((rc == -21014) ? 0 : rc, errMessage, context, jobContext);
 }
 if(rc == -21014) {
     // SIP-21014: Error registering start of Match failed during post to Cleanse
Server: Base Object C_CUSTOMER is empty or no more records to match
     // Regard it as normal completion so that test could continue the left jobs
     rc = 0;
 }
```

ExecuteBatchMatchAnalyze

An ExecuteBatchMatchAnalyze request conducts a search to gather match statistics, but does not perform the match process. If areas of data with the potential for huge match requirements are discovered, the MDM

Hub moves the records to a hold status, which allows a data steward to review the data manually before proceeding with the match process.

Request Parameters

The ExecuteBatchMatchAnalyze request contains the following parameters:

TableName

Name of the base object table.

validateTableName

Optional. Name of the table that contains the row ID values to validate. Default is null.

Response Fields

The ExecuteBatchMatchAnalyze request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

Return code for the interaction. The RetCode field uses the following return codes:

- 0. Indicates a successful transaction.
- -21014. Indicates that the base object is empty or no records match the request.

Usage Example

The following example runs a match analysis batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(
    new File(context.getTestPTTStartDir() + "siperian-client.properties")
    );
ExecuteBatchMatchAnalyzeRequest l_req = new ExecuteBatchMatchAnalyzeRequest();
l_req.setTableName("<BO_TABLE_NAME>");
l_req.setValidateTableName("<TABLE_NAME_CONTAINING_ROWID_TO_VALIDATE>");
ExecuteBatchMatchAnalyzeResponse l_res = (ExecuteBatchMatchAnalyzeResponse)
    sipClient.process( l_req );
System.out.println(l_res.getRetCode());
```

ExecuteBatchPromote

A ExecuteBatchPromote request calls a promote batch job.

Request Parameters

The ExecuteBatchPromote request contains the following parameters:

TableName

Name of the base object table.

AllowCommitInd

Optional. Indicates whether the changes can be committed. Default is true.

XrefListToBePromoted

Optional. Name of the table that has the list of XREF records to promote. Default is null.

Response Fields

The ExecuteBatchPromote request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

A return code for the interaction.

Usage Example

The following example runs a promote batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchPromoteRequest req = new ExecuteBatchPromoteRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchPromoteResponse executed = (ExecuteBatchPromoteResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchRecalculateBo

A ExecuteBatchRecalculateBo request calls a recalculate base object batch job.

Request Parameters

The ExecuteBatchRecalculateBo request contains the following parameters:

TableName

Name of the base object table.

RowidObjectTable

Optional. Name of the table that contains the foreign keys to the base object in the ROWID_OBJECT column.

Response Fields

The ExecuteBatchRecalculateBo request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

A return code for the interaction.

Usage Example

The following example runs a recalculate base object batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchRecalculateBoRequest req = new ExecuteBatchRecalculateBoRequest();
 req.setTableName(jobContext.getTableName());        // BO table name
```

```
 ExecuteBatchRecalculateBoResponse executed = (ExecuteBatchRecalculateBoResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchRecalculateBvt

ExecuteBatchRecalculateBvt calls the Recalculate BVT batch job.

Request Parameters

The ExecuteBatchRecalculateBvt request contains the following parameters:

TableName

Specifies the base object table name. Required.

RowidObject

Specifies the rowidObject. Required.

Response Fields

The ExecuteBatchRecalculateBvt API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Recalculate BVT batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchRecalculateBvtRequest req = new ExecuteBatchRecalculateBvtRequest();
 req.setTableName(jobContext.getTableName());        // BO table name
 req.setRowidObject(jobContext.getRowidObject());        // rowidObject
 ExecuteBatchRecalculateBvtResponse executed = (ExecuteBatchRecalculateBvtResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchResetMatchTable

ExecuteBatchResetMatchTable calls the Reset Match Table batch job.

Request Parameters

The ExecuteBatchResetMatchTable request contains the following parameters:

TableName

Specifies the base object table name. Required.

Response Fields

The ExecuteBatchResetMatchTable API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Reset Match Table batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchResetMatchTableRequest req = new ExecuteBatchResetMatchTableRequest();
 req.setTableName(jobContext.getTableName());        // BO table name
 ExecuteBatchResetMatchTableResponse executed = (ExecuteBatchResetMatchTableResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchRevalidate

ExecuteBatchRevalidate calls the Revalidate BO batch job.

Request Parameters

The ExecuteBatchRevalidate request contains the following parameters:

TableName

Specifies the base object table name. Required.

OnlyCmDirtyInd

Apply only for CM_DIRTY_IND = 1. Default is false. Optional.

RecalcBvtInd

Recalculate BVT. Default is false. Optional.

Response Fields

The ExecuteBatchRevalidate API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Revalidate BO batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchRevalidateRequest req = new ExecuteBatchRevalidateRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchRevalidateResponse executed = (ExecuteBatchRevalidateResponse)
sipClient.process( req );
```

```
String errMessage = executed.getMessage();
int rc = executed.getRetCode();
```

ExecuteBatchStage

ExecuteBatchStage calls the Staging batch job.

Request Parameters

The ExecuteBatchStage request contains the following parameters:

TableName

Specifies the staging table name. Required.

Response Fields

The ExecuteBatchStage API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Staging batch job on a staging table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchStageRequest req = new ExecuteBatchStageRequest();
 req.setTableName(jobContext.getTableName());        // STG table name
 ExecuteBatchStageResponse executed = (ExecuteBatchStageResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchSynchronize

ExecuteBatchSynchronize calls the Synchronize batch job.

Request Parameters

The ExecuteBatchSynchronize request contains the following parameters:

TableName

Specifies the base object table name. Required.

OnlyCmDirtyInd

Apply only for CM_DIRTY_IND = 1. Default is false. Optional.

Response Fields

The ExecuteBatchSynchronize API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Synchronize batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchSynchronizeRequest req = new ExecuteBatchSynchronizeRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchSynchronizeResponse executed = (ExecuteBatchSynchronizeResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

ExecuteBatchUnmerge

The ExecuteBatchUnmerge API calls the Unmerge batch job. You can unmerge records that were merged by a previous process.

Request Parameters

The ExecuteBatchUnmerge request contains the following parameters:

TableName

Specifies the base object that contains the records to unmerge. Required.

SourceTableName

Specifies the cross-reference records to unmerge. Required.

Response Fields

The ExecuteBatchUnmerge API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

UnmergedXrefsCount

Number of cross-reference records that were unmerged.

RetCode

Contains the return code.

Usage Example

The following example runs the Unmerge batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchUnmergeRequest req = new ExecuteBatchUnmergeRequest();
 req.setTableName(jobContext.getTableName());                    // BO table name
 req.setSourceTableName(jobContext.getSourceTableName());        // the list of XREFs to
unmerge table name
 ExecuteBatchUnmergeResponse executed = (ExecuteBatchUnmergeResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
```

```
int rc = executed.getRetCode();
```

ExecuteBatchValidateFKRelationships

ExecuteBatchValidateFKRelationships calls the Validate Foreign Key Relationships batch job.

Request Parameters

The ExecuteBatchValidateFKRelationships request contains the following parameters:

TableName

Specifies the base object table name. Required.

ListRowidColumn

Contains the Rowid list of columns participating in the FK to be validated. Optional.

CascadeValidateInd

Validates child foreign key relationships. Default is false. Optional.

Response Fields

The ExecuteBatchValidateFKRelationships API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the Validate Foreign Key Relationships batch job on a base object table:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 ExecuteBatchValidateFKRelationshipsRequest req = new
ExecuteBatchValidateFKRelationshipsRequest();
 req.setTableName(jobContext.getTableName());    // BO table name
 ExecuteBatchValidateFKRelationshipsResponse executed =
(ExecuteBatchValidateFKRelationshipsResponse) sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

FlagForAutomerge

The **FlagForAutomerge** API flags a record for automerge in the match table, C_<*base_object_name*>_MTCH. If a record in the match table has a `AUTOMERGE_IND` value of `1`, the record is merged during the next automerge process. If the **FlagForAutomerge** request is for a record that does not exist in the match table, the record is created in the match table and the `AUTOMERGE_IND` is set to `1`.

Required Parameters

The following table describes the required parameters:

Parameter	Description
MatchRuleUid	Specifies the match rule that the merged records are attributed to. The match rule UID needs to be specified in the following format: `MATCH_RULE.<TABLE_NAME>|<MATCH_RULESET_NAME>|<MATCH_RULE_NUMBER>`
UnmergedRecordKey	Specifies the record key of the unmerged record.
MatchedRecordKey	Specifies the record key of the matched record.

Optional Parameters

The **FlagForAutomerge** request does not have optional parameters.

Response Fields

The following table describes the response fields:

Parameter	Description
Message	Contains a message indicating if the **FlagForAutomerge** request was processed successfully.
InteractionID	Contains the interaction ID.

Usage Example

The following is a typical scenario for using the FlagForAutoMerge request:

- Queue a merge candidate for merge during the next automerge process.

FlagForAutomerge Usage Example

The following example shows how to flag record 111 to automerge with the record it matches with, in this case record 222:

```
FlagForAutomergeRequest request = new FlagForAutomergeRequest();
 request.setMatchRuleUid(SiperianObjectType.MATCH_RULE.makeUid("MyMatchRule"));
 request.setUnmergedRecordKey(RecordKey.sourceKey("111", "Acme"));
 request.setMatchedRecordKey(RecordKey.sourceKey("222", "Acme"));
 FlagForAutomergeResponse response = sipClient.process(request);
```

GenerateConstraints

The GenerateConstraints API generates missing constraints for a table. Use the GenerateConstraints API to create indexes after you use the RegisterCustomIndex to register an index that does not physically exist.

Request Parameters

The GenerateConstraints request contains the following parameters:

TableName

Specifies the base object table name. Required.

Scope

Specifies the scope of the constraints of the request. Required. Scope can be one of the following values:

- FK. Foreign keys.
- NI. Non-unique indexes.
- PK. Primary keys.
- UI. Unique indexes.
- UK. Unique keys.
- NP. Keys that are not primary keys.
- NK. Keys that are not foreign keys.
- ALL. All constraints.

AbortOnFail

If true, specifies that the job stops when it fails. Required.

SetFkDisabled

If true, sets the foreign keys to disabled. Required.

Analyze

Required.

Response Fields

The GenerateConstraints API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

RetCode

Contains the return code.

Usage Example

The following example runs the GenerateConstraints batch job on the C_CUSTOMER base object:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 GenerateConstraintsRequest req = new GenerateConstraintsRequest();
 req.setTableName(jobContext.getTableName());        // table name. e.g. C_CUSTOMER
 req.setScope("ALL");
 req.setAbortOnFail(true);
 req.setSetFkDisabled(false);
 req.setAnalyze(false);
 GenerateConstraintsResponse executed = (GenerateConstraintsResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

Get

Get uses a record key to retrieve a single row of data from the specified package. The row can include data from base objects and from child records (that is, content metadata such as History, Xref, Xref History, and

Raw) associated with the base object. You can use this request against the regular MDM packages (“PACKAGE.” SiperianObjectUid prefix).

You can also get lineage and trust information. The trust scores are returned for the package record and the cross reference records. The lineage information returned as indicator on the trust enabled fields indicating whether the specific field of the cross-reference record has won over other cross-references and is used on the base object.

When performing a **Get** request using a ROWID_OBJECT for a base object record that has been merged into another base object record, **Get** returns the surviving base object record. For example, if two base object records are merged, one with a ROWID_OBJECT value of ROWID_A and the other with a value of ROWID_B, the ROWID_OBJECT of the surviving base object could be ROWID_A. In this scenario, if you perform a **Get** request for ROWID_B, the **Get** response returns ROWID_A.

For MDM packages, you can use this request to retrieve the following types of the content metadata for underlying primary base object of the package and the trust score and the lineage information for the trust enabled columns:

SiperianObjectType	Description
XREF	Cross-reference data. If state management is enabled for the parent of the package, then this option will return only the cross reference records that are in the ACTIVE state.
PENDING_XREF	Cross-reference data that is in the PENDING state. This option is only valid when state management is enabled for the parent of the package. Otherwise, an exception is thrown.
DELETED_XREF	Cross-reference data that is in the DELETED state. This option is only valid when state management is enabled for the parent of the package. Otherwise, an exception is thrown.
XREF_HISTORY	Previous values for each of the underlying cross references of the specified base object. **Note:** Base object history has to be enabled
HISTORY	Previous values for the specified base object record. **Note:** Base object history has to be enabled.
RAW	Raw records associated with the specific base object record. **Note:** Raw retention needs to be enabled on at least one staging table belonging to the specified base object.

If the package is based on a query that joins multiple base objects, content metadata is returned only for the primary base object.

Required Parameters

The following table lists and describes the parameters that are required by the Get API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be queried.
RecordKey	Key to uniquely identify the record to be fetched.
SystemName	Name of the system for which XREF and XREF history must be retrieved.
RecordTypes	Types of records to retrieve.

Optional Parameters

The following table lists and describes the optional parameters that are used by the Get API:

Parameter	Description
EffectiveDate	The date for which you must retrieve values of the base object. **Note:** EffectiveDate must be used only for timeline-enabled base objects.
HistoryDate	The date for which you retrieve base object data that is effective at the specified point in time. The Get API does not return any record with a HIST_CREATE_DATE value that is equal to, or preceded by, the HistoryDate value. The Get API truncates milliseconds for HistoryDate. If you run the request on Oracle, the value of HistoryDate is adjusted for Daylight Saving Time (DST) when DST is enabled for the operating system. If the HistoryDate value is later than a record's HIST_CREATE_DATE value by less than one hour during DST, the API call does not return the record. **Note:** Use HistoryDate for timeline-enabled base objects only.
DataFilter	The SQL condition to be applied to the result set.

Use Case

The following is a common scenario for using the Get request:

- **Get** used to retrieve a row and its associated child records. The most common use of get is to retrieve a single row of data, with any associated child records.

Usage Example

The following example gets a record with ROWID_OBJECT key 782 from the package PARTY_ADDRESS_READ_PKG.

```
GetRequest request = new GetRequest();
RecordKey recordKey = new RecordKey();
recordKey.setRowid("782");
request.setRecordKey(recordKey); //Required
request.setSiperianObjectUID("PACKAGE.PARTY_ADDRESS_READ_PKG"); //Required
GetResponse response = sipClient.process(request);
```

Related SIF Requests

"GetSearchResults " on page 101, "Put " on page 118, "SearchQuery " on page 133

GetAggregatePeriod

The **GetAggregatePeriod** API retrieves the aggregate period for multiple base objects arranged into a tree with a specified root base object.

The aggregate period is an aggregate of the effective periods of all base object records in a request that encompasses the specified effective date. The advantage of an aggregated effective period is there is a minimal intersection of individual effective periods around the specified effective date.

Required Parameters

The following table lists and describes the parameters that are required by the GetAggregatePeriod API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be queried.
RecordKey	Key to uniquely identify the record to be fetched.
EffectiveDate	The effective date of the calculated effective period.
JoinUids	List of JoinObjectType instances. Each JoinObjectType describes a relationship between two elements of the base object tree. Each JoinObjectType contains a match path UID for the base object join. Optionally, each JoinObjectType can contain a list of filter fields.

Optional Parameters

The following table lists and describes the optional parameters that are used by the GetAggregatePeriod API:

Parameter	Description
RecordStates	The Hub state indicator to filter the records. The aggregate period depends on the effective periods of the filtered records. By default, only records in an ACTIVE state are taken into account.

Response Fields

The following table describes the fields returned by the **GetAggregatePeriod** response:

Field	Description
AggregatedPeriodStartDate	Start date of the calculated effective period.
AggregatedPeriodEndDate	End date of the calculated effective period.

Use Case

The following is a common scenario for using the GetAggregatePeriod API:

- The base object C_PHONE base object is a one-to-many child of C_ORGANIZATION. They are linked with the match path C_MT_ORG_PHONE. **GetAggregatePeriod** used to retrieve the C_PHONE records that have the value "FAX" value in the PHONE_TYPE field. In this scenario, you want to get the aggregate period for all three records on effective date September 1, 2014.

The following table describes the relevant fields for this scenario:

Base Object Name	Period Start Date	Period End Date
Organization	January 1, 2014	
Phone 1	January 1, 2014	December 31, 2014
Phone 2	May 1, 2014	

Usage Example

The following example returns a response that contains aggregatedPeriodStartDate of May 1, 2014 and aggregatedPeriodEndDate of December 31, 2014 from C_ORGANIZATION described in the use case.

```
GetAggregatePeriodRequest request = new GetAggregatePeriodRequest();
RecordKey recordKey = RecordKey.rowid("782")
request.setRecordKey(recordKey); //Required
request.setSiperianObjectUID("BASE_OBJECT.C_ORGANIZATION"); //Required
Date date=new Date(2014, 09, 01);
JoinObjectType sifJoin = new JoinObjectType();
sifJoin.setSiperianObjectUid("MATCH_PATH_COMPONENT.C_MT_ORG_PHONE");
Field filterField = new Field("C_MT_ORG_PHONE.PHONE_TYPE", "FAX");
sifJoin.setFilterFields(Collections.singletonList(filterField));
request.getJoins().add(sifJoin); //Required
GetAggregatePeriod response = sipClient.process(request);
```

GetAssignableUsersForTasks

The **GetAssignableUsersForTasks** API retrieves a list of users who can be assigned a list of specified tasks. The algorithm relies on the task assignment configuration by default but you can customize the configuration via the CMXUE.get_assignable_users_for_task user exit.

Required Request Parameters

The following table describes the required parameters for the **GetAssignableUsersForTasks** request:

Parameter	Description
AssignableTaskInfoList	The task type and subject area UID of a list of tasks.

Optional Request Parameters

The **GetAssignableUsersForTasks** request does not have optional parameters.

Response Fields

The following table describes the fields returned by the **GetAssignableUsersForTasks** response:

Field	Description
UserUIDs	Contains the UIDs of the users who are permitted to receive the tasks listed in the **GetAssignableUsersForTasks** request.

Usage Example

The code in the following example retrieves users who can have Merge tasks in the Person subject area assigned to them:

```
GetAssignableUsersForTasksRequest request = new GetAssignableUsersForTasksRequest();
List taskInfoList = new ArrayList();
taskInfoList.add(new AssignableTaskInfo("Merge","SUBJECT_AREA.test|Person"));
request.setAssignableTaskInfoList(taskInfoList);
GetAssignableUsersForTasksResponse response =
     (GetAssignableUsersForTasksResponse) sipClient.process(request);
```

GetAssignedRecords

GetAssignedRecords fetches the current user's records that were assigned by an "AssignUnmergedRecords" on page 58 request. Can request records in either the Unmerged or the Onhold state.

The request contains a package, a record state (UNMERGED or ON_HOLD), and a maximum number of records to return. The response contains a set of records and a token to use to fetch more results. Use "GetSearchResults " on page 101 to get subsequent sets of records.

Use Case

This is the common scenario for using the GetAssignedRecords request:

- **GetAssignedRecords used to retrieve assigned records for display in the user interface of a custom-designed application**—The most common use of GetAssignedRecords is to retrieve the records that are assigned to a specific user for display in a custom-designed UI.

Usage Example

The following example requests the UNMERGED records for the CUSTOMER_UPDATE package that are assigned to the user making this request.

```
GetAssignedRecordsRequest request = new GetAssignedRecordsRequest();
request.setSiperianObjectUID("PACKAGE.CUSTOMER_UPDATE");
request.setRecordsToReturn(10);
request.setRecordState(RecordState.UNMERGED);
request.setReturnTotal(false);

GetAssignedRecordsResponse response = (GetAssignedRecordsResponse)
sipClient.process(request);
```

Related SIF Requests

"AssignUnmergedRecords" on page 58, "ClearAssignedUnmergedRecords" on page 66

GetBatchGroupStatus

GetBatchGroupStatus returns the status of a batch group; polls for status after executing asynchronously. To learn more about batch groups, see the *Multidomain MDM Configuration Guide*.

Note: When making an asynchronous call, the runStatus of 0 (success) means that GetBatchGroupStatus was successfully placed in the async queue. To see the actual runStatus of the batch group, you can also specify a value in the jmsReplyTo field when making the call. The SIF response message containing the run status of the batch group will be returned on this queue. Alternatively, you can also use the Audit Manager in the Hub Console to enable the audit for "No System: GetBatchGroupStatus" and enable the audit XML. Then, use the GetBatchGroupStatus call again and then check C_REPOS_AUDIT:DATA_XML for the SIF response. The response will show the batch group's "failed" status. For more information regarding the Audit Manager, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the GetBatchGroupStatus request:

- **GetBatchGroupStatus with ExecuteBatchGroup** — After calling "ExecuteBatchGroup" on page 77, wait and then use GetBatchGroupStatus to see if the batch group executed successfully.

Related SIF Requests

"ExecuteBatchGroup" on page 77, "ResetBatchGroup" on page 126

GetBvt

GetBvt retrieves the best version of truth (BVT) from the specified package using a known key. The specified package must have a base object (BO) as its parent and the base object must be a link style BO instead of a merge style BO. This option can be configured in the schema manager of the Hub Console. The BVT is calculated on the set of records belonging to the same link group as the input record key.

Note: You can configure GetBvt requests in all systems when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, refer to the *Multidomain MDM Configuration Guide*.

State Management

You can include pending records in the BVT calculation if state management is enabled on the parent base object by adding setIncludePending(TRUE) to the request. For more information regarding how to enable state management, refer to the *Multidomain MDM Data Steward Guide* or the *Multidomain MDM Configuration Guide*.

Required Parameters

The following table lists and describes the parameters that are required by the GetBVT API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be queried.
RecordKey	Key to uniquely identify the record for which BVT must be retrieved.

Optional Parameters

The following table lists and describes the optional parameters that are used by the GetBVT API:

Parameter	Description
EffectiveDate	The date for which you must retrieve values of the base object. **Note:** EffectiveDate must be used only for timeline-enabled base objects.
HistoryDate	The date for which you retrieve base object data that is effective at the specified point in time. If HistoryDate is equal to or earlier than HIST_CREATE_DATE, no records are returned. For Oracle environments, milliseconds for the HistoryDate are truncated. If HistoryDate is later than HIST_CREATE_DATE by less than a second, no records are returned. If you have Daylight Saving Time (DST) enabled for the operating system and HistoryDate is later than HIST_CREATE_DATE by less than one hour during DST, no records are returned. **Note:** HistoryDate must be used only for timeline-enabled base objects.
IncludePending	If set to `true`, it includes pending records in BVT calculation. The default is `false`.

Use Case

The following is a common scenario for using the GetBVT request:

GetBVT used to retrieve the most relevant customer name — A typical use of GetBVT is to retrieve the best version of truth of a customer's first and last name.

Usage Example

The following example gets the BVT for a record with ROWID_OBJECT key 21 from the package, ADDRESS_UPDATE_PKG:

```
GetBvtRequest getBvtRequest = new GetBvtRequest();
getBvtRequest.setSiperianObjectUid(SiperianObjectType.PACKAGE.makeUid("ADDRESS_UPDATE") )
;
RecordKey recordKey = new RecordKey();
recordKey.setRowid("21");
getBvtRequest.setRecordKey(recordKey);
getBvtRequest.setIncludePending(false);
GetBvtResponse getBvtResponse = sipClient.process (getBvtRequest);
```

Related SIF Requests

"Get" on page 88, "Link" on page 112, "Unlink" on page 139

GetEffectivePeriods

The GetEffectivePeriods API retrieves the aggregate effective period for the specified base object record.

The GetEffectivePeriods request contains the key to the base object record for which the aggregate effective period must be retrieved. The response contains a list that includes all effective periods for the requested base object record.

Required Parameter

The RecordKey parameter is required to uniquely identify the record for which the aggregate effective period must be retrieved.

Optional Parameter

The HistoryDate parameter specifies the date for which you retrieve base object data that is effective at the specified point in time. If HistoryDate is equal to or earlier than HIST_CREATE_DATE, no records are returned.

For Oracle environments, milliseconds for HistoryDate are truncated. If HistoryDate is later than HIST_CREATE_DATE by less than a second, no records are returned. If you have Daylight Saving Time (DST) enabled for the operating system and HistoryDate is later than HIST_CREATE_DATE by less than one hour during DST, no records are returned.

Note: HistoryDate must be used only for timeline-enabled base objects.

Use Case

The following is a common scenario for using the GetEffectivePeriods request:

Retrieve the period for which a customer's billing address is valid.

Usage Example

The following example gets the effective period for a record with ROWID_OBJECT key 28 from the base object C_BO:

```
GetEffectivePeriodsRequest req=new GetEffectivePeriodsRequest();
RecordKey rec=new RecordKey();
rec.setRowid("28");
req.setRecordKey(rec);
req.setSiperianObjectUid("BASE_OBJECT.C_BO");
GetEffectivePeriodsResponse response =
(GetEffectivePeriodsResponse)sipClient.process(req);
```

GetEntityGraph

The **GetEntityGraph** request fetches a graph of entities and relationships related to a specified set of entities. The entities and relationships returned can be one or multiple hops away from the entities in the request.

Required Parameters

The following table describes the required parameters for the **GetEntityGraph** request.

Parameter	Description
HmConfigurationUid	UID of the HM Configuration.
EntityKeys	List of SiperianObjectRecordKey objects identifying entities for which multiple levels of related relationships and entities will be retrieved.

Optional Parameters

The following table describes the optional parameters for the **GetEntityGraph** request.

Parameter	Description
RecordStates	Specifies the Hub State Indicator value that the returned elements must have. **Note:** Only use RecordStates if State Management is enabled for all entity and relationship base objects.
EffectiveDate	Specifies that date for which the returned elements must be in effect. **Note:** Only use EffectiveDate for timeline-enabled base objects.
EntityGraphFilter	Specifies the limit on the graph depth (number of hops), breadth (number of relationships at each hop), and the total number of relationships.

Response Fields

The following table describes the fields returned by the **GetEntityGraph** response.

Field	Description
Records	A list of relationship and entity record objects.
EntityInfos	Contains additional information about an entity returned by **GetEntityGraph**. Each entity returned in the records has a corresponding EntityInfo.
TotalGraphReturned	If `true`, the entire graph was returned. If `false`, the entire graph was not returned.
ListNode	If `true`, the maximum breadth limit was reached for the entity. If `false`, the maximum breadth limit was not reached for the entity.

Use Case

This is the common scenario for using the GetEntityGraph request:

- **Fetch the entities and relationships associated with a specific HM entity or entities** — If you have Hierarchy Manager and have populated it with data, you can use GetEntityGraph to get the entities and relationships associated with one or more entities.

GetEntityGraph Request Usage Example

The following example shows how to use the GetEntityGraph request:

```
GetEntityGraphRequest request = new GetEntityGraphRequest();
request.setHmConfigurationUid("HM_CONFIGURATION.Default|Master");

ArrayList keys = new ArrayList();
SiperianObjectRecordKey key = new SiperianObjectRecordKey();
key.setRecordKey(RecordKey.rowid("123"));
key.setSiperianObjectUid("HM_ENTITY_TYPE.Company");
keys.add(key);

key = new SiperianObjectRecordKey();
key.setRecordKey(RecordKey.rowid("456"));
key.setSiperianObjectUid("HM_ENTITY_TYPE.Company");
keys.add(key);

request.setEntityKeys(keys);

EntityGraphFilter filter = new EntityGraphFilter();
filter.setActiveRelsOnly(true); // Only get current employees

// Only get 3 levels of relationships
filter.setMaximumDepth(3);

// Only traverse Entities that have less than 10 Relationships
filter.setMaximumBreadth(10);

// Do not return more than 100 total Relationships
filter.setMaximumRelationships(100);

request.setEntityGraphFilter(filter);

GetEntityGraphResponse response =
   (GetEntityGraphResponse) sipClient.process(request);

// Get List of Record objects for Entities and Relationships.
List recs = response.getRecords();

// Get EntityInfo object for each Entity returned.
List entInfos = response.getEntityInfos();
```

Related SIF Requests

"GetOneHop" on page 99

GetLookupValue

GetLookupValue enables an application program to obtain the display value corresponding to a key value for the specified object columns. This API is used to retrieve the user friendly descriptions for specific code values when a package contains only the code value and the developer needs to display the user friendly description of the code in the user interface. This request is also useful when displaying an individual record.

The request contains a list of LookupFields. Each LookupField contains an identifier for the column and a foreign key value.

The response contains a record that has a field for each LookupField. The order of the fields matches the order of the LookupFields in the request. In each field, the name is the lookup (foreign key) value and the value is the lookup display name.

This request is intended to be used together with the "GetLookupValues" on page 98 and "SearchLookupValues" on page 128 requests. The difference between these APIs is that the GetLookupValue API retrieves descriptions only for the specified code values, while the GetLookupValuesRequest and the SearchLookupValuesRequest return the list of valid lookup code values and lookup code descriptions for the specified lookup column.

Use Case

This is the common scenario for using the GetLookupValue request:

- **Fetch the valid values for a particular field and display them in a UI**—In a custom UI, you can use GetLookupValue to fetch a list of valid values for a field. You can then display these values as a set of selections for the user.

Related SIF Requests

"GetLookupValues" on page 98, "SearchLookupValues" on page 128, "DeleteRelationship" on page 71

GetLookupValues

GetLookupValues enables an application program to populate fields of a user interface with a list of values for a given column. This request is similar to the "GetLookupValue" on page 97 request, but the response contains a list of lists rather than a single list.

This request can be used on any foreign key column. A foreign key to a lookup table has a limited set of values. Other foreign keys can have large numbers of possible values. This request is intended and most useful for lookup tables, when you want to display the list of acceptable values to a user.

The response contains a record for each column that has fields with the lookup information. In each field, the name is the lookup (foreign key) value and the value is the lookup display name.

Use Case

This is the common scenario for using the GetLookupValues request:

- **Fetch the valid values for a set of fields and display them in a UI**—In a custom UI, you can use getLookupValues to fetch a list of valid values for a set of fields. You can then display these values as a set of selections for the user.

Related SIF Requests

"GetLookupValue" on page 97, "SearchLookupValues" on page 128, "DeleteRelationship" on page 71

GetMatchedRecords

GetMatchedRecords returns records that are candidates to match a specified record.

The request contains a package and a record. The response contains a collection of potentially matching records from the specified package.

You can configure GetMatchedRecords requests in all systems when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

On IBM DB2, you cannot use functions such as TRIM, LTRIM, and RTRIM in the sortCriteria field of the GetMatchedRecords API. The select statement of the GetMatchedRecords API uses ORDER BY clause that cannot be combined with functions such as TRIM, LTRIM, and RTRIM.

State Management

If Hub state is specified in the request (see setRecordStates(ArrayList)), the parent Base Object of the specified package must have state management enabled. For more information about how to enable state management, see the *Multidomain MDM Cleanse Adapter Guide* or the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the GetMatchedRecords request:

- **Fetch the match candidates for a specified record, display them in a UI, and use the merge request to merge the match candidate the user selects**—After using GetMatchedRecords to retrieve candidate matches for a record, you can display the results in a UI for a user. The user can then select a candidate. Use merge to merge the two records.

Related SIF Requests

"Merge " on page 114

GetMergeHistory

GetMergeHistory returns a tree representing the merge history for a specified base object record. The root node of the tree is the surviving rowid. The child nodes represent the records that have been merged into the surviving record. Each node contains the rowid and merge date of the record.

The request specifies a package and a key to identify the record. The response contains a tree of (rowid, merge date) pairs.

Note: You can configure GetMergeHistory requests according to a specific system when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the GetMergeHistory request:

- **Fetch the list of merges from which the current record was formed**—get the list of merges, the product of whose cumulative changes have resulted in this record.

Related SIF Requests

"Merge " on page 114, "Unmerge " on page 139

GetOneHop

The GetOneHop Hierarchy Manager request fetches information about the entities directly related to the group of entities that you specify in a Hierarchy Manager configuration.

The request contains the Hierarchy Manager configuration, a list of entity keys, and the filtering criteria. The response contains lists of entity records and relationships, and a search token to use for fetching additional information. Use GetOneHop to understand relationships between records in a hierarchy.

For timeline-enabled entities, the request must include the EffectiveDate parameter value.

When you use GetOneHop to calculate the best version of the truth for many records, increase the value of the following properties in `cmxserver.properties`:

- `searchQuery.buildBvtTemp.MaxRowCount`
- `sif.search.result.query.temptableTimeToLive.seconds`

For example, if you use GetOneHop for more than 10,000 records, set the values as shown in the following example:

```
sif.search.result.query.temptableTimeToLive.seconds=3600
searchQuery.buildBvtTemp.MaxRowCount=100000
```

If you process a higher number of records, increase the values for the `searchQuery.buildBvtTemp.MaxRowCount` and `sif.search.result.query.temptableTimeToLive.seconds` properties.

If you enable timeline, start with a value of `5000` for the `searchQuery.buildBvtTemp.MaxRowCount` property.

You must also increase the transaction timeout value of the application server.

Use Case

This following scenario uses the GetOneHop request:

Fetch one level of entities and relationships associated with a specific HM entity or entities

If the Hierarchy Manager is populated with data, use GetOneHop to get a single level of entities and relationships associated with one or more entities.

GetOrsList

GetOrsList retrieves a list of the Operational Record Stores (ORS) registered in the master database.

Required Parameters

The **GetOrsList** request does not have any required parameters. The ORS ID is not required because this API request operates on the master database.

Response Fields

The **GetOrsList** response can contain the information described in the following table:

Field	Description
MetaDataOrs	Contains the display name, the physical name, and the ID of the ORS. Each ORS in the list is represented by a MetaDataOrs object.

Use Case

The following scenarios are common uses of the **GetOrsList** API:

- Retrieve the list of ORS databases to display them for selection in a custom client application.
- Retrieve the ORS ID to use as an input using MetaDataOrs.getOrsId() in subsequent calls where the ORS ID is required but is not hard-coded.

GetOrsList Request Usage Example

The following example gets the list of all registered ORS databases:

```
GetOrsListRequest request = new GetOrsListRequest();
GetOrsListResponse response = (GetOrsListResponse)sipClient.process(request);
```

GetOrsList Response Usage Example

The following example displays the returned list of all registered ORS databases:

```
for(Iterator iter=response.getOrsList().iterator(); iter.hasNext();) {
//iterate through response records
MetaDataOrs ors = (MetaDataOrs) iter.next();
System.out.println("ORS Display Name"+ ors.getDisplayName());
System.out.println(" Physical name" + ": " + ors.getName());
System.out.println(" ORS Id" + ": " + ors.getOrsId());
};
```

GetOrsMetadata

GetOrsMetadata retrieves the metadata for the current repository. In order to successfully export the repository, your ORS must be in a valid state. The **GetOrsMetadata** request provides the same functionality as the Export tool in the MDM Hub Console. For more information about the Export tool, see the *Multidomain MDM Repository Manager Guide*.

Required Parameters

The **GetOrsMetadata** API does not have any required parameters.

Optional Parameters

The **GetOrsMetadata** API does not have any optional parameters.

Response Fields

The **GetOrsMetadata** response contains the information described in the following table:

Field	Description
ChangeListXml	Contains the XML string representing the exported repository.

Usage Example

The examples shows how to use the **GetOrsMetadata** API to retrieve metadata:

```
GetOrsMetadataRequest request = new GetOrsMetadataRequest ();
GetOrsMetadataResponse response = (GetOrsMetadataResponse) sipClient.process(request);
```

GetSearchResults

GetSearchResults retrieves additional pages of records for any API with paging enabled. The APIs that support paging are:

- GetAssignedRecords
- Get Matched Records
- GetOneHop
- GetTasks
- Search LookupValues
- SearchHmQuery
- SearchMatch
- SearchQuery

Use the SortCriteria parameter when using the preceding APIs to ensure the records are not returned in a random order. When the DisablePaging parameter for the preceding APIs is set to the default of `false`, a search token is returned.

You must use the search token within a limited period of time after you receive it. The default time limit for search token validity is 15 minutes. To learn more about changing this limit, contact Informatica Global Customer Support.

Required Parameters

The following table lists and describes the parameters that are required by the GetSearchResults request:

Parameter	Description
SearchToken	Identifies which search to return additional results for.
RecordsToReturn	The number of records to return.
FirstRecord	The index of the first record to return. This parameter is useful for returning subsequent pages of results. For example, if RecordsToReturn=`20` and FirstRecord=`41`, the third page of results is returned (records 41 to 60). **GetSearchResults** returns an error if the value of FirstRecord is 0, a negative value, or greater than the number of records returned by the original search query.

Optional Parameters

The **GetSearchResults** request does not have optional parameters.

Response Fields

The **GetSearchResults** response contains an array list of the records specified by the required parameters.

Use Case

This is the common scenario for using the GetSearchResults request:

- Fetch the next page of a set of records returned from a request that returns multiple pages. After using any request that returns the first of multiple pages of a set of records, you can use **getSearchResults** repeatedly to get the subsequent pages.

GetSearchResults Request Usage Example

The following example requests the second page of data from a search. Ten records are displayed per page:

```
...
SearchQueryResponse sqResponse = (SearchQueryResponse)
sipClient.process(request);
String searchToken = sqResponse.getSearchToken();

GetSearchResultsRequest request = new GetSearchResultsRequest();
request.setSearchToken(searchToken);
request.setRecordsToReturn(10);
request.setFirstRecord(11);
GetSearchResultsResponse response = (GetSearchResultsResponse)
sipClient.process(request);
```

GetSearchResults Response Usage Example

The code in the following example shows how to print out the records returned by the **GetSearchResults** response:

```
GetSearchResultsResponse response = new GetSearchResultsResponse();
int i=0;
for(Iterator iter=response.getRecords().iterator(); iter.hasNext();)
```

```
{ //iterate through response records
 System.out.println("Printing response record " + i);
 Record record = (Record) iter.next();
 Collection fields = record.getFields();
 for(Iterator fieldIter=fields.iterator(); fieldIter.hasNext();)
{ //iterate through rest of fields
 Field f = (Field) fieldIter.next();
 System.out.println(f.getName() + ": " + f.getValue());
 }
}
```

Related SIF Requests

- "GetAssignedRecords" on page 93
- "GetMatchedRecords" on page 98
- "GetOneHop" on page 99
- "GetTasks " on page 107
- "SearchHmQuery" on page 128
- "SearchLookupValues" on page 128
- "SearchMatch " on page 128
- "SearchQuery " on page 133
- "SearchRequestBase" on page 135

GetSiperianObjectCompatibility

GetSiperianObjectCompatibility obtains a checksum that represents the definition of the specified object in Informatica MDM Hub. This is used with ORS-specific APIs.

This API can be used to determine if an object on the server is compatible with a class in the client library for an ORS specific PACKAGE, MAPPING, or CLEANSE_FUNCTION. ORS specific APIs and objects are generated in the Hub Console's SIF Manager. This request should be used to determine if an objects definition on the server has changed since the last time ORS specific objects were generated. To resolve an incompatibility between a client object and its server counterpart is to regenerate the ORS specific objects. For more information about generating ORS specific objects, see the *Multidomain MDM Cleanse Adapter Guide*.

Use Case

This is the common scenario for using the GetSiperianObjectCompatibility request:

- **Fetch the checksum for an object to use when using ORS-specific APIs**—If you are using ORS-specific APIs, you can use GetSiperianObjectCompatibility.

GetSystemTrustSettings

GetSystemTrustSettings fetches the system-specific trust settings for the specified columns.

The request contains a list of columns and a system. The response contains a list of trust setting objects in the same order as the list of columns.

Note: You can configure GetSystemTrustSettings requests according to a specific system when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the getSystemTrustSettings request:

- **Fetch the system-specific trust settings for a set of columns**.

Related SIF Requests

"GetTrustGraphData" on page 109, "GetTrustScore" on page 109

GetTaskLineage

The **GetTaskLineage** API retrieves the following information, depending on the request parameter settings:

- The closed tasks for a specified user.
- The closed tasks for a specified user that are part of an active workflow process.
- The closed tasks for a specified user that are part of an completed workflow process.
- The open tasks that have a task in their lineage that is owned by the specified user.
- The lineage for a specific task.

TaskData

The `TaskData` object contains information about a task.

The following table lists the TaskData fields that you can configure:

Field	Description
TaskRecord	A link to a data record associated with a task.
Comment	An optional task comment.
TaskType	The task type.
SubjectAreaUID	The UID of the task subject area.
Title	The task title.
TaskID	The ROWID of the task. Cannot be set by user.
DueDate	The date when the task is due.
Priority	The priority of the task. `1`: High priority. `0`: Normal priority. The default is `0`. `-1`: Low priority.
StatusEnum	The workflow status. The default is `TaskStatusEnum.OPEN`.
OwnerUID	The user or role ID to whom the task is assigned.
InteractionID	The Interaction ID.
WorkflowProcessID	The ID of the workflow process that contains the task. Cannot be set by user.

Field	Description
CreateDate	The date when the task was created. Cannot be set by user.
Creator	The name of the user who created the task. Cannot be set by user.
LastUpdateDate	The date when the task was updated. Cannot be set by user.
LastUpdatedBy	The name of the user who updated the task. Cannot be set by user.
PreviousOwner	The name of the user or role to whom the task was previously assigned. The value is Null if the task is new or has not been assigned. Cannot be set by user.

TaskRecord

The `TaskRecord` object contains information about a record.

The following table describes the TaskRecord fields:

Field	Description
SiperianObjectUID	An identifier for an object in Informatica MDM Hub.
RecordKey	An identifier for a record in Informatica MDM Hub.
MatchRuleUID	An identifier for a match rule in Informatica MDM Hub. Only merge tasks require a MatchRuleUID.

Required Request Parameters

The **GetTaskLineage** request does not have required parameters.

Optional Request Parameters

The information that the **GetTaskLineage** response returns is based on the values in the optional request parameters. The following table describes the optional request parameters:

Parameter	Value
TaskID	The TaskID parameter identifies the task.
OwnerUID	The ID of the user to whom the task belongs.
WorkflowStatus	The status of the workflow process.
TaskPosition	If TaskPosition=`USER`, the response returns the task assigned to the specified user. If TaskPosition=`WORKFLOW`, the response returns the task that is currently active in the workflow process.

The following table describes the parameter values necessary to get the tasks completed by a specified user:

Parameter	Value
TaskID	NULL
OwnerUID	The user ID.
WorkflowStatus	ANY
TaskPosition	USER

The following table describes the parameter values necessary to get the tasks completed by a specified user that are part of an active workflow process:

Parameter	Value
TaskID	NULL
OwnerUID	The user ID.
WorkflowStatus	OPEN
TaskPosition	USER

The following table describes the parameter values necessary to get the tasks completed by a specified user that are part of a completed workflow process:

Parameter	Value
TaskID	NULL
OwnerUID	The user ID.
WorkflowStatus	CLOSED
TaskPosition	USER

The following table describes the parameter values necessary to get the open tasks that have a task in their lineage that is owned by the specified user:

Parameter	Value
TaskID	NULL
OwnerUID	The user ID.
WorkflowStatus	OPEN
TaskPosition	WORKFLOW

The following table describes the parameter values necessary to get the lineage for a specified task:

Parameter	Value
TaskID	The task ID.
OwnerUID	The OwnerUID value is ignored.
WorkflowStatus	The WorkflowStatus value is ignored.
TaskPosition	The TaskPosition value is ignored.

Response Fields

The following table describes the fields returned in the**GetTaskLineage** response:

Field	Description
Title	Contains the task title.
TaskType	Contains the task type.
Status	Contains the task status.

GetTaskLineage Request Usage Example

The code in the following example retrieves a list of tasks belonging to the user named '`siftester`' and are overdue.

```
GetTaskLineageRequest request = new GetTaskLineageRequest();
request.setOwner("siftester");
GetTaskLineageResponse response = (GetTaskLineageResponse) sipClient.process(request);
```

GetTaskLineage Response Usage Example

The code in the following example prints out the task information that the **GetTaskLineage** response returns:

```
GetTaskLineageResponse response = (GetTaskLineageResponse) sipClient.process(request);
int i=0;
for(Iterator iter=response.getTaskList().iterator(); iter.hasNext();)
{ //iterate through response records
  System.out.println("Printing task " + i);
  TaskMetaData task = (TaskMetaData) iter.next();
  System.out.println(task.getTitle()+", "+task.getTaskType()+", "+task.getStatus());
}
```

GetTasks

The **GetTasks** API retrieves a lists of tasks and task details. Optional parameters allow you to filter the tasks that **GetTasks** returns.

Required Parameters

The **GetTasks** request does not have required parameters.

Optional Parameters

Use the optional parameters to filter the list of tasks that **GetTasks** returns. The following table describes the optional parameters.

Parameter	Description
TaskMetadata	Contains the filter criteria to apply to tasks for the search. By default, **GetTasks** searches for tasks with priority=NORMAL and status=OPEN. To search for tasks with other priorities or statuses, specify the priority and status in the TaskMetadata parameter.
OverdueOnly	If set to `true`, **GetTasks** only returns tasks with due dates that have already passed.
SummaryOnly	If set to `true`, **GetTasks** only returns the task metadata. The records and comments associated with the task are not returned.
Unassigned	If set to `true`, **GetTasks** only returns unassigned tasks.
CanBeAssignedToUser	If set to `true`, **GetTasks** only returns tasks that can be assigned to the user specified in the **GetTasks** request.
BDDApplicationName	If set to `true`, **GetTasks** only returns tasks for an IDD instance.
DisablePaging	The default value of DisablePaging is `false`. If set to `false`, paging is enabled and a search token is returned. Use **GetSearchResults** to fetch subsequent pages of search results. If set to `true`, paging is disabled.

Response Fields

The **GetTasks** response returns a list of tasks that are filtered by the parameters specified in the **GetTasks** request. If DisablePaging is set to `false`, a search token is also returned.

Use Case

This is the common scenario for using the **GetTasks** request:

- Fetch a set of tasks that match the criteria specified in the request.

GetTasks Request Usage Example

The following example retrieves the overdue tasks assigned to the user named 'admin'.

```
GetTasksRequest request = new GetTasksRequest();
TaskMetaData taskMetadata = new TaskMetaData();
taskMetadata.setOwnerUid("USER.admin");
request.setTaskMetaData(taskMetadata);
request.setOverdueOnly(true);
GetTasksResponse response = (GetTasksResponse) sipClient.process(request);
```

Use the following line of code to return tasks with a priority of 1:

```
taskMetadata.setPriority(1);
```

Use the following line of code to return the task that has a task ID of '1234':

```
taskMetadata.setTaskId("1234");
```

GetTasks Response Usage Example

The code in the following example shows how to print out the records returned by the **GetTasks** response.

```
GetTasksResponse response = new GetTasksResponse();
int i=0;
for(Iterator iter=response.getTaskList().iterator(); iter.hasNext();)
{ //iterate through response records
  System.out.println("Printing task " + i);
  TaskMetaData task = (TaskMetaData) iter.next();
  System.out.println(task.getTitle()+", "+task.getTaskType()+", "+task.getStatus());
}
```

Related SIF Request

"GetSearchResults " on page 101

GetTrustGraphData

GetTrustGraphData request provides the information needed to plot a trust decay curve.

The request contains a TrustSetting, which specifies the graph type, the time units, and other parameters of the required graph. The response contains a list of trust values and dates that define the graph.

Note: You can configure GetTrustGraphData requests only for "no system" when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the GetTrustGraphData request:

- **In an application with a custom-designed UI, display a trust graph** —If you have a custom UI and must display a trust graph, use GetTrustGraphData to get the data on which the graph is based.

Related SIF Requests

"GetSystemTrustSettings" on page 103, "GetTrustScore" on page 109

GetTrustScore

GetTrustScore computes the trust score for a specified column, based on the specified trust override. The column must be trust-enabled in the Schema Manager of the Hub console. The trust score (type float) of the Admin system will be returned.

The request contains a column UID and a key identifying the base object record. The response contains the trust score.

Use Case

This is the common scenario for using the GetTrustScore request:

- **Compute the trust score for a column**—If you are displaying a record, you can use GetTrustScore to display that information about a column.

Usage Example

The following example retrieves the trust score for column FIRST_NAME on base object C_CONTACT for the record with rowid = 3:

```
GetTrustScoreRequest request = new GetTrustScoreRequest();
request.setColumnUid("COLUMN.C_CONTACT|FIRST_NAME"); // Required
```

```
request.setRecordKey(RecordKey.rowid("3")); // Required
GetTrustScoreResponse response = (GetTrustScoreResponse) sipClient.process(request);
```

Related SIF Requests

"GetSystemTrustSettings" on page 103, "GetTrustGraphData" on page 109

GetUnmergedRecordCount

GetUnmergedRecordCount reports the number of records that are not merged—either all such records or those assigned to the current user.

The request supplies the table and a boolean value that specifies whether or not to restrict the count to records assigned to the user. The response contains the number of unmerged records.

Note: You can configure GetUnmergedRecordCount requests only for "no system" when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the GetUnmergedRecordCount request:

- **In the data steward queue management screen in a custom UI, display the number of unmerged records**—If you have a custom UI with a data steward queue management screen in a custom UI, you can use this request to display the number of unmerged records.

Related SIF Requests

"AssignUnmergedRecords" on page 58, "ReassignRecords" on page 123

GetXrefForEffectiveDate

The GetXrefForEffectiveDate API retrieves multiple XREF records for the specified effective date. The response to a GetXrefForEffectiveDate request contains the aggregate period start date, and the aggregate period end date. Aggregate period is the intersection of all the periods that encompass the specified effective date.

Required Parameters

The following table lists and describes the parameters that are required by the GetXrefForEffectiveDate API:

Parameter	Description
SiperianObjectUid	Name and type of base object or package for which you must get the XREF for a specified effective date.
RecordKey	Key to uniquely identify the record for which you must get the XREF for a specified effective date.
SystemNames	Names of systems for which XREF and XREF history must be retrieved.
EffectiveDate	The date for which you must retrieve values of the base object. **Note:** EffectiveDate must be used only for timeline-enabled base objects.

Optional Parameters

The following table lists and describes the optional parameters that are used by the GetXrefForEffectiveDate API:

Parameter	Description
HistoryDate	The date for which you retrieve base object data that is effective at the specified point in time. If HistoryDate is equal to or earlier than HIST_CREATE_DATE, no records are returned. For Oracle environments, milliseconds for HistoryDate are truncated. If HistoryDate is later than HIST_CREATE_DATE by less than a second, no records are returned. If you have Daylight Saving Time (DST) enabled for the operating system and HistoryDate is later than HIST_CREATE_DATE by less than one hour during DST, no records are returned. **Note:** HistoryDate must be used only for timeline-enabled base objects.
OtherPeriods	Periods that do not encompass the specified effective date. If set to `true`, XREF records are retrieved for all periods that do not encompass the specified effective date. The default is `false` and the GetXrefForEffectiveDate API retrieves XREF records for periods that encompass the specified effective date.
RecordStates	System state of the XREF record. The record can be in ACTIVE, PENDING, or DELETED state.
DataFilter	Used by IDD to apply security filter.

Use Case

The following is a typical scenario in which the GetXrefForEffectiveDate request is used:

Determining the address of residence of a client for a specific date — If your clients change their address of residence frequently, and you need to know what their address was on a specific effective date, you can use the GetXrefForEffectiveDate API in combination with the PreviewBVT API to get the address that was effective on the specific date. The cross-reference records used by the PreviewBVT API to calculate the address for the specified effective date are used by the GetXrefForEffectiveDate API to retrieve the address that was effective on the specified effective date.

Usage Example

The following example shows how to retrieve XREFs for a record with ROWID_OBJECT = 28, for an effective date 10/10/2011.

```
GetXrefForEffectiveDateRequest req=new GetXrefForEffectiveDateRequest();
req.setUsername("admin");
Password passwd=new Password();
passwd.setPassword("admin");
passwd.setEncrypted(false);
req.setOrsId("orcl.informatica.com-cmx_ors");
RecordKey rec=new RecordKey();
rec.setRowid("28");
req.setRecordKey(rec);
req.setSiperianObjectUid("BASE_OBJECT.C_BO");
Date date=new Date(2011, 10, 10);
req.setEffectiveDate(date);
GetXrefForEffectiveDateResponse response =
(GetXrefForEffectiveDateResponse)client.process(req);
```

Related SIF Requests

" PreviewBVT" on page 115

Link

Note: The Link API is deprecated.

Link links two or more base object records using the specified groupRecordKey as the group ID. Unlike a merge operation, when records are linked, the original base object records continue to exist and the cross reference records are not directly associated with the grouping record. However, the cross reference records are grouped together in a link group with the rowid of the groupRecordKey specified in the LinkRequest. If the records specified for linking have been previously linked, then nothing is changed and the API returns a success message.

In order to be able to use the Link request on a base object, the base object must first be configured to be a link-style BO instead of a merge-style BO. This option can be configured in the Schema Manager of the Hub Console.

In order to use a link group, the "GetBvt" on page 94 request must be invoked. This retrieves the best version of truth (BVT) for the specified link group accounting for the combined cross reference records of all base object records in the link group.

Related SIF Requests

"GetBvt" on page 94, "Unlink" on page 139

ListSiperianObjects

ListSiperianObjects returns basic metadata for a list of MDM Hub objects of the specified type. The metadata contains basic information such as SiperianObjectUID, display name, and description. To get an object's complete metadata, use "DescribeSiperianObject" on page 71.

Restrictions

Only admin users can access private resources through SIF API requests.

Required Parameters

The **ListSiperianObjects** request does not have required parameters.

Optional Parameters

Use the optional parameters to filter the list of objects returned by **ListSiperianObjects**. The following table describes the optional parameters.

Parameter	Description
ParentUID	Restricts the returned list to objects that are children of the parent specified by ParentUID.
ObjectType	Restricts the returned list to objects of the type specified by ObjectType.
UserResourcesOnly	When UserResourcesOnly is `true`, restricts the returned list to those objects for which the user has security resource privileges enabled. When UserResourcesOnly is `false`, the returned list is not restricted by security resource privileges.
PrivilegeType	Restricts the returned list to objects with a specific secure resource privilege enabled for the user. Possible values are `CREATE`, `DELETE`, `DESIGN`, `EXECUTE`, `MERGE`, `NONE`, `READ`, `UPDATE`, `WRITE`.

Response Fields

The following table describes the response fields:

Parameter	Description
Uid	Contains the SiperianObjectUID.
Name	Contains the object name.
DisplayName	Contains the display name.
Description	Contains the object description.

Use Case

The following scenario is a common use for the **ListSiperianObjects** request:

- For a particular base object, get a list of packages. If you have a custom UI, you can use this request to get a list of packages for a base object so the user can select a base object for the current operation.

ListSiperianObjects Request Usage Examples

The following example shows how to request the metadata of the columns for the base object "C_PARTY":

```
ListSiperianObjectsRequest request = new ListSiperianObjectsRequest();
request.setObjectType(SiperianObjectType.COLUMN);
request.setParentUid(SiperianObjectType.PACKAGE.makeUid("C_PARTY"));
request.setUserResourcesOnly(false); //ignores security access configuration of object
ListSiperianObjectsResponse response = (ListSiperianObjectsResponse)
sipClient.process(request);
```

When retrieving a list of users, you can set the parentUID to a Role UID to restrict the list to users that belong to a specific role. To retrieve a list of admin users, the role can be set to `ROLE.REQUEST_ADMIN_USERS_ONLY` using `request.setParentUid(SiperianObjectType.ROLE.makeUid("REQUEST_ADMIN_USERS_ONLY"))`. The following example shows how to request the metadata of users that belong to the role of `Manager`.

```
ListSiperianObjectsRequest request = new ListSiperianObjectsRequest();
request.setObjectType(SiperianObjectType.USER);
request.setParentUid(SiperianObjectType.ROLE.makeUid("Manager"));
request.setUserResourcesOnly(false); //ignores security access configuration of object
ListSiperianObjectsResponse response = (ListSiperianObjectsResponse)
sipClient.process(request);
```

ListSiperianObjects Response Usage Example

The following example shows how to print out the object metadata returned by the **ListSiperianObjects** response:

```
ListSiperianObjectsResponse response = new ListSiperianObjectsResponse();
int i=0;
for(Iterator iter=response.getMetaDataObjectList().iterator(); iter.hasNext();) { //
iterate through metadata objects
  System.out.println("Printing metadata object " + i);
  MetaDataBase metadata = (MetaDataBase) iter.next();
  System.out.println("/tuid="+metadata.getUid());
  System.out.println("/tname="+metadata.getName());
  System.out.println("/tdisplay name="+metadata.getDisplayName());
  System.out.println("/tdescription="+metadata.getDescription());
}
```

Related SIF Requests

"DescribeSiperianObject" on page 71

Merge

Merge merges two base object records, creating a single, consolidated base object record by merging all the XREF records from the two base objects.

When two records are merged, one is designated the source record, one is designated the target record. The request merges the source record into the target record. This means that after the merge the ROWID for the combined record is that of the target record. All foreign keys pointing to the source record now point to the target record.

For example, there may be one base object record with the name “Alex Watson” and another with the name “Alexander Watson”; each base object record has its own set of cross reference records. These records are determined to represent the same person so the records are merged. The result is a single base object record that has all the cross reference records from the original two base object records. The consolidated value for each field in the record is determined by the trust configuration.

Important: When you merge two records, Informatica MDM Hub does not check the match status of the records, it just merges the records as you specify. Using this class, it is possible to merge two completely dissimilar records, resulting in a nonsense record. For more information about merging, see the *Multidomain MDM Configuration Guide*.

Note: An alternate to Merge is Multimerge, which can be used to merge two or more records in a single operation.

For more information, refer to the Merge Settings Tab on the Match/Merge Setup Details dialog in the MDM Hub Schema Manager, the *Multidomain MDM Configuration Guide*, and the *Multidomain MDM Data Steward Guide*.

State Management

When you merge records in a base object with state management enabled, you can merge records in any state. The target ROWID survives in the base object regardless of the hub state. Survivorship of values are based on the trust scores and the last updated date of the source record.

For more information on state management, see the *Multidomain MDM Configuration Guide*, and the *Multidomain MDM Data Steward Guide*.

Use Case

The following scenario uses the merge request:

Merge used with GetMatchedRecords. You can use GetMatchedRecords to get a list of match candidates for a specified record. You can then display that list in a UI. If the user selects one of the candidate records, you can use merge to merge the two records.

Related SIF Requests

“GetMatchedRecords” on page 98, “Unmerge ” on page 139

MultiMerge

MultiMerge merges multiple base object records that have been identified as representing the same object and allows specifying the field level overrides for the merged record. MultiMerge is a more generic form of the “Merge ” on page 114 request and should be used for merging groups of records. Merge API should be used for pair-wise merges.

For example, there may be multiple base object records with the same account number “1234567” and account type “Personal”. Each base object record has its own set of cross reference records. When these records are merged with the call to the MultiMerge request, the result is a single BO record with that has the cross references from all the merged base object records. The consolidated value for each field in the

merged record is either determined through the survivorship rules based on the cross references of the records that are being merged or they are specified through the override values in the API.

When you use the MultiMerge API to specify an override value, a cross-reference record with a source system of SYS0 is created for the override values. The SYS0 source system ensures that the override values have the highest trust score.

PreviewBVT

The PreviewBVT API enables you to preview what a base object record would look like if you merge a specified set of records or apply pending updates to the records.

Required Parameters

The following table lists and describes the parameters that are required by the PreviewBVT API:

Parameter	Description
SiperianObjectUid	Name and type of base object or package that you need to use for previewing BVT.
RecordKeyList	Keys of records for which you need to preview the BVT. **Note:** If you include multiple records in the record key list, the BVT of only the first record can be previewed.
FilterClause	An SQL WHERE clause that can be applied to an XREF table. If you specify one record key for the RecordKeyList parameter, FilterClause is used to preview the effect of applying pending updates.

Optional Parameters

The following table lists and describes the optional parameters that are used by the PreviewBVT API:

Parameter	Description
EffectiveDate	The date for which values of the base object must be retrieved. **Note:** EffectiveDate must be used only for timeline-enabled base objects.
HistoryDate	The date for which you retrieve base object data that is effective at the specified point in time. If HistoryDate is equal to or earlier than HIST_CREATE_DATE, no records are returned. For Oracle environments, milliseconds for HistoryDate are truncated. If HistoryDate is later than HIST_CREATE_DATE by less than a second, no records are returned. If you have Daylight Saving Time (DST) enabled for the operating system and HistoryDate is later than HIST_CREATE_DATE by less than one hour during DST, no records are returned. **Note:** HistoryDate must be used only for timeline-enabled base objects.
PromotePairedXrefs	Ensures that the record preview matches what a base object record would look like after the record is promoted. If the approval tasks might have active and pending cross-reference records from the same source system, set the property to `true`. Default is `false`.

Use Case

The following is a typical scenario for using the PreviewBVT request:

Preview the BVT for base objects for which the trust level may change over time — Trust level for base object columns change over time and only the latest value reflects in the base objects. You can use the PreviewBVT request to preview the BVT value for a base object record at a specific point in time (past, present, or future).

Determine the value of a record for a specific effective date — If a record has cross-references with more than one period of effectiveness, then the PreviewBVT API can be used to calculate the BVT value of the record for a specific effective date.

Usage Example

The following example shows the merge preview of three records with ROWID_OBJECTs 1, 2, and 3 into a record with ROWID_OBJECT = 1, using the package P_PARTY.

```
PreviewBvtRequest request = new PreviewBvtRequest ();
request.setSiperianObjectUid(SiperianObjectType.PACKAGE.makeUid("C_PARTY"));
ArrayList recordKeys = new ArrayList();
recordKeys.add(RecordKey.rowid("1"));
recordKeys.add(RecordKey.rowid("2"));
recordKeys.add(RecordKey.rowid("3"));
request.setRecordKeyList (recordKeys);
PreviewBvtResponse response = (PreviewBvtResponse) sipClient.process(request);
```

PromotePendingXrefs

PromotePendingXref promotes or flags for promotion the XREF records specified in the request.

State Management

Promote means to change the state of a record from PENDING to ACTIVE. When the flagForPromote option is set, then this API request will queue the specified xref records for promotion using the next run of the PROMOTE batch process. Otherwise, the request will immediately promote the specified xref records from PENDING to ACTIVE. Here's the behavior of this request based on various XREF states:

- ACTIVE and DELETED records will return an error.
- PENDING records will be made ACTIVE.

Required Parameters

The following table lists and describes the parameters that are required by the PromotePendingXref API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be promoted.
XrefKey	Key to uniquely identify the record to be promoted.
SystemName	Name of system for which XREF must be promoted.
SourceKey	Source key of the cross-reference record that must be promoted.

Optional Parameters

The following table lists and describes the optional parameters that are used by the PromotePendingXref API:

Parameter	Description
ColumnNames	Names of columns that must be promoted to the ACTIVE state. Data is lost after promotion, in case of columns that are not specified. This parameter is only available for online promotion and is ignored when the FlagForPromote parameter is set to `true`.
FlagForPromote	Flags records for promote when set to `true`. The records are promoted when the batch process is executed in the MDM Hub.
PeriodStartDate	This parameter is used to specify the period start date for timeline-enabled base objects.
PeriodEndDate	This parameter is used to specify the period end date for timeline-enabled base objects.

Usage Example

The following example immediately promotes the "FIRST_NAME" and "LAST_NAME" fields for the XREF record with sourceKey=1234 and system=CRM in the package CUSTOMER_UPDATE. If the XREF record is ACTIVE or DELETED, an error will be returned. If the XREF record is PENDING, it will be made ACTIVE.

```
PromotePendingXrefsRequest request = new PromotePendingXrefsRequest();
ArrayList columnNames = new ArrayList();
columnNames.add("FIRST_NAME");
columnNames.add("LAST_NAME");
request.setColumnNames(columnNames); // Optional
XrefKey xrefKey = new XrefKey();
xrefKey.setSourceKey("1234");
xrefKey.setSystemName("CRM");
ArrayList xrefKeys = new ArrayList();
xrefKeys.add(xrefKey);
request.setXrefKeys(xrefKeys); // Required
request.setSiperianObjectUID("PACKAGE.CUSTOMER_UPDATE"); //Required

PromotePendingXrefsResponse response = (PromotePendingXrefsResponse)
sipClient.process(request);
```

The following example flags the XREF record with sourceKey=1234 and system=CRM in the package CUSTOMER_UPDATE for promotion the next time the Promote batch process is run.

```
PromotePendingXrefsRequest request = new PromotePendingXrefsRequest();
XrefKey xrefKey = new XrefKey();
xrefKey.setSourceKey("1234");
xrefKey.setSystemName("CRM");
ArrayList xrefKeys = new ArrayList();
xrefKeys.add(xrefKey);
request.setFlagForPromote(true); // Optional
request.setXrefKeys(xrefKeys); // Required
request.setSiperianObjectUID("PACKAGE.CUSTOMER_UPDATE"); //Required

PromotePendingXrefsResponse response = (PromotePendingXrefsResponse)
sipClient.process(request);
```

Related SIF Requests

"Delete" on page 69, "Restore" on page 127

Put

Use the Put API to create or update a record. Use an insert operation to create a cross-reference record in the cross-reference table, and then create a base object record in the base object table. Use an update operation to update a cross-reference record. Trust rules and validation rules determine if the base object record is also updated.

A record contains base object fields or package fields. A record can contain all fields or a subset of the fields. For example, a package may contain the fields FIRST_NAME and LAST_NAME. If you want to update only the LAST_NAME field, you can omit the FIRST_NAME field from the Put request. If you have not specified a value for a field, the Put API sets the value to null.

After a Put request, MDM Hub calculates the best version of the truth for all columns in the record, even for columns not included in the Put request.

State Management

You can enable state management in the Hub Console. If state management is enabled for a package, you can set the value of HUB_STATE_IND to specify the initial state of a record when you insert the record.

You cannot specify a HUB_STATE_IND value to change the state of an active record to deleted. You cannot specify a HUB_STATE_IND value to change the state of a pending record to active.

You can specify a HUB_STATE_IND value to change the state of an active record to pending. The Put call creates a pending cross-reference record with the same source as the active cross-reference record. The two cross-reference records have the same PKEY_SRC_OBJECT value, but one record is active and the other record is pending.

Use the Delete, Restore, or PromotePendingXrefs APIs to change the state of a record.

The following table lists the possible values for HUB_STATE_IND and the state these values represent:

HUB_STATE_IND Value	State	Description
1	ACTIVE	The record is approved. Default is 1.
0	PENDING	The record is not approved.
-1	DELETED	The record is not used in the MDM Hub processes.

Transaction Support

When executed within an EJB context, this request can be part of a transaction with other requests. If there is a failure in any of the requests within a transaction, the Put API rolls back the entire transaction.

Validation Rule Processing

The Put API applies the applicable rule with the highest downgrade percentage.

Trust Override

You can override the trust settings for a trust-enabled column as part of the Put request. The following portion of code sets the maximum trust to 90%, sets the minimum trust to 40%, sets the unit of time for the trust graph to months, sets the number of units over which the trust decays to 12, and sets the trust graph decay to linear.

```
field = new TrustOverrideField();
field.setName("CITY");
field.setStringValue(city);
field.setTrustOverride(TrustSetting.createTrustSettingCustom( 90, 40,
```

```
TrustTimeUnit.MONTH, 12, TrustGraphType.LINEAR));
data.setField(field);
```

If the Put request contains a `trustOverrideField` parameter, the parameter must have a value regardless of whether trust is enabled or disabled. The Put response ignores the `trustOverrideField` values for columns that do not have trust enabled.

Putable System Columns

A putable column is a system column that you can update or insert data into. A system column can be putable, not putable, or putable enabled by the user.

The following table describes the putable status of the system columns:

System Column	Putable Status
CM_DIRTY_IND	Never putable.
CONSOLIDATION_IND	Never putable.
CREATE_DATE	You can set as putable in the base object column properties.
CREATOR	You can set as putable in the base object column properties.
DELETED_BY	You can set as putable in the base object column properties.
DELETED_DATE	You can set as putable in the base object column properties.
DELETED_IND	You can set as putable in the base object column properties.
HUB_STATE_IND	You can set as putable in the base object column properties.
INTERACTION_ID	Always putable.
LAST_ROWID_SYSTEM	Never putable.
LAST_UPDATE_DATE	Always putable.
ROWID_OBJECT	Never putable.
UPDATED_BY	You can set as putable in the base object column properties.

Put API Behavior for the Create_Date Column

Multiple factors can affect the behavior of the Put API request on the Create_Date column.

The Putable property affects the Put insert operation behavior for Create_Date.

If you set Create_Date to Putable, the Put API populates the base object record and the cross-reference record with the value in the Put request. Null is a permissible value. If you do not provide a value in the request, the Put API populates the base object record and cross-reference record with the SYSDATE value.

If you do not set Create_Date to Putable, the Put API populates the base object record and cross-reference record with the SYSDATE value.

The Putable property affects the Put update operation behavior for Create_Date.

If you set Create_Date to Putable, the Put API populates the base object record and the cross-reference record with the value in the Put request. Null is a permissible value. If you do not provide a value in the request, the column retains the current value.

If you do not set Create_Date to Putable, the base object record and the cross-reference record column retains the current value. If the Hub creates a cross-reference record, the Put API populates the cross-reference record column with the SYSDATE value.

Put API Behavior for the Last_Update_Date Column

The Put API updates the LAST_UPDATE_DATE column with a value you specify in the Put request or with the SYSDATE value.

- If LAST_UPDATE_DATE is Putable and you provide a value for the LAST_UPDATE_DATE column in the Put request, the Put API populates the base object record and cross-reference record with this value. Null is a permissible value.
- If you do not provide a value for the LAST_UPDATE_DATE column in the Put request, the Put API populates the base object record and cross-reference record with the SYSDATE value.
- If LAST_UPDATE_DATE in not Putable, the Put API populates the base object record and cross-reference record with the SYSDATE value.

Put API Behavior for the SRC_LUD Column in the Cross-Reference

Multiple factors can affect the behavior of the Put API request on the SRC_LUD column in the cross-reference record.

- If LAST_UPDATE_DATE is Putable and you provide a LAST_UPDATE_DATE value, the Put API populates the SRC_LUD column with the LAST_UPDATE_DATE value.
- If you do not provide LAST_UPDATE_DATE value, but the Create_Date column is putable and you provide a Create_Date value, the Put API populates the SRC_LUD column with the Create_Date value.
- If you do not provide a LAST_UPDATE_DATE value and you do not provide a Create_Date value, the Put API populates the SRC_LUD column with the SYSDATE value.

Put API Behavior for Foreign Key Columns that are Not Nullable

If you configure a default value for a foreign key column and either specify a null value or do not specify a value, the Put API inserts the default value into the corresponding column of the cross-reference table.

If you do not configure a default value for a foreign key column and either specify a null value or do not specify a value in the Put request, the MDM Hub generates an exception.

Requirements

Consider the following requirements when using the Put API.

- You must specify the SystemName field to identify the cross-reference record and the system that provides the data.
- You must enable the Put API for the package.

Restrictions

Consider the following restrictions when using the Put API:

- You cannot insert a null value into a nonnullable column, such as a unique key column. You must provide a value for nonnullable columns because the Put API sets empty fields to null.
- You cannot use Put to insert or update a read-only column.
- You cannot use Put to insert or update a system column unless the column is putable.

- Validation rules only run if all columns used in the validation rule are present in the Put request.
- The Put request fails for packages that are based on base objects flagged for tokenization. A base object record needs to be tokenized if the ROWID_OBJECT value of the base object record is present in the associated dirty table.
- A Put request to update relationships can result in improperly formed relationship records. Use the AddRelationship API and the UpdateRelationship API to add or update relationship records.
- Escape special characters like ' and ~ with a backslash character.

Required Parameters

Use the required parameters to specify the data to insert or update and to specify which record is to receive the data. The following table describes the required parameters.

Parameter	Description
RecordKey	When you insert a record, use RecordKey to specify the system name of the record to insert. The system generates the ROWID_OBJECT value for the record. You cannot specify the ROWID_OBJECT value for a new record. If you specify the ROWID_OBJECT value for a record that does not exist, the Put request fails. When you update a record, use RecordKey to specify the system name, ROWID_OBJECT, and source key of the record to update. You can also specify the Global Business Identifier (GBID), as applicable.
Record	Contains the data to update or insert. The SiperianObjectUid field for the record specifies the type and the name of the base object or put-enabled package used to identify the affected base object and constrain the fields that you can set.

Optional Parameters

Use the optional parameters to specify a LastUpdateDate and request a source key.

The following table describes the optional parameters:

Parameter	Description
GenerateSourceKey	Use for keyless systems such as an application that does not persist source data. When set to `true`, the MDM Hub generates a source key if you do not specify one. If you insert a record from a keyless system, you can request the MDM Hub to generate a unique PKEY_SRC_OBJECT for the record. If you request a primary key when you insert a base object, the key generator generates a key and passes it to the Put part of this request. If the cross-reference ID does not exist when you update a base object, the Put API creates a cross-reference record. If the cross-reference ID exists when you update a base object, the Put API updates the cross-reference record.
BypassPostLoadUE	Determines if Put API calls the PostLoad user exit. Set to `true` to prevent recursive PostLoad user exit calls. For example, when a batch load calls the PostLoad user exit, which then calls a Put API that also calls the PostLoad user exit.
PeriodStartDate	Applicable for base objects for which you track data change events. Specifies the period start date for record versions.
PeriodEndDate	Applicable for base objects for which you track data change events. Specifies the period end date for record versions.

Parameter	Description
PeriodReferenceTime	Applicable for base objects for which you track data change events. Specifies a reference date within an effective period to identify a record version that you want to update. Default is null.
timelineAction	Applicable for base objects for which you track data change events. Specifies the action to perform on a record version during the load process. Use one of the following values: - 0. Adds a record version for a new effective period without maintaining contiguity between the record versions. - 1. Updates data in an existing record version. The effective period of the record does not change. - 2. Updates the effective period of a record version. An update to an effective period of a record version is through an increase or decrease of the effective start or end date. - 4. Adds a record version for a new effective period while maintaining contiguity between the record versions. Default is 0.
isFillOnGap	Applicable for base objects for which you track data change events. Ensures that contiguity between the effective dates of record versions is maintained when you add new record versions. If set to `true`, when you can add a new record version to the base object, the MDM Hub maintains the contiguity between effective periods of record versions. If set to `false`, the MDM Hub rejects any addition of record version that breaks the contiguity between effective periods of record versions. The default is false.

Response Fields

The Put response can contain the information described in the following table:

Field	Description
RecordKey	Contains the ROWID_OBJECT of the base object affected by the Put API. When performing a Put request using a ROWID_OBJECT for a base object record that has merged into another base object record, Put response returns the ROWID_OBJECT of the surviving base object record. RecordKey also contains a primary key if you set GenerateSourceKey to `true`.

ActionType	Indicates the action that the Put API performed. The possible values are:
	`- Insert`
	`- Update`
	`- Update cross-reference`
	`- No Action`
	The Tokenize API requires the value of ActionType. `Insert` indicates that a record has not yet been tokenized and tokens need to be created. `Update` and `Update cross-reference` indicate that a record has been tokenized and the tokens need to be regenerated.

Use Cases

The following examples are the common scenarios for using the Put request:

- The user provides data in a UI for creating or updating a record, and then the UI calls the Put API to insert or update the record.

- The Put API used in combination with the Tokenize API. A Put request followed by a Tokenize request inserts or updates the record and encodes it for matching. The Put response contains an action type string to use as an input to the Tokenize request. The Put API operation and the Tokenize operation can occur in the same transaction.
- The Put API used in combination with the Cleanse API. The most common use of cleanse is when the individual fields are cleansed before the record is passed on to the Put API.

Put Request Usage Example

The following example updates a record with ROWID_OBJECT key 782 using the package ADDRESS_UPDATE:

```
PutRequest request = new PutRequest();
request.setRecordKey(RecordKey.rowid("782", "SALES"));
Record record = new Record();
record.setSiperianObjectUid(SiperianObjectType.PACKAGE.makeUid("ADDRESS_UPDATE"));
record.setField( new Field("ADDRESS_LINE1", "123 Main St.") );
record.setField( new Field("CITY", "Anytown") );
request.setRecord( record );
PutResponse response = (PutResponse) sipClient.process(request);
```

ReassignRecords

ReassignRecords reassigns the specified records assigned for manual merge evaluation to another user.

The new user will now be responsible for these records.

Use Case

This is the common scenario for using the ReassignRecords request:

- **In a custom UI, allow data stewards to reassign the records in their queue**—If you have a custom UI, in the screen for managing data steward's queues, you might have a button that uses this request. This would allow data stewards to reassign the records in their queue.

Related SIF Requests

"AssignUnmergedRecords" on page 58, "ClearAssignedUnmergedRecords" on page 66

RegisterCustomIndex

A RegisterCustomIndex request registers user-defined indexes in the repository.

Some batch processes drop and re-create indexes based on the repository information. If you do not register custom indexes, the batch processes might drop them.

Request Parameters

The RegisterCustomIndex request contains the following parameters:

TableName

Name of the repository table.

RowidColList

A list of row ID column values that have custom indexes. MDM Hub ignores leading and trailing spaces in the row ID column. Separate each row ID column value with the tilde (~) character.

IndexType

Optional. Type of index to be registered. The IndexType parameter uses one of the following index types:

- FK. A foreign key index.

- PK. A primary key index.
- NI. A non-unique key index.
- UI. A unique key index.

Create and register only the non-unique type indexes. Default is NI.

Response Fields

The RegisterCustomIndex request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

A return code for the interaction.

Usage Example

The following code sample registers a custom index:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 RegisterCustomIndexRequest req = new RegisterCustomIndexRequest();
 req.setTableName(jobContext.getTableName());        // table name
 req.setRowidColList("SVR1.DA~SVR1.DB");              // column list
 RegisterCustomIndexResponse executed = (RegisterCustomIndexResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

RegisterCustomTableObject

A RegisterCustomTableObject request calls a register custom table object. You must register a custom stored procedure with Informatica MDM Hub to make it available to the users in the Batch Group tool. You can register the same custom job multiple times for different tables.

Request Parameters

The RegisterCustomTableObject request contains the following parameters:

RowidTable

Foreign key to the C_REPOS_TABLE.ROWID_TABLE table.

ObjFuncTypeCode

Code for the job type. Use code A for batch group custom jobs.

ObjFuncTypeDesc

Display name for the custom batch job in the Batch Groups tool.

ObjectName

Name of the custom job.

Response Fields

The RegisterCustomTableObject request returns the following fields:

InteractionId

An identifier for the request.

Message

A brief message about the status of the request.

RetCode

A return code for the interaction.

Usage Example

The following code sample registers the custom stored procedure CMXBG.EXAMPLE_JOB:

```
SiperianClient sipClient = SiperianClient.newSiperianClient(new
File( context.getTestPTTStartDir() + "siperian-client.properties" ) );
 RegisterCustomTableObjectRequest req = new RegisterCustomTableObjectRequest();
 req.setRowidTable(jobContext.getRowIdTable());        // rowid table. e.g. SVR1.44A
 req.setObjFuncTypeCode("A");
 req.setObjFuncTypeDesc("Custom call");
 req.setObjectName("CMXBG.EXAMPLE_JOB");
 RegisterCustomTableObjectResponse executed = (RegisterCustomTableObjectResponse)
sipClient.process( req );
 String errMessage = executed.getMessage();
 int rc = executed.getRetCode();
```

RegisterUsers

RegisterUsers enables an application to register selected users from the enterprise's authentication system (for example, LDAP) with Informatica MDM Hub. Then MDM Hub can use its existing access control capabilities to manage tasks like role assignments.

The application provides a list of user names, and MDM Hub fetches their information from the external system. Informatica MDM Hub ignores additional registrations of the same user profile from the external authentication system. However, it reports errors if the username is already registered using a different, or no, external profile, or if the name does not exist in the external authentication system.

The MDM Hub registers the users within a transaction. If an error occurs, it rolls back all changes.

Provisioned users can be grouped and assigned MDM Hub security roles using the MDM Hub Console. For more information, see the *Multidomain MDM Configuration Guide*.

Automatically provisioned users can be removed from the Informatica user database either using the Informatica Administration Console or using "UnregisterUsers" on page 141.

Transaction Support

When executed within an EJB context, this request can be part of a transaction with other requests. If there is a failure in any of the requests within a transaction, the entire transaction is rolled back.

Use Case

This is the common scenario for using the RegisterUsers request:

- **Checking external authentication**—Before you start a logical unit of work, check to see what the user is authorized to do.

Related SIF Requests

"Authenticate" on page 60

RemoveMatchedRecords

RemoveMatchedRecords removes matches associated with a base object record from the <*base object*>_MTCH table.

For example, if you decide records B and C do not match to record A, you can remove the following match pairs from <*base object*>_MTCH table:

- A-B
- A-C
- B-A
- C-A

Request Parameters

The RemoveMatchedRecords request contains the following parameters:

SiperianObjectUid

Specifies the type and name of the base object or package. Required.

RecordKey

Specifies the record key that does not match with the list of record keys. Required.

RecordKeyList

Contains the list of record keys. Required.

Response Fields

The RemoveMatchedRecords API returns the following fields:

InteractionId

Contains the interaction ID.

Message

Contains a message regarding the status of the request.

Usage Example

The following code sample removes matches A-B, A-C, B-A, and C-A from C_PARTY_MTCH:

```
RemoveMatchedRecordsRequest request = new RemoveMatchedRecordsRequest();
 request.setSiperianObjectUid(SiperianObjectType.PACKAGE.makeUid("C_PARTY"));
 request.setRecordKey(RecordKey.sourceKey("A", "Acme"));
 ArrayList keys = new ArrayList();
 keys.add(RecordKey.sourceKey("B", "Acme"));
 keys.add(RecordKey.sourceKey("C", "Acme"));
 request.setRecordKeyList(keys);
 RemoveMatchedRecordsResponse response = sipClient.process(request);
```

ResetBatchGroup

ResetBatchGroupfinds the last execution status of the given batch group, and if its status is failed, sets it to incomplete. For more information about batch groups, see the *Multidomain MDM Configuration Guide*.

Use Case

This is the common scenario for using the ResetBatchGroup request:

- **Resetting a batch group after** "GetBatchGroupStatus" on page 93 **returns and unsuccessful status**

Related SIF Requests

"ExecuteBatchGroup" on page 77, "GetBatchGroupStatus" on page 93

Restore

Restore reinstates the specified XREF record(s) in the Hub. Restore changes the state of records from DELETED to ACTIVE state. If an attempt is made to restore an active or pending record, an error is returned. After an XREF record is restored, the state of the parent BO record will be active.

Required Parameters

The following table lists and describes the parameters that are required by the Restore API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object to be restored.
XrefKey	Key to uniquely identify the record to be restored.
SystemName	Name of the system for which the record must be restored.
SourceKey	Source key of the record that must be restored.

Optional Parameters

The following table lists and describes the optional parameters that are used by the Restore API:

Parameter	Description
PeriodStartDate	This parameter is used to specify the period start date for timeline-enabled objects.
PeriodEndDate	This parameter is used to specify the period end date for timeline-enabled base objects.

Usage Example

The following example restores the XREF record with sourceKey=1234 and system=CRM from the package CUSTOMER_UPDATE. If the XREF record is pending or active, an error will be returned. If the XREF record is deleted, it will be made active.

```
RestoreRequest request = new RestoreRequest();
XrefKey xrefKey = new XrefKey();
xrefKey.setSourceKey("1234");
xrefKey.setSystemName("CRM");
ArrayList xrefKeys = new ArrayList();
xrefKeys.add(xrefKey);
request.setXrefKeys(xrefKeys); // Required
request.setSiperianObjectUID("PACKAGE.CUSTOMER_UPDATE"); //Required

RestoreResponse response = (RestoreResponse) sipClient.process(request);
```

Related SIF Requests

"Delete" on page 69, "PromotePendingXrefs" on page 116

SearchHmQuery

SearchHmQuery is used to search HM Entities or Relationships. The filter, aggregate and sort criteria can reference any columns in the Display Packages associated with Entity Type / Relationship Type in the search request. The criteria can use any operators supported by the underlying database.

The value stored in GETLIST_LIMIT column of CMX_SYSTEM.C_REPOS_DATABASE table for the ORS determines the maximum number of records that can be returned. GetSearchResultsRequest can be used to get subsequent pages of records.

The request contains the HM configuration, the type of the entity or relationship sought, and an SQL specification of the query. The response contains the sought records and a search token to use to fetch additional data.

Use Case

This is the common scenario for using the SearchHmQuery request:

- **Search for specific HM entity or entities** — If you have Hierarchy Manager and have populated it with data, you can use SearchHmQuery to search for entities and relationships associated with one or more entities.

Related SIF Requests

"SearchRequestBase" on page 135, "GetSearchResults " on page 101, "SearchHmQuery" on page 128

SearchLookupValues

SearchLookupValues request searches for lookup values that match a given lookup display name (lookup code description).

This request is typically used with foreign key columns that have a large number of possible values. The request includes a lookup column, a lookup display value to search for, and a comparison operator. It also includes a record whose fields contain the lookup information. In each field, the name is the lookup (foreign key) value, and the value is the lookup display name. For detailed information on configuring relationships and lookups in the MDM Hub, see the *Multidomain MDM Configuration Guide*.

This request allows search criteria to be specified on the lookup display name compared to "GetLookupValues" on page 98 which retrieves all lookup values for a lookup column.

A system parameter determines the maximum number of records that can be returned. Use "GetSearchResults " on page 101 to get subsequent sets of records.

Use Case

This is the common scenario for using the SearchLookupValues request:

- **Search for lookup values**—In a custom UI, you can use SearchLookupValues to find that value from amongst the available lookup values.

Related SIF Requests

"GetLookupValue" on page 97, "GetLookupValues" on page 98

SearchMatch

A SearchMatch request generates a list of search results by identifying matching records in a package or base object using MDM Hub match columns and, optionally, match rule sets. For information on configuring match columns and match rule sets, see the *Multidomain MDM Configuration Guide*.

Records must be tokenized or the records cannot be returned as match results. Use the **Tokenize** API or batch job to tokenize records.

SearchMatch returns a list of matching records. This is unlike the Hub match batch process, which creates a list of match candidates in the Operational Reference Store that you retrieve using **GetMatchedRecords**.

When you perform an extended search, the default match rule set must have fuzzy match keys. If fuzzy match keys are absent, the **SearchMatch** API does not return search results.

For information about performing exact matches on fuzzy base objects, see Exact Matches on Fuzzy Base Objects.

Required Parameters

The required SearchMatch Request parameters are described in the following table:

Parameter	Description
RecordsToReturn	Number of records to return.
MatchColumnField or RecordsToMatch	Specifies which match columns or match rules are used for matching: - MatchColumnField: The name of a single match column or a list of match columns to use in matching. - RecordsToMatch: The Hub uses all possible match columns configured for the record or records listed in RecordsToMatch. The Hub does not use the match columns that are based on fields for which the records have no value. Use `MatchColumnUid` to restrict match columns further. The match columns that result from using the RecordsToMatch parameter override the match columns specified by MatchColumnField.
SiperianObjectUid	Package or base object to search.
MatchType	Specifies how match rules are used in the search. If you do not specify the MatchType, **SearchMatch** uses the default MatchType of `NONE`. You can specify one of the following match types: - `BOTH`. Uses the automatic and the manual merge match rules. - `AUTO`. Uses the automatic merge match rules. - `NONE`. Uses the automatic and the manual merge match rules. If you selected exact-match columns in the **Selected Match Columns** list, and a selected record contains a value for that exact-match column, the match process identifies duplicate records based on the exact-match column plus the match rules. - `DBFILTERED`. Increases performance when a fuzzy search is based on a nonselective term.

Undermatching when using MatchType AUTO and BOTH

The `AUTO` and `BOTH` MatchType may result in undermatching if the search request contains empty fields. Users may not provide data for every field when performing a search from a custom client application. If an empty field is configured as an exact match column in the match rules, any potential search result containing data in these fields is excluded from the search results returned by **SearchMatch**.

MatchType NONE

A SearchMatch with MatchType `NONE` can be performed in one of two ways:

- Without using match rules. The match is performed by giving all columns equal weight, with no required fields. Undermatching is avoided because empty fields in the search request are ignored, which prevents relevant records from being excluded from the search results. However, overmatching and misleading match scores can occur because nonselective fields, such as Gender, are given the same weight as selective fields, such as Customer ID.

- Using a match rule set that has Enable Search by Rules enabled in the Hub. The undermatching that can occur when using MatchType `AUTO` and `BOTH` is avoided because empty fields provided by the search request are ignored and not used to exclude search results. Overmatching and misleading match scores are avoided because the match rules give the columns an appropriate weight.

MatchType DBFILTERED

The DBFILTERED MatchType increases performance when a fuzzy search based on a nonselective term, such as Name="JOHN", also provides values for exact match columns. Nonselective fuzzy search terms provide excessive search results which cause **SearchMatch** to take longer than expected to complete. When the MatchType is DBFILTERED, **SearchMatch** performs in one of two ways:

- If the fuzzy search term is selective, for example, JONATHAN LIVINGSTONE, SearchMatch functions as it does when MatchType is set to `BOTH`. Initial filtering is performed using the match key ranges generated by the Hub, and then additional filtering is performed by the Process Server using the exact match columns. This type of filtering reduces the number of database joins required and provides optimal SearchMatch performance for selective fuzzy search terms.
- If the fuzzy search term is nonselective, for example, JOHN, initial filtering is done in the database on the exact match columns and match key ranges. This results in fewer records being returned to the Process Server for matching than would occur using the MatchType `AUTO` or `BOTH`. This type of filtering provides optimal SearchMatch performance for nonselective fuzzy search terms.

Consider the following when using the DBFILTERED MatchType:

- A DBFILTERED match cannot be performed using the following types of exact match columns:
  - Exact match columns that have match subtype, non-equal matching, null matching, or segment matching enabled.
  - Exact match columns that are based on a concatenation of base object columns.
- Ensure the lock on the schema has been released. API performance decreases when the schema is locked.
- DBFILTERED **SearchMatch** performance decreases as the number of fuzzy match rules increases. Create a match rule set to use specifically for DBFILTERED SearchMatch that is limited to the match rules essential for database filtering.
- Ensure low cardinality (preferably 1:1) between the base object providing the match key and the exact match columns used for filtering to increase database filtering performance.
- Add custom column indexes to some of the exact match columns used for filtering to improve database filtering performance. Use as few custom column indexes as possible to avoid decreasing batch job performance.

Optional Parameters

The optional SearchMatch Request parameters are described in the following table:

Parameter	Description
includePending	includePending determines if **SearchMatch** includes pending records in the results. `true`: **SearchMatch** includes pending records in results. "Enable match on pending records" must be enabled for the base object in the Match Merge Hub setup. `false`: **SearchMatch** does not include pending records in results. The default is `false`. This parameter has no effect for base objects for which State Management is not enabled. See the *Multidomain MDM Configuration Guide* for more information on State Management.
sortCriteria	A string containing a comma-separated list of column names and a sort direction. For example, "LAST_NAME ASC, FIRST_NAME ASC" returns the search results sorted by last name and then first name, in ascending order. Use DESC to sort in descending order. The results are returned in random order if you do not specify the sort order, unless the MatchType is `NONE`. If the MatchType is not `NONE`, RecordsToReturn is specified, and sortCriteria is not specified, then the records are sorted based on the match score.
MatchRuleSetUid	A string containing the name of a match rule set. If a match rule set is not specified using this parameter, SearchMatch uses the default match rule set. You must use one of the following formats: - A fully qualified SiperianObjectUid followed by the match rule set. For example, "MATCH_RULE_SET.C_PARTY|Main". You must include the MATCH_RULE_SET prefix. - Set to NULL to use the default match rule set. To set MatchRuleSetUid to NULL, omit MatchRuleSetUid from the **SearchMatch** request.
setDisablePaging	This parameter determines if paging is disabled. `true`: paging is disabled. `false`: paging is enabled. The default is `false`. Use **GetSearchResults** to fetch subsequent pages of search results.

Response Fields

The SearchMatch Response contains a list of matching records as well as the information described in the following table for each record:

Field	Description
DEFINITIVE_MATCH_IND	Indicates whether a match was made using a match rule enabled for automatic merging. Matches made using auto-merge match rules typically result in closer matches than those made using manual-merge match rules. `1`: Match made using an auto-merge match rule. `0`: Match made without using a manual-merge match rule.
RULESET_NAME	Indicates which match rule set was used to make the match. The value for RULESET_NAME is `GENERATED SEARCH` when the MatchType is `NONE`.

Field	Description
RULE_NUMBER	Indicates the rule number of the match rule that was used to make the match. The value for RULE_NUMBER is `1` when the MatchType is `NONE`.
MATCH_SCORE	Indicates the match score of the result. If MatchType is equal to `NONE`, SearchMatch returns the match score, if available, so that the search results can be ranked by the match score. The match score is ignored when sorting if you specify a sort order using the sortCriteria parameter.

Use Case

This is the common scenario for using the SearchMatch Request:

- Generate and then return a list of the possible matches in a given package or base object. Use the returned list of match candidates to merge or link records using **Merge**, **MultiMerge**, or **Link**.
- The user specifies search criteria in a UI and the UI calls SearchMatch to find similar records and display the results to the user for editing.

SearchMatch Request Usage Example

The code in the following example searches for a match to 'EXAMPLES CORP' in the Organization_Name match column of the PARTY_ADDRESS_READ_PKG package. The results will be sorted based on the values in the NAME column in descending order. SearchMatch will use the default MatchType of `NONE` since no MatchType is specified.

```
SearchMatchRequest request = new SearchMatchRequest();
request.setRecordsToReturn(5);
request.setSiperianObjectUid("PACKAGE.PARTY_ADDRESS_READ_PKG");
Field orgNameField = new Field("Organization_Name");
orgNameField.setStringValue("EXAMPLES CORP");
request.addMatchColumnField(orgNameField);
request.setSortCriteria("NAME DESC");
SearchMatchResponse response = (SearchMatchResponse) sipClient.process(request);
```

SearchMatch Response Usage Example

The code in the following example shows how to print out the records returned by the SearchMatch response.

```
SearchMatchResponse response = new SearchMatchResponse();
int i=0;
for(Iterator iter=response.getRecords().iterator(); iter.hasNext();) {
//iterate through matched records
  System.out.println("Printing matched record " + i);
  Record record = (Record) iter.next();
  BigDecimal definitiveMatch =
record.getField("DEFINITIVE_MATCH_IND").getBigDecimalValue();
  if(definitiveMatch.intValue()==1) System.out.println("Matched on an
auto-merge rule");
  Collection fields = record.getFields();
  for(Iterator fieldIter=fields.iterator(); fieldIter.hasNext();) {
//iterate through rest of fields
    Field f = (Field) fieldIter.next();
    System.out.println(f.getName() + ": " + f.getValue());
  }
}
```

Related SIF Requests

"Tokenize " on page 138, "GetMatchedRecords" on page 98, "GetSearchResults " on page 101, "SearchQuery " on page 133, "Merge " on page 114, "MultiMerge" on page 114, "Link" on page 112

SearchQuery

SearchQuery searches for records in a package, base object, or remote package based on an SQL condition clause. The condition clause can reference any column in the package, base object, or remote package and can use operators supported by the target database.

If you perform a SearchQuery in which you use a ROWID_OBJECT value for a base object record that is merged into another base object record, no records are returned. For example, two base object records, one with a ROWID_OBJECT value of ROWID_A and the other with a value of ROWID_B are merged. After the merge, the ROWID_OBJECT value of the surviving base object could be ROWID_A. In this case, if you perform a SearchQuery in which you use a ROWID_OBJECT of ROWID_B, no records are returned because a base object with a ROWID_OBJECT of ROWID_B no longer exists in the base object table.

On IBM DB2, you cannot use functions such as CAST in the sortCriteria field of the SearchQuery API. The select statement of the SearchQuery API uses the ORDER BY clause that you cannot combined with functions such as CAST.

On IBM DB2, you cannot use the `lower` function in the filterCriteria field of the SearchQuery API. IBM DB2 does not support the `lower` function on the graphic physical data type that corresponds to the NCHAR and NVARCHAR data type of the MDM Hub.

When you specify the effectiveDate parameter along with the filterCriteria parameter in SearchQuery, ensure that the columns that you specify in the filterCriteria parameter are qualified.

Informatica recommends that you use the filterCriteria parameter together with the filterParameter when possible so that the database can reuse the cached query for better performance.

For example, if you have a filterCriteria parameter of `lower(name)=lower`, the following sample shows how to combine the filterParameter in SearchQuery:

```
<urn:filterParameter>
<urn:stringValue>Peter</urn:stringValue>
</urn:filterParameter>
```

Note: Ensure that you list the parameters with the appropriate data types according to the database.

The following sample shows that the C_PARTY object qualifies the FIRST_NAME column:

```
<urn:orsId>orcl.informatica.com-MDM_SAMPLE</urn:orsId>
<urn:siperianObjectUid>BASE_OBJECT.C_PARTY</urn:siperianObjectUid>
<urn:filterCriteria>C_PARTY.FIRST_NAME = ?</urn:filterCriteria>
<urn:filterParameter>
<urn:stringValue>John</urn:stringValue>
</urn:filterParameter>
<urn:effectiveDate>2014-03-17T20:06:00.000</urn:effectiveDate>
```

Required Parameters

The following table lists and describes the parameters that the SearchQuery API requires:

Parameter	Description
SiperianObjectUid	Name and type of package, base object, XREF table, XREF history table, history table, or merge history table that you need to query.
RecordsToReturn	Sets the limit to the number of records that the SaerchQuery API must retrieve.

Parameter	Description
FilterCriteria	SQL clause to filter search results for the columns of the package that is queried. Use the FilterCriteria parameter to specify the literal expressions (FIRST_NAME = 'JOHN') or use it in combination with FilterParameter.
FilterParameter	Specifies the parameter values to filter as a list.

Optional Parameters

The following table describes the optional SearchQuery API parameters:

Parameter	Description
EffectiveDate	The date for which you must retrieve values of the base object. **Note:** Use EffectiveDate only for timeline-enabled base objects.
HistoryDate	The date for which you retrieve base object data that is effective at the specified point in time. If HistoryDate is equal to or earlier than HIST_CREATE_DATE, SearchQuery does not return any records. For Oracle environments, SearchQuery truncates milliseconds for HistoryDate. In the following situations, SearchQuery does not return any records: - HistoryDate is later than HIST_CREATE_DATE by less than a second - Daylight Saving Time (DST) is enabled for the operating system and HistoryDate is later than HIST_CREATE_DATE by less than one hour during DST **Note:** Use HistoryDate only for timeline-enabled base objects.
DisablePaging	If set to `true`, it disables the paging mechanism and you can retrieve multiple pages of data. Set the parameter set to `true` for queries that return a predictable number of rows. Use **GetSearchResults** to fetch subsequent pages of search results.
RecordStates	Specifies the Hub state indicator to use for filtering the search result.
JoinUids	Specifies a list of UIDs to join with SiperiaObjectUID.
RemoveDuplicates	If set to `true`, SearchQuery removes duplicates from the result set. Enable the parameter when there is a possibility of duplicates in the result set. The default is `false`.
AdvancedMode	If set to `true`, the advanced mode of search query processing is enabled. The default is `false`. When AdvancedMode is `true`, you can use FilterCriteria with the advanced operators EXISTS and COUNT. However, when AdvancedMode is `true`, you cannot use these operators in sortCriteria.
UncommittedRead	If set to `true`, SearchQuery returns results without waiting for pending updates to commit. If set to `false`, SearchQuery does not return results until all pending data changes are committed. The default is `false`.

Retrieving Large Record Sets

To control multiple records that the query must return and set the data page size for paging, you need the SearchRequestBase. The SearchRequestBase is the base class for search requests with parameters for paging and sorting.

Case Sensitivity

The SearchQuery API is case sensitive. Use the same case as in the Operational Reference Store when you specify a filter criteria. If the case is different from the one in the Operational Reference Store, records are

not found. However, to control the case sensitivity of SearchQuery, you can use the CASE_INDICATOR column in C_REPOS_TABLE. The values for the case indicator are UPPER, LOWER, and NULL. Specify a case indicator to indicate that all data in the corresponding table is in the case that you specify. The case indicator setting converts the filter criteria to the appropriate case. To implement searches that are not case sensitive, you do not require queries based on functions.

The following table describes the settings for the CASE_INDICATOR column:

Name	Description
UPPER	Converts the WHERE clause and any parameter of the query to uppercase before you run the request. All the data in the package in the request must be in uppercase.
LOWER	Converts the WHERE clause and any parameter of the query to lowercase before you run the request. All the data in the package in the request must be in lowercase.
NULL	Query does not require case conversions before you run the request. The case of data in the package that is in the request does not matter.

Use Case

The scenario where you search for records in a package uses the SearchQuery request. In a custom UI, use **SearchQuery** to allow a data steward to find a particular record.

Usage Example

The following example shows the search for a record from the package PARTY_ADDRESS_READ_PKG, where PARTY_FULL_NAME starts with 'WAGNER':

```
SearchQueryRequest request = new SearchQueryRequest();
request.setRecordsToReturn(5);
request.setSiperianObjectUid("PACKAGE.PARTY_ADDRESS_READ_PKG");
request.setFilterCriteria("PARTY_FULL_NAME LIKE ?");
ArrayList params = new ArrayList(2);
params.add(new Parameter("WAGNER%"));
request.setFilterParameter(params);
SearchQueryResponse response = (SearchQueryResponse) sipClient.process(request);
```

SearchRequestBase

SearchRequestBase is the base class for search requests (SearchQuery and SearchMatch) with parameters for paging and sorting.

Paging Support

The parameters for the paging mechanism to return large result sets are as follows:

- **Maximum number of records returned**—This parameter can be specified at the ORS level and it is stored in the CMX_SYSTEM.C_REPOS_DATABASE.GETLIST_LIMIT parameter. This limit takes precedence over the values specified using setRecordsToReturn(int). The search queries will be limited to the minimum value of the two for any search request. For SearchMatchRequest API, there are also additional parameters that can be specified on the Process Server to control the number of matches the Hub will attempt before returning the results.
- **Number of records**—setRecordsToReturn(int). When paging is enabled this parameter specifies the size of the first page of data returned by the search API. Subsequent pages can be returned using the GetSearchResultsRequest API. Alternatively, for requests that have paging disabled this method would specify the limit of the total number of rows returned by the search API.

- **The paging mechanism is enabled by default**— disable paging by using the setDisablePaging() methods of APIs such as SearchQuery and SearchMatch that support paging. If the setDisablePaging() method is set to `false`, then a search token is returned. The search token is used by GetSearchResults to return additional results.
- **The search token is deleted**—After a period of inactivity longer than the `sif.search.result.query.timeToLive.seconds` setting specified in the Hub Server properties. GetSearchResults calls made after the token is expired would result in an error.

For more information, refer to the SIF API Javadocs.

SearchResponseBase

SearchResponseBase is the base class for search responses.

Each response contains a list of records, a search token to use to fetch more results, and an optional count of matching records.

For more information, refer to the SIF API Javadocs.

SetPassword

SetPassword request changes the password for the user. The existing password must be specified in the request as well as the new password. Passwords are specified as Password objects. The password specified must adhere to the password policy configured in the Hub. After you change the password, acquire a write lock in the Hub Console to refresh the cache for the new password to take effect.

Use Case

This is the common scenario for using the SetPassword request:

- **Allow a Informatica MDM Hub administrator to set a password for a user within Informatica MDM Hub** — In an application for Informatica MDM Hub administrators, you can include functionality to allow the administrator to change passwords for the Informatica MDM Hub users.

SetRecordState

SetRecordState enables a client application to assign one of a predefined set of state indicator values to a specified set of base object records. The state indicator value is stored in the CONSOLIDATION_IND column. The consolidation indicator can be one of the following values:

Indicator Value	State Name	Description
1	CONSOLIDATED	This record has been consolidated (determined to be unique) and represents the best version of the truth.
2	UNMERGED	This record has gone through the match process and is ready to be consolidated.
3	QUEUED_FOR_MATCH	This record is a match candidate in the match batch that is being processed in the currently-executing match process.

Indicator Value	State Name	Description
4	NEWLY_LOADED	This record is new (load insert) or changed (load update) and needs to undergo the match process.
9	ON_HOLD	The data steward has put this record on hold until further notice. Any record can be put on hold regardless of its consolidation indicator value. The match and consolidate processes ignore on-hold records. For more information, see the *Multidomain MDM Data Steward Guide*.

Restrictions

Consider the following when using the **SetRecordState** API:

- This request cannot be used within a transaction.
- The consolidation state can only be set for base object records with a Hub_State_Ind value of 1, meaning the record has an ACTIVE hub status.

Required Parameters

Parameter	Description
RecordKey	Identifies the record that requires a state change. See "RecordKey" on page 154 for more information. **Note:** If the ROWID_OBJECT is provided, the systemName is not validated.
RecordState	The name of the state to which the record will be set. The state name can be one of the following: - CONSOLIDATED - UNMERGED - QUEUED_FOR_MATCH - NEWLY_LOADED - ON_HOLD
SiperianObjectUID	The package or base object used to identify the record that requires a state change.

Optional Parameters

The **SetRecordState** request does not have any optional parameters.

Response Fields

The following table describes the response fields:

Parameter	Description
Messsage	Contains the status message of the **SetRecordState** request.

Use Case

This is the common scenario for using the SetRecordState request:

- In a client application, allow a user to explicitly set the state of a record.

SetRecordState Request Usage Example

The following example sets the record state of the record with a ROWID_OBJECT value of 782 to QUEUED_FOR_MATCH using the ADDRESS_UPDATE package:

```
SetRecordStateRequest request = new SetRecordStateRequest();
request.setSiperianObjectUid("PACKAGE.ADDRESS_UPDATE");
request.setRecordState(RecordState.QUEUED_FOR_MATCH);
RecordKey recordKey = new RecordKey();
recordKey.setRowid("782");
request.addRecordKey(recordKey);
SetRecordStateResponse response = (SetRecordStateResponse) sipClient.process(request);
```

SetRecordState Response Usage Example

The following example prints the status message from the **SetRecordState** response:

```
//Construct the request
..
//Process the request
SetRecordStateResponse response =
(SetRecordStateResponse) sipClient.process(request);

//Print the message from the response object
System.out.println("Message: " + response.getMessage());
```

Tokenize

The **Tokenize** API generates the match keys that the match engine and the **SearchMatch** request use when performing fuzzy matches. The **Merge** request does not use these match keys. **Tokenize** can also regenerate match tokens for a record that was previously tokenized.

In Informatica MDM Hub, you can also manually generate match tokens or configure the Hub to generate match tokens after the load process completes. For more information, see the Batch Jobs Reference in the *Multidomain MDM Configuration Guide*.

If you request **Tokenize** to regenerate tokens for a record that has not changed since the previous Tokenize, the request succeeds and reports that it tokenized zero records.

Transaction Support

When executed within an EJB context, **Tokenize** can follow a **Put** or **CleansePut** request in a single transaction. If there is a failure in any of the requests within a transaction, the entire transaction is rolled back.

Required Parameters

The following table describes the required **Tokenize** parameters.

Parameter	Description
setRecordKey(RecordKey)	Indicates the key of the record to be tokenized.
setActionType(String)	Action type returned from a previous **Put** or **CleansePut** request. `Insert` indicates that the record has not been tokenized. `Update` and `Update xref` indicate that the record has been previously tokenized and the tokens need regenerated.
SiperianObjectUid	The name of the package or base object.

Use Cases

The following scenarios are the common uses for the **Tokenize** request:

- Use **Put** to insert or update the record in the package. Then call **Tokenize** to generate match keys.
- Call **CleansePut** to cleanse the data and insert or update the record in the package. Then call **Tokenize** to generate match keys.

Tokenize Request Usage Example

The following example shows how to generate match tokens for a record with ROWID_OBJECT 782 using the ADDRESS_UPDATE package. This record is being tokenized for the first time, so an ActionType of `Insert` is used. The **Put** response provides the value of ActionType.

```
TokenizeRequest request = new TokenizeRequest();
request.setSiperianObjectUid( SiperianObjectType.PACKAGE.makeUid( "ADDRESS_UPDATE" ) );
RecordKey recordKey = new RecordKey();
recordKey.setRowid("782");
request.setRecordKey( recordKey );
request.setActionType("Insert");
TokenizeResponse response = (TokenizeResponse) sipClient.process(request);
```

Related SIF Requests

"SearchMatch " on page 128, "CleansePut" on page 62, "Put " on page 118

Unlink

Unlink decouples two or more base object records with the group ID specified in the groupRecordKey field. The records being unlinked must have been previously linked using the Link API.

Unmerge

Unmerge unmerges two rows in a base object. **Unmerge** provides the same unmerge functionality as the Data Manager tool in the Data Steward workbench. This request restores all foreign keys updated in the merge.

Unmerging can take a long time to complete, so you may want to run the unmerge asynchronously. You cannot use unmerge within a transaction. For more information about unmerging and merging, see the *Multidomain MDM Configuration Guide*.

If the request is unsuccessful, the program throws an Informatica request exception. If the request is successful, then the response also indicates that the unmerge succeeded.

Note: You can configure Unmerge requests according to a specific system when using the Hub Console Audit Manager to audit requests made by external applications. Once auditing for a particular SIF API request is enabled, Informatica MDM Hub captures each SIF request invocation and response in the audit log. For more information, see the *Multidomain MDM Configuration Guide*.

Cross-reference Added Directly to a Base Object

Typically when a new cross-reference is inserted, a new base object record is created for it. But Put or CleansePut APIs and the "Load By ROWID" batch job can also add a cross-reference record directly to a base object. This means that the cross-reference record was never the only cross-reference for a base object record. When such a cross-reference record is identified to be unmerged, it is deleted from the system and there is no independent base object record to reinstate for it.

Linear Unmerge

During a linear unmerge, no attention is paid to the process by which the records were originally merged.

Consider a scenario where initially there are three base object records with ROWID_OBJECT values of 1, 2, and 3. Each of these base objects has a single cross-reference record with corresponding source keys 1, 2, and 3, and SALES as the SystemName value. If you merge ROWID_OBJECT 3 into 2, and then merge ROWID_OBJECT 2 into 1, then as a result of the merge a single base object record with ROWID_OBJECT 1, and three cross-reference records remain.

Now, if sourceKey 2 of the SALES system is targeted for a linear unmerge, then the base object of sourceKey 2 is reinstated and the cross-reference of sourceKey 2 is associated with the reinstated base object. The resulting base object records are as follows:

- ROWID_OBJECT=1 has two cross-reference records with sourceKeys 1 and 3.
- ROWID_OBJECT=2 has one cross-reference records with sourceKey 2.

Tree Unmerge

During a tree unmerge, the process by which the records were originally merged determines the outcome.

Consider a scenario where initially there are three base object records with ROWID_OBJECT values of 1, 2, and 3. Each of these base objects has a single cross-reference record with corresponding source keys 1, 2, and 3, and SALES as the SystemName value. If you merge ROWID_OBJECT 3 into 2, and then merge ROWID_OBJECT 2 into 1, then at the time record 2 was merged into record 1, record 3 had already been merged into 2. So the cross-references for sourceKeys 2 and 3 are removed from the base object and the base object record for sourceKey 2 is reinstated. The resulting base object records are as follows:

- ROWID_OBJECT=1 has one cross-reference record with sourceKey 1.
- ROWID_OBJECT=2 has two cross-reference records with sourceKeys 2 and 3.

  Note: In both cases, the consolidated field values in the base object record are recalculated after the unmerge.

Cascade Unmerge

Unmerge performs a cascade unmerge if this feature is enabled for this base object in the Schema Manager in the Hub Console. With cascade unmerge, when records in the parent object are unmerged, Informatica MDM Hub also unmerges affected records in the child base object.

Required Parameters

The following table lists and describes the parameters that are required by the Unmerge API:

Parameter	Description
SiperianObjectUid	Name and type of the package or base object containing the record to be unmerged.
SystemName	Name of the system for which the record must be unmerged.
SourceKey	Source key of the record that must be unmerged.
TreeUnmerge	Set to true for a tree unmerge operation or false for a linear unmerge operation.

Use Case

This is the common scenario for using the Unmerge request:

- In a custom UI, use unmerge to allow a data steward to manually unmerge records. A data steward might unmerge records when, for example, a merge of two records was done in error.

Related SIF Requests

"Merge " on page 114

UnregisterUsers

UnregisterUsers enables an application to unregister selected users from Informatica MDM Hub. The application sends a list of user names, presumably representing users in the enterprise's authentication system (for example, LDAP).

The application provides a list of user names, and Informatica MDM Hub removes them. Informatica MDM Hub ignores unregistrations of users that are not registered in Informatica MDM Hub.

Informatica MDM Hub unregisters the users within a transaction. If an error occurs, it rolls back all changes.

Transaction Support

When executed within an EJB context, this request can be part of a transaction with other requests. If there is a failure in any of the requests within a transaction, the entire transaction is rolled back.

Use Case

This is the common scenario for using the UnregisterUsers request:

- **Bulk unregistering with Informatica MDM Hub** — Based on external authentication information, you can use unregistreUsers to bulk unregister users.

Related SIF Requests

"RegisterUsers" on page 125

UpdateMatchRecord

UpdateMatchRecord creates or updates records in the match table. The match table contains the pairs of matched records in a base object after you run a match process on the base object. Use the UpdateMatchRecord request to add or update records that qualify for a merge with the record being matched.

Request Parameters

An UpdateMatchRecord request contains the following parameters:

recordKey

Identifies the record that is a match to the record being matched.

matchedRecordKey

Identifies the record for which you add a match in the match table. The row ID of the matched record is the row ID of the consolidated record when you perform a merge operation.

autoMerge

Optional. Indicates whether to merge the matched records when the records are matched.

Set to true to merge matched records. Default is false.

matchReverse

Optional. Indicates whether you want to reverse the order of the match.

For example, if record 1 is the matched record and record 2 matches record 1, then record 2 is merged with record 1. The row ID of the consolidated record is that of record 1. If the matchReverse indicator is set to true, record 1 is merged with record 2. The row ID of the consolidated record is that of record 2.

Set to true to reverse the match order. Default is false.

matchRuleUid

Optional. Identifies the match rule you use to match the records.

taskId

Optional. Identifies the task that you want to associate with the match table record.

Response

The UpdateMatchRecord request returns a success message on updating the match table.

Usage Example

The following sample code adds a match record for the records with keys 111 and 222 in the match table:

```
UpdateMatchRecordRequest request = new UpdateMatchRecordRequest();
request.setMatchRuleUid(SiperianObjectType.MATCH_RULE.makeUid("MyMatchRule"));
request.setRecordKey(RecordKey.sourceKey("111", "Acme"));
request.setMatchedRecordKey(RecordKey.sourceKey("222", "Acme"));
UpdateMatchRecordResponse response = (UpdateMatchRecordResponse)
sipClient.process(request);
```

UpdateRelationship

UpdateRelationship Hierarchy Manager request updates a Relationship between two Entities. The existing relationship record is updated if the Start Date, End Date, or custom columns for the Relationship record are modified. If the update request changes the Hierarchy, Relationship Type, or one or both Entities in the Relationship, the current Relationship record is deleted (see "DeleteRelationship" on page 71) and a new Relationship record is added (see "AddRelationship" on page 57). When a new Relationship record is added, the RecordKey returned in the UpdateRelationshipResponse will be different from the one specified in the UpdateRelationshipRequest.

The request identifies the HM configuration and hierarchy, the relationship type, the records, and a number of optional parameters. The response contains the record key for the updated relationship. Informatica MDM Hub infers the types of the entities being related, and thus the base objects containing those entities, from the relationship type.

Adding a New Relationship for a Foreign Key Relationship Type

Use UpdateRelationshipRequest instead of AddRelationshipRequest to add a new Relationship for a Foreign Key Relationship Type because adding that Relationship really involves updating an existing record in the FK Relationship Base Object. For example, if there is a FK Relationship Base Object C_PERSON with the following columns:

- Rowid Object: Primary Key for the FK Relationship and the Entity.
- Rowid Company: FK that refers to the records in C_COMPANY Entity Base Object. The value in this column has a non-null when there a HM FK Relationship between C_PERSON and C_COMPANY. The column value is set to null when the Relationship between a Company and a Person is deleted.
- Any other columns needed for "Person" Entity Type and "Person To Company" FK Relationship Type.

The Put Package associated with the "Person To Company" FK Relationship Type would have the following columns:

- Rowid Object (C_PERSON.Rowid_Object): Primary Key for the FK Relationship
- Rowid BO1 (C_PERSON.Rowid_Object): BO1 in the FK Relationship
- Rowid BO2 (C_PERSON.Rowid_Company): BO2 in the FK Relationship

In other words either rowid BO1 or rowid BO2 in the Put Package maps to the Rowid Object column in the Base Object. Updating the FK column (Rowid BO2 / C_PERSON.Rowid_Company in the example) to a non-null value is equivalent to adding a new FK Relationship. Also note that the RecordKey specified in setRecordKey() and setBo1RecordKey() in this example would be the same.

Note: UpdateRelationshipRequest cannot be used to modify Relationship Type if the old or the new Relationship Type is a FK Relationship Type. To do that, use DeleteRelationshipRequest to delete the old Relationship. Then use UpdateRelationshipRequest if the new Relationship is a FK Relationship Type or AddRelationshipRequest if the new Relationship is not a FK Relationship Type.

Use Case

This is the common scenario for using the UpdateRelationship request:

- **Update a relationship between two HM entities** — If you have Hierarchy Manager and have populated it with entities, you can use the UpdateRelationship request to modify a relationship between two entities.

Related SIF Requests

"AddRelationship" on page 57, "DeleteRelationship" on page 71

UpdateTask

Use the **UpdateTask** API to do one of the following:

- Reassign a task.
- Change the task data.
- Append to the task comment.

TaskData

The `TaskData` object contains information about a task.

The following table lists the TaskData fields that you can configure:

Field	Description
TaskRecord	A link to a data record associated with a task.
Comment	An optional task comment.
TaskType	The task type.
SubjectAreaUID	The UID of the task subject area.
Title	The task title.
TaskID	The ROWID of the task. Cannot be set by user.
DueDate	The date when the task is due.
Priority	The priority of the task. `1`: High priority. `0`: Normal priority. The default is `0`. `-1`: Low priority.
StatusEnum	The workflow status. The default is `TaskStatusEnum.OPEN`.
OwnerUID	The user or role ID to whom the task is assigned.
InteractionID	The Interaction ID.

Field	Description
WorkflowProcessID	The ID of the workflow process that contains the task. Cannot be set by user.
CreateDate	The date when the task was created. Cannot be set by user.
Creator	The name of the user who created the task. Cannot be set by user.
LastUpdateDate	The date when the task was updated. Cannot be set by user.
LastUpdatedBy	The name of the user who updated the task. Cannot be set by user.
PreviousOwner	The name of the user or role to whom the task was previously assigned. The value is Null if the task is new or has not been assigned. Cannot be set by user.

TaskRecord

The `TaskRecord` object contains information about a record.

The following table describes the TaskRecord fields:

Field	Description
SiperianObjectUID	An identifier for an object in Informatica MDM Hub.
RecordKey	An identifier for a record in Informatica MDM Hub.
MatchRuleUID	An identifier for a match rule in Informatica MDM Hub. Only merge tasks require a MatchRuleUID.

Required Request Parameters

The following table describes the required **UpdateTask** request parameters:

Parameter	Description
TaskData	Specifies the task to update.

Optional Request Parameters

The **UpdateTask** API does not have any optional request parameters.

Response Fields

The following table describes the fields the **UpdateTask** response returns:

Parameter	Description
Message	Contains a message indicating if the **UpdateTask** request was processed successfully.
InteractionID	The interaction ID.

Use Cases

The following scenario is a common use case for using the **UpdateTask** request:

- Change existing task data.

Usage Example

The following example updates an existing task:

```
UpdateTaskRequest request = new UpdateTaskRequest();
TaskData task = new TaskData();
request.setTaskData(task);
task.setTaskId("1234");
task.setTitle("Research and resolve item");
task.setComment("This task has been updated.");
task.setDueDate(new Date());
task.setSubjectAreaUid("SUBJECT_AREA.test|Person");
task.setTaskType("ReviewNoApprove");
UpdateTaskResponse response = (UpdateTaskResponse) sipClient.process(request);
```

ValidateChangeList

ValidateChangeList validates a change list against the current ORS. It applies the specified change list to the current repository, executing all of the changes in simulation mode without modifying the ORS, and then returns any errors.

Required Parameters

The following table describes the required parameters:

Parameter	Description
ChangeListXml	Contains the XML string representing the change list to validate.

Optional Parameters

The following table describes the optional parameters:

Parameter	Description
OwnerPassword	Contains the owner password. The default is `""`.
TransactionAttributeType	If set to `NOT_SUPPORTED`, the request does not support a transactional context. If set to `REQUIRED`, the request does requires a transactional context. If set to `REQUIRES_NEW`, the request requires a new transactional context. If set to `SUPPORTS`, the request supports but does not require a transactional context.
ValidateDataIntegrity	If set to `true`, data integrity validation is required. If set to `false`, data integrity validation is not required. The default is `false`.

Response Field

The following table describes the response fields:

Field	Description
Messages	Contains an array of error messages.
Success	If `true`, the change list executed without errors. If `false`, the change list executed with errors.

ValidateMetadata

The **ValidateMetadata** API validates the metadata for the current ORS and returns a list of issues. **ValidateMetadata** only returns a message if metadata issues are found. You must iterate through the list of messages to determine:

- If the **ValidateMetadata** request ran without exceptions.
- If there are any metadata issues.

Required Parameters

The **ValidateMetadata** request does not have required parameters.

Optional Parameters

The following table describes the optional parameters:

Parameter	Description
Checks	Contains the Checklet IDs that specify which validation checks to perform.
TransactionAttributeType	If set to `NOT_SUPPORTED`, the request does not support a transactional context. If set to `REQUIRED`, the request does requires a transactional context. If set to `REQUIRES_NEW`, the request requires a new transactional context. If set to `SUPPORTS`, the request supports but does not require a transactional context.

Response Fields

The following table describes the response fields:

Parameter	Description
Message	Contains a message indicating if the **ValidateMetadata** request was processed successfully.
ErrorID	Contains an Error ID if any errors are returned.
Level	Contains the error level if any errors are returned. The possible error levels are: - FATAL ERROR - ERROR - WARNING - INFORMATIONAL
Text	Contains the error message if any errors are returned.

ValidateTasks

The **ValidateTasks** API checks each merge task specified in the request to verify there is a match table record. The **ValidateTasks** API can also validate external workflow engine merge tasks in addition to Hub merge tasks.

TaskData

The `TaskData` object contains information about a task.

The following table lists the TaskData fields that you can configure:

Field	Description
TaskRecord	A link to a data record associated with a task.
Comment	An optional task comment.
TaskType	The task type.
SubjectAreaUID	The UID of the task subject area.
Title	The task title.
TaskID	The ROWID of the task. Cannot be set by user.
DueDate	The date when the task is due.
Priority	The priority of the task. `1`: High priority. `0`: Normal priority. The default is `0`. `-1`: Low priority.
StatusEnum	The workflow status. The default is `TaskStatusEnum.OPEN`.
OwnerUID	The user or role ID to whom the task is assigned.
InteractionID	The Interaction ID.
WorkflowProcessID	The ID of the workflow process that contains the task. Cannot be set by user.
CreateDate	The date when the task was created. Cannot be set by user.
Creator	The name of the user who created the task. Cannot be set by user.
LastUpdateDate	The date when the task was updated. Cannot be set by user.
LastUpdatedBy	The name of the user who updated the task. Cannot be set by user.
PreviousOwner	The name of the user or role to whom the task was previously assigned. The value is Null if the task is new or has not been assigned. Cannot be set by user.

TaskRecord

The `TaskRecord` object contains information about a record.

The following table describes the TaskRecord fields:

Field	Description
SiperianObjectUID	An identifier for an object in Informatica MDM Hub.
RecordKey	An identifier for a record in Informatica MDM Hub.
MatchRuleUID	An identifier for a match rule in Informatica MDM Hub. Only merge tasks require a MatchRuleUID.

Required Request Parameters

The following table describes the required **ValidateTasks** request parameters:

Parameter	Description
TaskData	Specifies the task to validate.

Optional Request Parameters

The **ValidateTasks** API does not have any optional request parameters.

Response Fields

The **ValidateTasks** response returns the TaskValidationResult which contains the following information for each task specified in the request:

Parameter	Description
TaskID	This parameter identifies the task.
isValid	If `true`, the task is valid. If `false`, the task is not valid.
errorCode	Contains an error code, if any errors are returned.
errorMessage	Contains an error message, if any errors are returned.

Use Cases

The following scenario is a common use case for using the **ValidateTasks** request:

- Validate a merge task to ensure there is a match table record.

Usage Example

The code in the following example validates a task:

```
ValidateTasksRequest request = new ValidateTasksRequest();
TaskMetaData task = new TaskMetaData();
task.setTaskId("1234");
task.setTitle("Research and resolve item");
task.setDueDate(new Date());
task.setSubjectAreaUid("SUBJECT_AREA.test|Person");
task.setTaskType("Merge");
ArrayList tasks = new ArrayList();
tasks.add(task);
request.setTasks(tasks);
ValidateTasksResponse response = (ValidateTasksResponse) sipClient.process(request);
```

CHAPTER 11

Troubleshooting

APPENDIX A

Identifiers

This appendix includes the following topics:

- List of Identifiers, 150
- SiperianObjectUID, 150
- RecordKey, 154

List of Identifiers

The SIF SDK uses the following identifiers to identify the MDM Hub resources:

- SiperianObjectUID
- RecordKey

SiperianObjectUID

SiperianObjectUID is a string identifier that the MDM Hub uses to identify an object.

Use the following syntax to specify a SiperianObjectUID:

```
<Object Type>.<Object Name>|<Child Object Name>
```

The object type in SiperianObjectUID varies based on the SIF request. For example, a `CleanseRequest` object uses the CLEANSE_FUNCTION object type.

The following table lists the object types that you can use in a SiperianObjectUID:

Object Type	Syntax	Example
AUDIT_TABLE	AUDIT_TABLE.<Audit Table Name>	-
BASE_OBJECT	BASE_OBJECT.<Base Object Table Name>	BASE_OBJECT.C_CONSUMER
BATCH_GROUP	BATCH_GROUP.<Job Group Name>	BATCH_GROUP.SAM Batch Group
CASCADE_UNMERGE	-	-

Object Type	Syntax	Example
CLEANSE_FUNCTION	CLEANSE_FUNCTION.<Cleanse Function Library Name>|<Cleanse Function Name>	CLEANSE_FUNCTION.Data Conversion|Format Boolean
CLEANSE_LIBRARY	CLEANSE_LIBRARY.<Cleanse Function Library Name>	CLEANSE_LIBRARY.Data Conversion
CO_CHILD_MANY	-	-
CO_CHILD_ONE	-	-
CO_CONFIGURATION	-	-
CO_FIELD	-	-
CO_LOOKUP_OBJECT	-	-
CO_ORS_SCHEMA	-	-
CO_REFERENCE_MANY	-	-
CO_REFERENCE_ONE	-	-
CO_ROOT_OBJECT	-	-
COLUMN	COLUMN.<Base Object Table Name>|<Column Name>	COLUMN.C_CONSUMER|DOB
CS_CONFIGURATION	-	-
CS_EXCEPTION_HANDLER	-	-
CS_EXECUTE	-	-
CS_INPUT	-	-
CS_ORS_SCHEMA	-	-
CS_OUTPUT	-	-
CS_PARAMETER	-	-
CS_PARAMETER_DEFINITION	-	-
CS_SERVICE	-	-
CS_STEP	-	-
CUSTOM_RESOURCE	-	-
DATA_SECURITY_FILTER	-	-
DISTINCT_SYSTEM	-	-

Object Type	Syntax	Example
GROUP	-	-
HISTORY	HISTORY.<Base Object Table Name>	HISTORY.C_CONSUMER
HM_BLOB	-	-
HM_CONFIGURATION	HM_CONFIGURATION.<HM Profile Name>|<HM Sandbox Name>	HM_CONFIGURATION.Default|Master
HM_ENTITY_OBJECT	-	-
HM_ENTITY_TYPE	-	-
HM_HIERARCHY	HM_HIERARCHY.<HM Hierarchy Code Name>	HM_HIERARCHY.Default
HM_PACKAGE	-	-
HM_PACKAGE_COLUMN	-	-
HM_PROFILE	HM_PROFILE.<HM Profile Name>	HM_PROFILE.Default
HM_RELATIONSHIP_OBJECT	-	-
HM_RELATIONSHIP_TYPE	HM_RELATIONSHIP_TYPE.<HM Relationship Type>	HM_RELATIONSHIP_TYPE.Org To Person
HM_SANDBOX	HM_SANDBOX.<HM Sandbox Name>	HM_SANDBOX.Master
IMMUTABLE_SYSTEM	-	-
INDEX	-	-
LANDING_TABLE	-	-
MAPPING	MAPPING.<Map Name>	MAPPING.Stage CRM Customer
MATCH_COLUMN	MATCH_COLUMN.<Base Object Table Name>|<Match Column Name>	MATCH_COLUMN.C_CUSTOMER|Person_Name
MATCH_KEY	-	-
MATCH_PATH_COMPONENT	MATCH_PATH_COMPONENT.<Match Path Component Name>	MATCH_PATH_COMPONENT.C_MT_MATCH_1_CHILD
MATCH_PATH_COMPONENT_FILTER	-	-
MATCH_POPULATION	-	-
MATCH_RULE	MATCH_RULE.<Base Object Table Name>|<Match Rule Set Name>|<Rule Number>	MATCH_RULE.C_CUSTOMER|Main|1

Object Type	Syntax	Example
MATCH_RULE_SET	MATCH_RULE_SET.<Base Object Table Name>|<Match Rule Set Name>	MATCH_RULE_SET.C_CUSTOMER|Main
MERGE_HISTORY	-	-
MESSAGE_QUEUE_RULE	-	-
METADATA	-	-
METADATA_MANAGER	-	-
METSYSTEM	-	-
OTHER_TABLE	-	-
PACKAGE	PACKAGE.<Package or View Object Name>	PACKAGE.CUSTOMER_UPDATE
PRIMARY_KEY_INDEX	-	-
PRIMARY_KEY_MATCH_RULE	-	
QUERY	QUERY.<Query Group Name>|<Query Name>	QUERY.Read-Only Queries|Constant And Function Test
QUERY_GROUP	QUERY_GROUP.<Query Group Name>	QUERY_GROUP.Read-Only Queries
RAW	RAW.<Base Object Table Name>	RAW.C_CONSUMER
RELATIONSHIP	RELATIONSHIP.<Reference by Base Object Table Name>(<Column Name>).<Reference to Base Object Table Name>(<Column Name>)	RELATIONSHIP.C_ADDRESS(CONSUMER_FKEY).C_CONSUM
REMOTE_PACKAGE	REMOTE_PACKAGE.<Remote View Name>	REMOTE_PACKAGE.CUSTOMER_VIEW
REPOSITORY_SETTINGS	-	-
RESOURCE	-	-
ROLE	ROLE.<Role Name>	ROLE.B_READ
SEQUENCE	-	-
STAGING_TABLE	-	-
SUBJECT_AREA	-	-
SYSTEM	SYSTEM.<System Name>	SYSTEM.CRM
SYSTEM_COLUMN_TRUST	-	-

Object Type	Syntax	Example
TABLE	-	-
TASK_ASSIGNMENT_CONF	-	-
TASK_TYPE	-	-
UNIQUE_KEY_INDEX	-	-
USER	USER.<User Name>	USER.p_b_read_user
VALIDATION_RULE	-	-
WORKFLOW_ENGINE	-	-
XREF	XREF.<Base Object Table Name>	XREF.C_CONSUMER
XREF_HISTORY	XREF_HISTORY.<Base Object Table Name>	XREF_HISTORY.C_CONSUMER

RecordKey

A RecordKey uniquely identifies a record in the MDM Hub. A record is a collection of fields that include a list of names and values. Each name in the record must be unique.

You can use a combination of the following parameters to identify a record:

- rowid. The ROWID_OBJECT column value of the record.
- systemName. Name of the system to which the record belongs.
- sourceKey. The PKEY_SRC_OBJECT column value of the record.
- GBID. Global Business Identifier of an object. You can use one or more GBIDs.

The following sample code uses the `PutRequest` object to insert a record based on the `recordKey` identifier:

```
…
RecordKey recordKey = new RecordKey();
   recordKey.setSystemName( systemName );
recordKey.setSourceKey( sourceKey );
    …
putRequest.setRecordKey(recordKey );
…
```

APPENDIX B

Frequently Asked Questions

How can I perform exact matches on fuzzy base objects?

To perform exact matches on fuzzy base objects, manually add the following property to the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxcleanse.properties` file:

```
cmx.server.match.exact_match_fuzzy_bo_api=1
```

Note: Restart the application server after setting the `cmx.server.match.exact_match_fuzzy_bo_api` property.

Where can I find the debug log file for SIF requests?

You can find the `cmxserver.log` file in the following directory:

`<MDM Hub Installation Directory>\hub\server\log`

How can I change the processing time periods of SIF search requests?

You can add the following properties to the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxserver.properties` file to change the processing time periods of the SIF search requests:

Properties	Description	Default
sif.search.result.refresh.interval.seconds	Specifies the time interval to run the cleanup process for cached search requests. The cleanup process also cleans the temporary tables that the unmerge process creates.	1 second
sif.search.result.query.timeToLive.seconds	Specifies the number of seconds for an unused search request to remain cached. After the specified time period, the cleanup process removes the cached search requests.	900 seconds
sif.search.result.drop.batch.record.count	Specifies the number of cached search requests to process. The number of searches that you specify are fetched until all the expired searches are processed.	200 searches

Properties	Description	Default
sif.search.result.drop.batch.interval.milliseconds	Specifies the number of milliseconds to wait after processing each batch of search results. Use the `sif.search.result.drop.batch.interval.millisecond` property to insert a delay after processing each batch of search results.	0 milliseconds
cmx.server.match.max_time_searcher	Specifies the maximum time period for a search request to run. If a search request does not complete within the specified time period, the search stops.	99999999 seconds

How to optimize the performance of a SearchMatch request?

- Multi-threaded range processing:
  - Use multiple parallel threads to process the search ranges. By default, multi-threaded range processing is disabled. Set the following property to `true` in the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxcleanse.properties` file:

    ```
    cmx.server.match.searcher.database.worker.multithreaded
    ```
  - Change the number of threads to use to process a SearchMatch request. Set the following property in the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxcleanse.properties` file:

    ```
    cmx.server.match.searcher_thread_count
    ```

    Set to 1 to use one thread for the SearchMatch API. Restart the application server after you set the `cmx.server.match.searcher_thread_count` property.
- Fine-tuning the DBFILTERED feature:
  - By default, DBFILTERED is invoked when the SearchMatch record has only one SSA_KEY for fuzzy match key column. Set the following DBFILTERED threshold property in the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxcleanse.properties` file:

    ```
    cmx.server.match.searcher.dbfiltered.max.key.size
    ```

    The DBFILTERED feature is invoked when the SearchMatch record has a SSA_KEY that is less than or equal to the value of the `cmx.server.match.searcher.dbfiltered.max.key.size` property.
  - Specify the resultset size in number of rows: Set the following property to an optimal value in the `<MDM Hub Installation Directory>\hub\cleanse\resources\cmxcleanse.properties` file:

    ```
    cmx.server.match.searcher.resultset.size
    ```
- Routinely determine if the match key tables that are associated with base objects need to be reorganized. Reorganize match key tables based on their primary key column, SSA_KEY.

INDEX

A

AcceptUnmatchedRecordsAsUnique request 56
access protocols
  using SIF 12
addRelationship operation 57
AddRelationship request 57
APIs
  RegisterCustomIndex 123
  RegisterCustomTableObject 124
  RemoveMatchedRecord 126
ApplyChangeList
  rollbackStrategy field 57
applyChangeList call 57
ApplyChangeList request 57
AssignUnmergedRecords request 58
asynchronous requests
  making 40
Audit request 59
Authenticate request 60

B

batch APIs
  ExecuteAutoMatchAndMerge 72
  ExecuteBatchAutomerge 73
  ExecuteBatchBVTSnapshot 74
  ExecuteBatchDelete 74
  ExecuteBatchExternalMatch 75
  ExecuteBatchGenerateMatchTokens 76
  ExecuteBatchKeyMatch 77
  ExecuteBatchLoad 78
  ExecuteBatchMatch 79
  ExecuteBatchMatchAnalyze 80
  ExecuteBatchPromote 80
  ExecuteBatchRecalculateBo 81
  ExecuteBatchRecalculateBvt 82
  ExecuteBatchResetMatchTable 82
  ExecuteBatchRevalidate 83
  ExecuteBatchStage 84
  ExecuteBatchSynchronize 84
  ExecuteBatchUnmerge 85
  ExecuteBatchValidateFKRelationships 86
Batch Group APIs, about 52
build_war macro 30
business entity services
  about 23

C

CanUnmergeRecords request 60
cascade unmerge
  Unmerge 139
Cleanse 31
cleanse request 61
CleansePut
  state management 62
cleansePut request 32, 62
CleanTable
  about 60
ClearAssignedUnmergedRecords request 66
consolidation
  indicator 136
  state 136
content metadata
  DELETED_XREF 88
  HISTORY 88
  PENDING_XREF 88
  RAW 88
  XREF 88
  XREF_HISTORY 88
createChangeList call 66
CreateChangeList request 66
CreateTask 67

D

Data APIs, about 52
Data Retrieval APIs, about 52
Data Steward APIs, about 52
Data Update / Insert APIs, about 52
Delete
  state management 69
DELETED_XREF
  content metadata 88
DeleteRelationship request 71
Deleterequest 69
DescribeSiperianObject request 71

E

Eclipse 13
ExecuteBatchAutoMatchAndMerge
  about 72
ExecuteBatchAutomerge
  about 73
ExecuteBatchBVTSnapshot
  about 74
ExecuteBatchDelete
  about 74
ExecuteBatchExternalMatch
  about 75
ExecuteBatchGroup request 77
ExecuteBatchKeyMatch
  about 77
ExecuteBatchLoad
  about 78

ExecuteBatchMatch
about 79
ExecuteBatchMatchAnalyze
about 80
ExecuteBatchPromote
about 80
ExecuteBatchRecalculateBo
about 81
ExecuteBatchRecalculateBvt
about 82
ExecuteBatchResetMatchTable
about 82
ExecuteBatchRevalidate
about 83
ExecuteBatchStage
about 84
ExecuteBatchSynchronize 84
ExecuteBatchUnmerge
about 85
ExecuteBatchValidateFKRelationships
about 86
ExecuteGenerateMatchTokens
about 76

F

FlagForAutomerge 86
foreign key relationship type
adding new relationship, UpdateRelationship 142

G

GenerateConstraints
about 87
Get 33
Get request 88
GetAggregatePeriod request 90
GetAssignableUsersForTasks 92
GetAssignedRecords request 93
GetBatchGroupStatus request 93
GetBvt
state management 94
GetBvt request 94
GetEffectivePeriods API 95
GetEntityGraph request 96
GetLookupValue request 97
GetLookupValues request 98
GetMatchedRecords
state management 98
GetMatchedRecords request 98
GetMergeHistory request 99
GetOneHop request 99
GetOrsList request 100
getOrsMetadata call 101
GetOrsMetadata request 101
GetSearchResults request 101
GetSiperianObjectCompatibility request 103
getSystemTrustSettings request 103
GetTaskLineage 104
GetTasks request 107
GetTrustGraphData request 109
GetTrustScore request 109
GetUnmergedRecordCount request 110
GetXrefForEffectiveDate
optional parameters 110
required parameters 110
GetXrefForEffectiveDate *(continued)*
usage example 110
use case 110
GetXrefForEffectiveDate API 110
GetXrefForEffectiveDate request 110

H

HISTORY
content metadata 88

I

index.html 16

J

JAR files
ORS-Specific, downloading 30
ORS-Specific, using 30
Javadoc
about 16
JMS event message schema
elements 43
JMS Event Messages
about 42

L

linear unmerge
Unmerge 139
Link request 112
ListSiperianObjects request 112
Load Process vs. SIF Put
validation rules 118

M

Merge
state management 114
Merge request 114
Merge Workflow APIs, about 52
Metadata APIs, about 52
Metadata Management APIs, about 52
Miscellaneous APIs, about 52
MultiMerge request 114

O

ORS-specific API
properties 28
ORS-Specific API
API archive table
maintenance 31
archive table 31
build_war macro 30
downloading client JAR 30
SIF parameters 37
using 30
ORS-specific APIs
repository object statuses 29

ORS-specific APIs
repository objects 28
using 27
ORS-Specific SIF API
classes 31
generating and deploying 29

P

paging support
SearchRequestBase 135
PENDING_XREF
content metadata 88
PreviewBVT
optional parameters 115
required parameters 115
usage example 115
use case 115
PreviewBVT API 115
PreviewBVT request 115
PromotePendingXref request 116
PromotePendingXrefs
state management 116
proxies 13
Put
state management 118
transaction support 118
Put request 118

R

RAW
content metadata 88
ReassignRecords request 123
RecordKey
about 154
RegisterCustomIndex
about 123
RegisterCustomTableObject
about 124
RegisterUsers
transaction support 125
RegisterUsers request 125
RemoveMatchedRecords
about 126
Repository Manager APIs, about 52
ResetBatchGroup request 126
Restore request 127
rollbackStrategy field (ApplyChangeList) 57

S

SearchHmQuery request 128
SearchLookupValues request 128
SearchMatch request 128
SearchMatchColumn 35
SearchMatchRecord 35
searchQuery
case sensitivity 133
retrieving large record sets 133
SearchQuery 36
searchQuery request 133
SearchRequestBase
paging support 135
SearchRequestBase request 135
SearchResponseBase request 136
Security Access Manager workbench
permissions 46
using with SIF 45
Services Integration Framework (SIF) 10
SetPassword request 136
SetRecordState request 136
SIF
access protocols 12
asynchronous requests, making 40
Javadoc, about 16
using Security Access Manager workbench 45
SIF API
ORS-Specific, removing 30
ORS-Specific, renaming 30
SIF calls
applyChangeList 57
createChangeList 66
getOrsMetadata 101
SIP_HOME environment variable 15
siperian-sifsdk.zip, about 15
siperian.sif.jms.queue
about 40
SiperianObjectUID
about 150
SOAP protocol 13
state management
CleansePut 62
Delete 69
GetBvt 94
GetMatchedRecords 98
Merge 114
PromotePendingXrefs 116
Put 118
State Management APIs, about 52

T

Task APIs, about 52
Tokenize request 138
transaction support
Put 118
RegisterUsers 125
UnregisterUsers 141
transactions
using 23
tree unmerge
Unmerge 139

U

Unlink request 139
Unmerge
cascade unmerge 139
linear unmerge 139
tree unmerge 139
XREF added directly to BO 139
Unmerge request 139
UnregisterUsers
transaction support 141
UnregisterUsers request 141
UpdateMatchRecord 141
UpdateMatchRecord request 141
UpdateRelationship
adding new relationship for foreign key type 142
UpdateRelationship request 142

UpdateTask 143
User Management APIs, about 52

V

ValidateChangeList request 145
ValidateMetadata 146
ValidateTasks 147
validation rules
  Load Process vs. SIF Put 118

W

Web Services Description Language (WSDL), about 13
web services, about 13

X

XML over HTTP
  using 14
XREF
  content metadata 88
XREF_HISTORY
  content metadata 88