

Oracle® Financial Services Interconnect File Data Synchronization Integration Guide



Innovation Release 14.8.2.0.0
G52967-02
April 2026

ORACLE®

Copyright © 2026, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1	Domain Integration Guide	
2	Introduction	
3	Integration via Discovery Service	
3.1	Add Connector Dependency	1
3.2	Enable Component Scanning	2
3.3	Configure Interconnect System Parameters	2
3.4	Implement the Interconnect Event Handler Interface	2
3.5	Return an Acknowledgment (Ack) Response	4
4	Integration via Routing Hub (OBRH)	
4.1	Configure Oracle Banking Routing Hub (OBRH)	1
4.2	Configure Interconnect	2
	Index	

Preface

- [Purpose](#)
- [Audience](#)
- [Documentation Accessibility](#)
- [Critical Patches](#)
- [Diversity and Inclusion](#)
- [Conventions](#)

Purpose

This guide outlines the process for product teams to adopt and integrate Interconnects File Services within designated environments.

Audience

This guide is intended for software developers and technical leads on product teams who are responsible for integrating Interconnects File Services into their designated environments

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Critical Patches

Oracle advises customers to get all their security vulnerability information from the Oracle Critical Patch Update Advisory, which is available at [Critical Patches, Security Alerts and Bulletins](#). All critical patches should be applied in a timely manner to make sure effective security, as strongly recommended by [Oracle Software Security Assurance](#).

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Domain Integration Guide

This topic provides information on domain integration.

Interconnect Framework — Integration Reference

Version: Refer to \${RELEASE_VERSION} / \${ARTIFACT_VERSION} in your release pipeline.

2

Introduction

This guide describes how to integrate domain applications with the Interconnect framework using two supported integration patterns:

1. **Integration via Discovery Service** — Library connector-based integration.
2. **Integration via Routing Hub (OBRH)** — API-based integration through Routing Hub.

3

Integration via Discovery Service

This topic describes about the integration with external systems through the Discovery Service.

Overview

The **Connector Service** facilitates seamless communication between the Interconnect framework and domain applications. It abstracts event handling, payload binding, and acknowledgment responses, allowing domain teams to focus on core processing logic without dealing with transport-layer concerns.

- [Add Connector Dependency](#)
This topic describes the connector required to enable integration with dependent services or external systems.
- [Enable Component Scanning](#)
This topic describes how to enable component scanning to automatically detect and register application components.
- [Configure Interconnect System Parameters](#)
This topic describes the system parameters required to configure and enable interconnect functionality.
- [Implement the Interconnect Event Handler Interface](#)
This topic describes how to implement the Interconnect Event Handler interface to handle interconnect events within the system.
- [Return an Acknowledgment \(Ack\) Response](#)
This topic describes how to return an acknowledgment (Ack) response after processing an interconnect event.

3.1 Add Connector Dependency

This topic describes the connector required to enable integration with dependent services or external systems.

Add the following dependency to your build.gradle file:

```
dependencies {  
    implementation("release.obma.cmc.${RELEASE_VERSION}.services:cmc-inc-  
file-connector:${ARTIFACT_VERSION}")  
}
```

Note

Replace `${RELEASE_VERSION}` and `${ARTIFACT_VERSION}` with the appropriate version identifiers aligned to your release pipeline.

3.2 Enable Component Scanning

This topic describes how to enable component scanning to automatically detect and register application components.

The Connector Service components must be discovered by Spring at application startup. Add the connector's base package to your `@ComponentScan` annotation:

```
@ComponentScan({"oracle.fsgbu.cmc.inc.file.connector"})
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Note

If component scanning is not configured, the connector beans — including event listeners and handlers — will not be initialized, and event callbacks will fail silently.

3.3 Configure Interconnect System Parameters

This topic describes the system parameters required to configure and enable interconnect functionality.

Navigate to **System Maintenance** → **Transaction Details** in Interconnect and configure the following parameters:

Table 3-1 System Maintenance_Transaction Details - Field Description

Field	Description
Data Exchange Type	Select Service.
Application Name	The application name as registered in the Discovery Service.
Application Id	Populated in the appld request header.
Implementation Name	Spring Bean name of the class implementing InterconnectEventHandler (camelCase, lowercase first character).
User Id	Populated in the userId request header.
Chunk Size	Determined based on payload volume.

Best Practice: Avoid setting the chunk size excessively high. An optimally calculated chunk size maintains the right balance between throughput and system stability during batch processing.

3.4 Implement the Interconnect Event Handler Interface

This topic describes how to implement the Interconnect Event Handler interface to handle interconnect events within the system.

Each domain application must implement the Interconnect Event Handler<T> interface to handle incoming events from Interconnect.

Interface Definition

```
public interface InterconnectEventHandler<T> {

    /**
     * Returns the target DTO class for automatic Map-to-DTO binding.
     * Enables automatic conversion of the incoming Map payload into
     * a strongly typed DTO using Jackson's ObjectMapper.
     */
    Class<T> payloadType();

    /**
     * Converts the raw Map payload into a domain DTO instance.
     * Override only if custom conversion logic is required.
     */
    default T assembleDto(Map<String, Object> payload, Class<T> dtoClass) {
        return ObjectMapperFactory.getInstance().convertValue(payload,
            dtoClass);
    }

    /**
     * Indicates whether the handler wants asynchronous processing.
     * Currently not supported – must return false.
     */
    default boolean wantsAsync() {
        return false;
    }

    /**
     * Optional pre-processing hook invoked before event processing begins.
     * Useful for input validation or payload enrichment.
     */
    default void onPreProcessEvent(EventContext ctx, T payload, String
        recordId) {
        // no-op by default
    }

    /**
     * Core domain processing method.
     * Must be implemented to handle incoming event payloads.
     * Returns an Ack indicating success or failure.
     */
    Ack onProcessEvent(EventContext ctx, T payload, String recordId);

    /**
     * Asynchronous version of event processing.
     * Not yet supported.
     */
    default void onProcessEventAsync(EventContext ctx, T payload, String
        recordId) {
        // no-op by default
    }

    /**
```

```

        * Optional post-processing hook invoked after event processing completes.
        * Useful for cleanup operations or audit logging.
        */
        default void onPostProcessEvent(EventContext ctx, T payload, String
recordId) {
            // no-op by default
        }
    }
}

```

Example Implementation

```

@Component("customerDataHandler")
public class CustomerDataHandler implements
InterconnectEventHandler<CustomerDto> {

    @Override
    public Class<CustomerDto> payloadType() {
        return CustomerDto.class;
    }

    @Override
    public Ack onProcessEvent(EventContext ctx, CustomerDto payload, String
recordId) {
        try {
            customerService.processCustomerData(payload);
            return Ack.success(recordId);
        } catch (Exception ex) {
            log.error("Failed to process customer data for record {}",
recordId, ex);
            return Ack.failed(recordId, ex.getMessage());
        }
    }
}

```

Note

The Spring bean name (customerDataHandler) must exactly match the **Implementation Name** configured in Interconnect.

3.5 Return an Acknowledgment (Ack) Response

This topic describes how to return an acknowledgment (Ack) response after processing an interconnect event.

Upon completion of domain processing, the handler must return an Ack object to signal the processing outcome to the Connector Service.

Ack Class Definition

```

@Builder
public record Ack(String recordId, Status status, String message) {

    public enum Status { RECEIVED, SUCCESS, FAILED }
}

```

```

    public static Ack success(String recordId) {
        return new Ack(recordId, Status.SUCCESS, "ok");
    }

    public static Ack received(String recordId) {
        return new Ack(recordId, Status.RECEIVED, null);
    }

    public static Ack failed(String recordId, String msg) {
        return new Ack(recordId, Status.FAILED, msg);
    }
}

```

Usage Examples

```

// Acknowledge successful processing
return Ack.success(recordId);

// Acknowledge processing failure
return Ack.failed(recordId, "ERR_VALIDATION_FAILED");

```

Tip: Always use a well-defined, namespaced error code (e.g., `ERR_VALIDATION_FAILED`) instead of a free-text message. This enables locale-aware translation on the consumer side, ensures consistency across domain implementations, and improves traceability during event reconciliation and incident investigation.

Integration Checklist — Discovery Service

Table 3-2 Integration Checklist — Discovery Service

Step	Action
1	Add connector dependency to build.gradle.
2	Enable component scanning for the connector package.
3	Configure system parameters in Interconnect.
4	Implement the Interconnect Event Handler<T> interface.
5	Return appropriate Ack responses from the event handler.

Additional Notes

- Asynchronous processing (`wantsAsync = true`) is **not yet supported**. The method must return `false`.
- DTO binding is performed automatically using Jackson's `ObjectMapper`. Custom conversion logic can be applied by overriding `assembleDto()`.
- Ensure the domain application name is registered correctly in the **Discovery Service** before enabling service-based data exchange.

4

Integration via Routing Hub (OBRH)

This topic describes how to integrate the system using the Routing Hub (OBRH) for message routing and communication between services.

Overview

The **API Integration** capability allows domain applications to interact with Interconnect through the **Oracle Banking Routing Hub (OBRH)**. This integration pattern is well-suited for REST API-driven data exchange, providing routing, transformation, and decoupling between Interconnect and domain systems.

- [Configure Oracle Banking Routing Hub \(OBRH\)](#)
This topic describes how to configure Oracle Banking Routing Hub (OBRH) to enable routing and processing of integration messages.
- [Configure Interconnect](#)
This topic describes how to configure interconnect settings to enable communication between integrated systems.

4.1 Configure Oracle Banking Routing Hub (OBRH)

This topic describes how to configure Oracle Banking Routing Hub (OBRH) to enable routing and processing of integration messages.

Create a Service Consumer

1. Navigate to **Routing Hub** → **Service Consumers**.
2. Click **New Service Consumer** and provide the required consumer metadata (name, description, etc.).
- 3.

Create a Service Provider

1. Within the created Service Consumer, navigate to the **Service Providers** tab.
2. Click **New Service Provider** and configure the following:

Table 4-1 Create a Service Provider

Field	Description
Host & Port	Target domain host and port where APIs will be invoked
Headers	Request headers required by the target domain
Endpoint	API endpoint path that will receive data from Interconnect

Create a Consumer Service

1. Within the same Service Consumer, navigate to the **Consumer Services** tab.
2. Click **New Consumer Service** and configure:

Table 4-2 Create a Consumer Service

Field	Description
Transformation	Data transformation logic for payload conversion or field mapping
Routing	Routing rules for directing requests to the appropriate provider

Once complete, OBRH will be capable of routing and transforming API requests between Interconnect and domain applications.

4.2 Configure Interconnect

This topic describes how to configure interconnect settings to enable communication between integrated systems.

After completing OBRH configuration, link Interconnect to OBRH as follows:

1. Navigate to **Create System** → **System Details** → **Transaction Details** in Interconnect.
2. Set **Data Exchange Method** to API.
3. Provide the following configuration parameters.

Table 4-3 Configuration Parameters

Parameter	Description
Service Consumer	Must exactly match the name configured in OBRH
Consumer Service	Must exactly match the consumer service name within the same Service Consumer in OBRH

4. Select the **API Type**.

Table 4-4 API Type

API Type	Behavior
Single	One REST API call is made per record
Batch	One REST API call is made per chunk (chunk size configurable)

Note

When **Batch** mode is selected, an additional field is displayed to configure the number of records per API call (chunk size).

Integration Flow

Once configured, Interconnect automatically invokes OBRH using the configured Service Consumer and Consumer Service names. OBRH routes the request to the appropriate domain API endpoint based on the service provider and routing configuration. The domain processes the received payload and returns a response in the format expected by Interconnect.

Request & Response Specifications

Request Body Sent by Interconnect

For both Single and Batch modes, Interconnect sends a `Map<String, ObjectNode>` structure where:

- **Key:** `recordId` — Interconnect's internal record identifier
- **Value:** `com.fasterxml.jackson.databind.node.ObjectNode` — the actual JSON payload for the record

Single Mode — Example Request Body

```
{
  "1446467591934476288": {
    "customerName": "Tom",
    "age": "31",
    "id": "14134846241400000"
  }
}
```

Batch Mode — Example Request Body (chunk size = 3):

```
{
  "1446467591934476289": { "customerName": "Jack", "age": "30", "id":
"14134846241400000" },
  "1446467591934476290": { "customerName": "Van", "age": "27", "id":
"14134846241400000" },
  "1446467591934476291": { "customerName": "Sandy", "age": "26", "id":
"14134846241400000" }
}
```

Expected Response from Domain Systems

Interconnect expects domain systems to return a JSON array of per-record processing outcomes:

```
[
  {
    "recordId": "1446467591934476288",
    "status": "SUCCESS",
    "message": "OK"
  },
  {
    "recordId": "1446467591796064256",
    "status": "FAILED",
    "message": "Customer already exists"
  }
]
```

Allowed status values

Table 4-5 Allowed status values

Value	Description
SUCCESS	Record was processed successfully
FAILED	Record processing failed; include a descriptive message

Note

Single Mode: Even in Single API mode, the domain must return a **list containing a single element**.

Recommended: OBRH Response Transformation

It is strongly recommended to leverage OBRH's **response transformation** capability to decouple domain response structures from Interconnect's expected format. This approach eliminates the need for domain-side code changes whenever Interconnect's response contract evolves.

OBRH transformation can handle scenarios such as:

- Mapping domain-specific status codes to SUCCESS or FAILED.
- Extracting error descriptions into the message field.
- Wrapping a single response object into the required list structure.

Integration Checklist — Routing Hub**Table 4-6 Integration Checklist — Routing Hub**

Step	Action
1	Create Service Consumer, Service Provider, and Consumer Service in OBRH.
2	Configure transformation and routing logic in OBRH.
3	Configure Transaction Details in Interconnect.
4	Interconnect invokes OBRH using configured Service Consumer and Consumer Service names.
5	OBRH routes payloads to the target domain API endpoint.

Additional Notes

Ensure naming consistency between OBRH and Interconnect configurations. Any mismatch in Service Consumer or Consumer Service names will result in routing failures.

Index

A

Add Connector Dependency, [1](#)

C

Configure Interconnect, [2](#)

Configure Interconnect System Parameters, [2](#)

Configure Oracle Banking Routing Hub (OBRH), [1](#)

D

Domain Integration Guide, [1](#)

E

Enable Component Scanning, [2](#)

I

Implement the Interconnect Event Handler
Interface, [2](#)

Integration via Discovery Service, [1](#)

Integration via Routing Hub (OBRH), [1](#)

Introduction, [1](#)

R

Return an Acknowledgment (Ack) Response, [4](#)