ORACLE
CLOUD

Using External Data Integration Services for Oracle ERP Cloud, Release 13

ORACLE

Table of Contents

Overview

Business organizations typically have a recurring need for the streamlined management of inbound and outbound data in areas such as initial data conversion, master data creation and maintenance, regular transaction processing, and fiduciary compliance. Oracle ERP Cloud offers a comprehensive collection of tools and feature sets to meet these requirements. Oracle ERP integration scenarios generally involve system-to-system integration flows between distinct on-premise systems, third-party or legacy systems, and Cloud systems.

Inbound Data Management

Oracle ERP Cloud Bulk Data Import Services accommodate:

- High-volume data import scenarios

- Support of legacy data migration, as well as recurring bulk data import

- Automation of end-to-end import flows with web service architecture

- Tracking of import processes for completion, errors, and resubmission

- Notifications in the form of e-mail and callback to automate data validation and error resolution

- Predefined import templates for business objects

Outbound Data Management

Oracle ERP Cloud Bulk Data Export Services deliver:

- Standard prebuilt reports across applications that can be run on demand

- BI Publisher report capabilities that empower users with custom reporting tools

- Efficient data extract formats such as XML, CSV, and TXT

- Automation of end-to-end export flows with web service architecture

- Tracking of export processes for completion, error tracking, and resubmission

- Features which empower businesses with notification such as e-mail and callback to initiate downstream business tasks or operations

Inbound Data Overview

There are several scenarios where data from on-premise or external business systems needs to be imported into Oracle ERP Cloud to consummate business transactions such as:

- Recurring billing transactions originating from on-premise or PaaS-based applications which will be imported into Oracle ERP Cloud.

- Claims generated from on-premise insurance claim processing applications, which require the creation of Payables invoices for remitting payments.

- Journal entries from legacy applications which will be imported into Oracle ERP Cloud.

External data integration services for accommodating inbound data in Oracle ERP Cloud include the following components:

- Templates to structure, format, and generate the data file according to the requirements of the target application objects.

- File-based load process to load the data file(s) into the respective product application interface tables.

- Application-specific data import processes to transfer data from product application interface tables to the relevant product application tables.

The following flow diagram outlines the steps involved in the process:
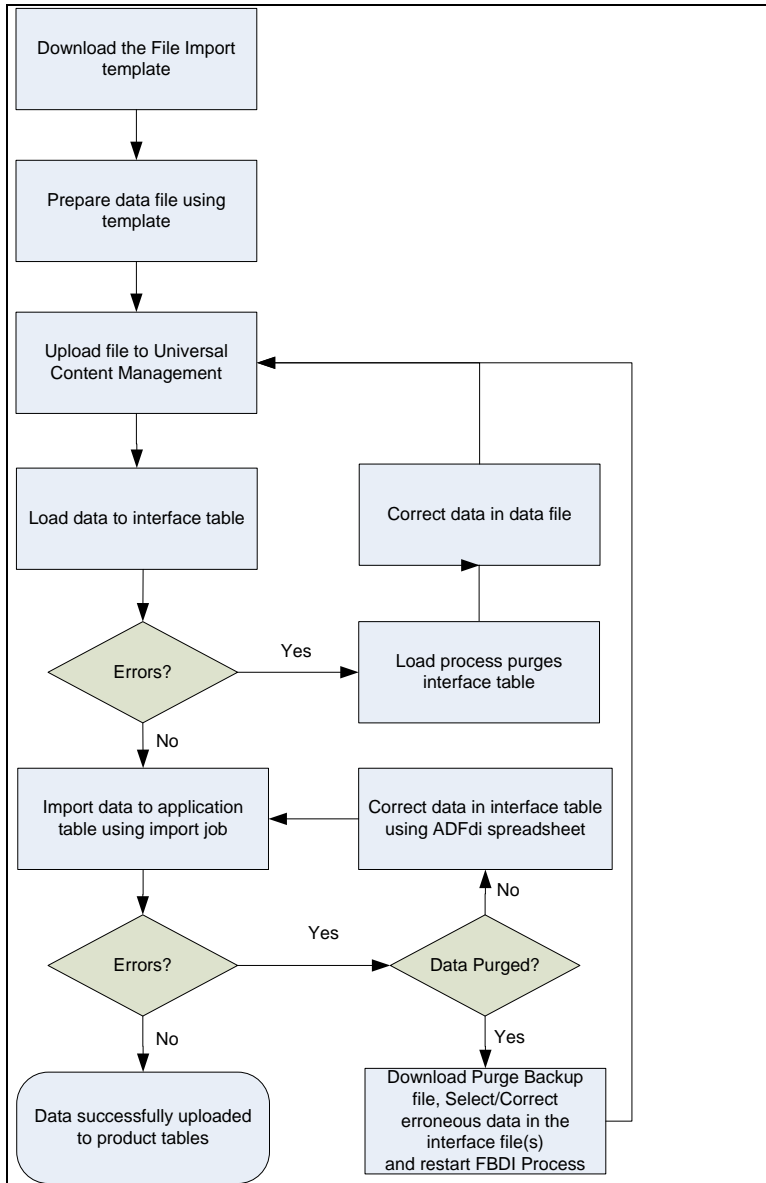
Figure 1: External data integration conceptual process flow

To automatically import data into Oracle ERP Cloud:

1. Create the data file using the applicable inbound business object template.

2. Invoke the Oracle ERP Integration Service to initiate import.

3. Provide notification through asynchronous callback upon completion.

4. Deliver the import status and information using callback to access logs or errors.

5. Review any errors, if applicable, and take appropriate action for error resolution.

The following diagram outlines the steps involved in the automated data import process:
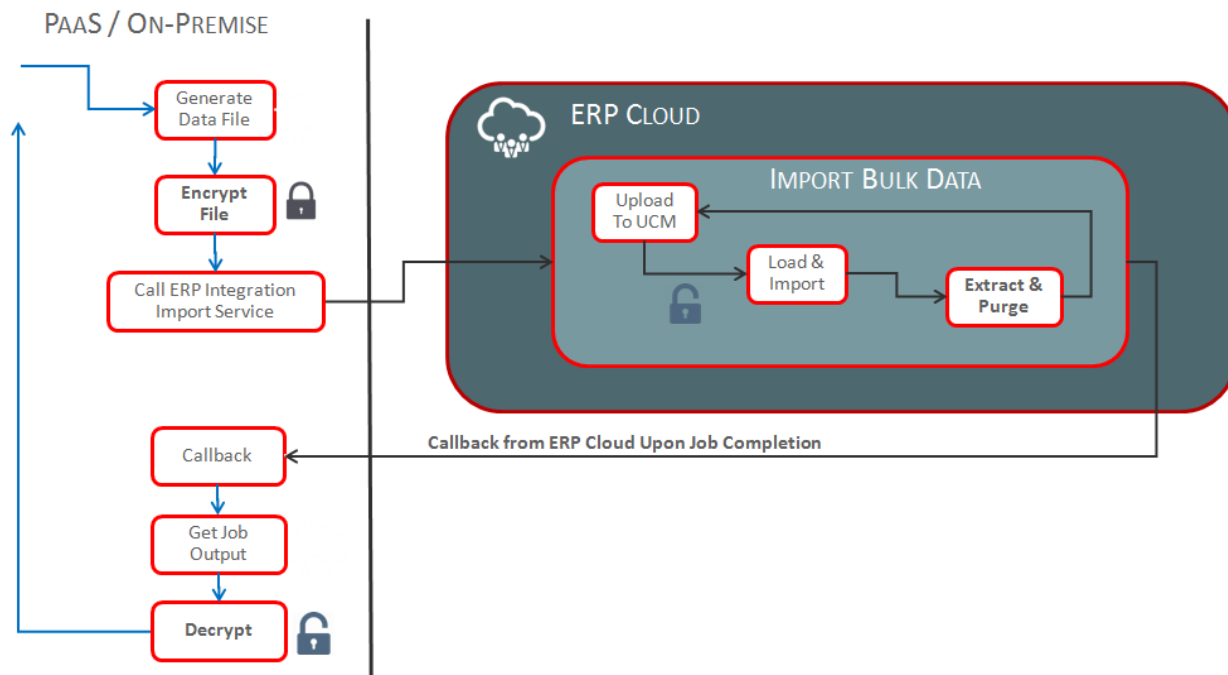
Figure 2: Inbound data integration orchestration flow

> **Note**: After completion, Oracle ERP Cloud extracts data from the interface and error tables, includes the ESS job log files, and uploads the files to the UCM server in a ZIP format. Once uploaded successfully to the respective UCM account, data from the interface and error tables will be purged.

Outbound Data Overview

- Global statutory or fiduciary requirements drive diverse reporting and data extract needs. In these types of business scenarios, the flow of data from Oracle ERP Cloud is utilized for either (1) end-state reporting to internal

business stakeholders, financial institutions, government agencies, tax authorities, or third parties or (2) as an intermediate means to perform additional downstream tasks. The seamless launch of a payables register, trial balance, and reconciliation reports represent some of the examples in practice.

- Automated payment data extract from Oracle ERP Cloud to update downstream external applications.
- Existing master data extracts, such as customers, suppliers, and so on, to synchronize with external applications.

Exporting data from Oracle ERP Cloud typically consists of the following steps:

1. Create a BI Publisher report(s) using the respective Enterprise Scheduler (ESS) job or BI Publisher Dashboard.

2. Invoke the Oracle ERP Integration Service to initiate the respective export job.

3. Provide notification through asynchronous callback upon completion.

4. Deliver the status and information using callback to access extracted data file(s) from the Oracle ERP Cloud.

5. Review any errors if applicable and take appropriate action, such as process the data extracted for further downstream business operation needs.

The following diagram outlines the steps involved in the automated data export process:
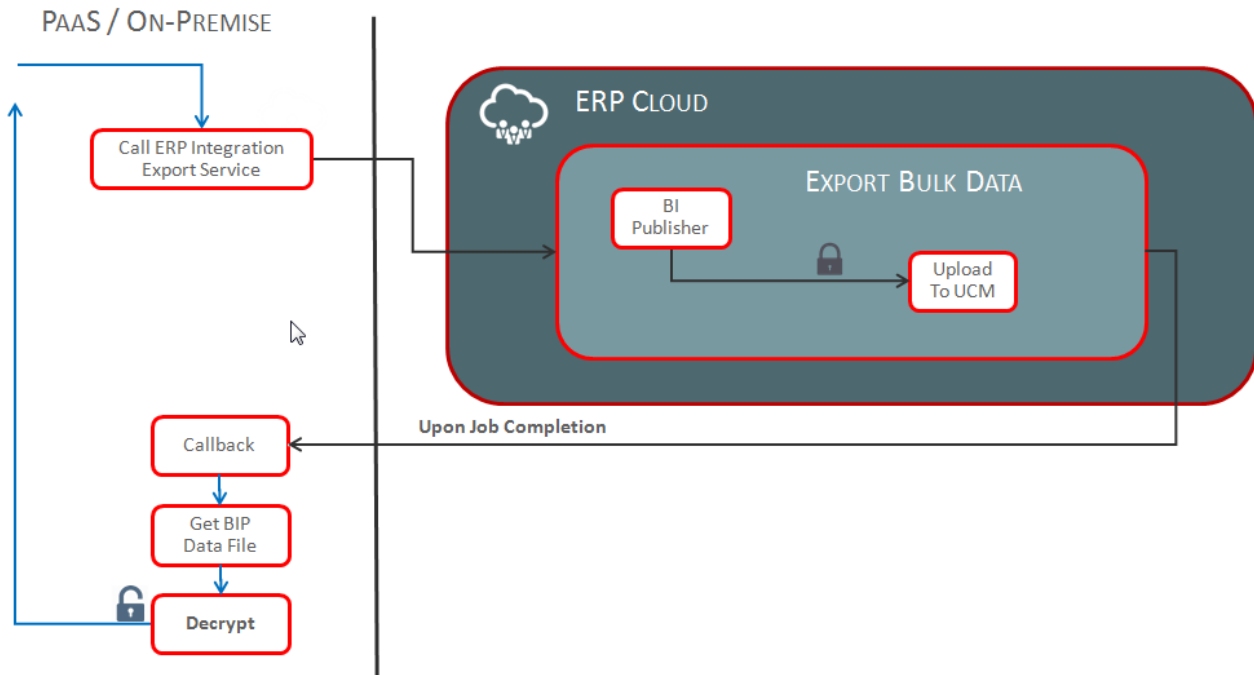
Figure 3: Outbound data integration orchestration flow

What's New

With Oracle ERP Cloud, Release 12, several new features have been added to further simplify and enhance the Oracle ERP Integration Service capabilities:

1. New operations importBulkData and exportBulkData that further simplify bulk data management.

2. Encryption option to secure data files for import and export processes.

3. Capability of efficiently handling large files.

4. Purging of product application interface tables where needed.

5. Multi-threading bulk import process.

Automated End-to-End Inbound (Bulk Import) Orchestrated Flow

To illustrate the inbound data integration with Oracle ERP Cloud, the Journals Import flow will be used as an example.

Prerequisites

- Import a certificate into your local keystore. For more information, see Appendix 5: Testing Web Service using a Client Proxy.

- Configure the sample web service tester. For more information, see Appendix 5: Testing Web Service using a Client Proxy.

- Identify the user name and password to call the Import Journals process.

- Verify that the user has access to the AttachmentsRead role in the UCM server. For more information, see Appendix 1: Security Prerequisites to Download the Job Output File.

- Verify the end point URL for the web service. For more information, see the automation web service at `https://<hostname>.<domainname>/publicFinancialCommonErpIntegration/ErpIntegrationService?WSDL`.

Flow Steps

1. Generate the data file for the object you want to import. For more information, see Generating the Inbound Data File.

2. Prepare the request payload for the ImportBulkData operation of the Oracle ERP Integration Service. This web service operation performs the following tasks:

    a. Uploads the data file to the UCM server.

    b. Loads data from the file on the UCM server to the respective product interface table(s).

    c. Imports the data from the product interface table(s) to the respective Oracle ERP product main table(s).

    d. Extracts errors and logs into a ZIP file and uploads them to the respective UCM account.

    e. Purges the interface and errors tables related to the respective import job.

    f. Notifies users upon completion of all ESS jobs using bell, e-mail, or callback URL as defined in the payload.

3. Receive a bell, e-mail, or callback notification for the request identifier returned by the web service operation in step 2.

4. Prepare the payload for the getDocumentForDocumentId operation to download the output file.

Generating the Inbound Data File

The File-Based Data Import guides in the Oracle Help Center (http://docs.oracle.com) include integration templates to help you prepare external data for loading and importing. Each template includes table-specific instructions, guidelines, formatted spreadsheets, and best practices for preparing the data file for upload. Use the templates to ensure that your data conforms to the structure and format of the target application tables.

When preparing external data using the templates for the purposes of import, the following tasks are required:

- Download the applicable import template

- Prepare data using the correct spreadsheet import template

Downloading a Template

To download a template:

1. Open the File-Based Data Import guide for your cloud service. Locate the import process of interest.

2. View the list of files:

   - Control files describe the logical flow of the data load process.

   - Spreadsheet templates include the worksheets and macros for structuring, formatting, and generating your data file.

     > **Note**
     >
     > You can use XML templates to import data into Oracle Data Integrator. For more information on using XML templates, see Appendix 8: Using XML Templates to Generate Data Files.

3. Click the applicable template URL in the File Links table to download the file. For example, click **JournalImportTemplate.xlsm** in the Journal Import topic.

Preparing Data Using the Spreadsheet Template

To prepare your data in a spreadsheet format:

1. Open the spreadsheet template. The first worksheet in each file provides instructions for using the template.

   > **Important**
   >
   > If you don't follow the instructions, you'll get data load errors and data import failures. If the file is machine generated, you must use UTF-8 encoding to avoid load errors.

2. Enter the required data and then save the file.

3. Click the **Generate CSV File** button.

The macro generates a comma-separated values (CSV) file and also compresses the file into a ZIP file. You must transfer the ZIP file to the Oracle Content Management Server (UCM).

Overview of Template Structure

The integration templates include the following characteristics:

- Each interface table is represented by a separate worksheet.

- Each interface table field is represented by a worksheet column with a header in the first row.

- Each column header contains bubble help text or help comments that include details about the column, such as the expected data type, length, and in some cases, other relevant instruction text.

- Columns are formatted, where applicable, to match the target field data type to eliminate data entry errors.

- The worksheet columns are in the order that the control file processes the data file.

For more information on the template structure, see the main Instructions worksheet in the template.

Template Requirements

To minimize the risks of an unsuccessful data load, ensure the following:

- Unused columns can be hidden, but they cannot be reordered or deleted.

> **Important**
>
> Deleting or reordering columns causes the load process to fail and results in an unsuccessful data load.

- External data must conform to the data types accepted by the control file and process for the associated database column.

- Date column values must appear in the YYYY/MM/DD format.

- Amount column values can't have separators other than a period (.) as the decimal separator.

- Negative values must be preceded by the minus (-) sign.

- Column values that require whole numbers include data validation to allow whole numbers only.

- For columns that require internal ID values, refer to the bubble help text for additional guidance about finding these values.

After you finish preparing the data in the applicable spreadsheet template worksheet(s), click the **Generate CSV File** button on the main Instructions worksheet to generate a ZIP file containing one or more CSV data files.

For more information on using XML templates to generate data files, see Appendix 8: Using XML Templates to Generate Data Files.

Automated End-to-End Outbound (Bulk Export) Orchestrated Flow

To illustrate the outbound data integration with Oracle ERP Cloud, the Extract Receivables Transactions flow will be used as an example.

Example

A batch of transactions is extracted from the application and sent to the customers. The transactions are extracted from the output file of the Print Receivables Transaction ESS process.

Prerequisites

- Import a certificate into your local keystore. For more information, see Appendix 5: Testing Web Service using a Client Proxy.

- Configure the sample web service tester. For more information, see Appendix 5: Testing Web Service using a Client Proxy.

- Identify the user name and password to call the Print Receivables Transaction process.

- Verify that the user has access to the AttachmentsRead role in the UCM server. For more information, see Appendix 1: Security Prerequisites to Download the Job Output File.

- Verify the end point URL for the web service. For more information, see the automation web service at: `https://<hostname>.<domainname>/publicFinancialCommonErpIntegration/ErpIntegrationSe rvice?WSDL.`

Flow Steps

1. Prepare the payload for the exportBulkData operation to request the data extract from Oracle ERP Cloud.
2. Receive a bell, e-mail, or callback notification for the request identifier returned by the web service operation in step 1.
3. Prepare the payload for the getDocumentForDocumentId operation to download the output file.

Flow Automation using the Oracle ERP Integration Web Service

A web service for Oracle Fusion Financials is an artifact that provides a standardized way of integrating other web-based applications or business system processes with Oracle ERP Cloud. Web services allow organizations to communicate with Oracle ERP Cloud without any application expertise. The Oracle ERP Integration Service is an external web service that provides robust web service operations, such as supporting the bulk import of data into the Oracle ERP Cloud, the bulk export of data from the Oracle ERP Cloud, and key value-added operations to retrieve files and purge interface and error data periodically.

Internal Web Service Name: oracle.apps.financials.commonModules.shared.erpIntegrationService.ErpIntegrationService

To access automation details using the Oracle ERP Integration Service, refer to the SOAP Web Services guide for your cloud services in the Oracle Help Center (http://docs.oracle.com).

Constructing the Oracle ERP Integration Service End Point URL

To obtain the physical end point of any specific instance:

1. Launch the ATK home page and sign in as a functional user.

2. Navigate to a dashboard or work area associated with the Payables Service.

3. In the Payables Invoice workbench, you can see a URL in the browser similar to `https://<hostname>.<domainname>/payables/faces/InvoiceWorkbench`.

   The "`<hostname>.<domainname>`" may be "`https://<pod-name>.<lba>.xxx.oraclecloud.com`". In this example "`<pod-name>.<lba>`" is the hostname and "`xxx.oraclecloud.com`" is the domain name.

   a. In this URL, capture "`https://<hostname>.<domainname>`".

   b. Append the static context root: "`/publicFinancialCommonErpIntegration/ErpIntegrationService`". "`https://<hostname>.<domainname>/publicFinancialCommonErpIntegration/ErpIntegrationService`" is the WSDL URL for the Oracle ERP Integration Service.

Critical Web Service Operations to Automate Integration Flows

The Oracle ERP Integration Service includes the following operations:

- importBulkData (Inbound): Imports data into Oracle ERP Cloud.

- exportBulkData (Outbound): Exports data from Oracle ERP Cloud.

- getDocumentForDocumentId: Retrieves data output file(s) from Oracle ERP Cloud.

Operation: importBulkData

The importBulkData operation uploads a file to the Oracle Universal Content Management (UCM) server based on the document details specified and submits an ESS job to load and import the uploaded files to an application table.

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Document	List of elements, each containing the details of the file to be uploaded. The details include the file content, file name, content type, file title, author, security group, and account. **Mandatory Document Attributes:** • **Content**: File content uploaded on to the UCM server. The value of the content tag is obtained by converting the file content into Base64 encoding. For a sample program for Base64 encoding, see Appendix 2: Sample Code for Preparing a Data File for Inbound and Outbound Flow. • **FileName**: Name of the file on the UCM server. **Optional Document Attributes:** • **ContentType:** Type of content uploaded such as zip, txt, or csv. • **DocumentTitle:** Title of the file on the UCM server. • **DocumentAuthor:** Author of the document. • **DocumentSecurityGroup:** A fixed value used to import or export documents on the UCM server. The security group for all the import processes is FAFusionImportExport. • **DocumentAccount:** Account under which the file is uploaded. For more information on the UCM account associated with the ESS process, see Appendix 3: Predefined Target UCM Accounts.	IN	Yes	java.lang.String
Job Details	The details of the ESS job used to import and process the uploaded file. The details include the primary job information (job definition name, job package name), ParameterList, and JobRequestId. To get the job package and definition name, see Viewing Details about Predefined Scheduled Processes: Procedure in the File-Based Data Import for Oracle Financials Cloud guide in the Oracle Help Center at http://docs.oracle.com.	IN	No, if the job property file is provided	java.lang.String
Notification Code	A two-digit number that determines how and when a notification is passed for the status of the import job. See the table below for the notification code values.	IN	Yes	java.lang.String
Callback URL	The callback URL of the web service you implemented to receive the ESS job status upon job completion.	IN	No	java.lang.String
Job Options	Optional parameters, comma separated. To enable data file encryption, you must provide the following options: FileEncryption=PGPUNSIGNED or PGPSIGNED FA_ALIAS=<ERP Cloud Key Alias Name> CUSTOMER_ALIAS=<Customer Key Alias Name> **Example:**	IN	No	java.lang.String

	FileEncryption=PGPUNSIGNED,FA_ALIAS=ERP_CLOUD_KEY,CUSTOMER_ALIAS=CUSTOMER_ERP_KEY			
Response Code	The response code that returns the request identifier for the first import job in the joblist - which is a load interface job.	OUT		java.lang.Long

Note

When a file upload to the UCM server fails, the remaining ESS jobs aren't executed and a response code of zero (0) appears.

The following table provides information on the notification codes:

Digit Position	Digit Value	Meaning
First digit	1	E-mail notification
	2	Bell notification
	3	Email and bell notification
Second digit	0	Send in any case (import failed or succeeded)
	1	Send on import success
	2	Send on import failure

Job Details

The job details include the job definition and package names, as well as the job parameters of the imported object. The following options may be used to specify the Job Details parameter associated with the importBulkData operation:

- Specify the Job Details parameter directly in the request payload

- Add the Job Property file as part of the data ZIP file

- Upload the Job Properties file to the UCM and add JobDetailFileName=<FileName.properties> in jobOptions

See Job Property File for the Bulk Import Process for advanced features on job details.

To get the job package, definition name, and list of parameters, see Viewing Details about Predefined Scheduled Processes: Procedure in the File-Based Data Import for Oracle Financials Cloud guide in the Oracle Help Center at http://docs.oracle.com. The following illustrates how to get the job details for a journal import:

Figure 4: Sample page to get job details

Specify the Job Details Parameter Value as Part of the Request Payload

The following example illustrates the Journal Import process with the parameters included in the request payload:

```
<soap:Body>
   <ns1:importBulkData
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
   <ns1:document
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/">
   <ns2:Content>
UEsDBBQAAAAIAKSUz0guTciD5gAAAKwDAAAPAAAAR2xJbnRlcmZhY2UuY3N2ZLBasMwDIbvg72DHkB1JFkJcW8tyXbZwliz7hxabwSyZCT
dYW8/Z/SWFgZm/8EY+/sl9KOqfEVGIcoTlsQy1n46bZuu6Q8en/3HcPJQj00/vfkRd/dFQANHmpDDDRIjESEz03z5PfCqQpcsNTZHZFFWkw
X27qvr4NwP6u9PDysotzt48Md3P65h307t0MP59cmP7XBcQ+EPKwp1Yu1LOeE8dXLxb1ZZFbc31aXQ4FpqRfkYk1rAXMrinLGBykScqiOjW
fT0kfal/iG8zUvUylmb5mlYupkRtcqkRiV69Ej7Un9K7gdQSwECFAAUAAAACACklM9ILk3Ig+YAAACsAwAADwAAAAAAAAAAABACAAAAAAAAA
R2xJbnRlcmZhY2UuY3N2UEsFBgAAAABAAEAPQAAABMBAAAAAA==</ns2:Content>
   <ns2:FileName>JournalsImportTEST_1234.zip</ns2:FileName>
   </ns1:document>
   <ns1:jobDetails
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/">
   <ns2:JobName>oracle/apps/ess/financials/generalLedger/programs/common,JournalImportLauncher</ns2:JobName
>
   <ns2:ParameterList1061,Payables,1,ALL,N,N,N </ns2:ParameterList>
   </ns1:jobDetails>
   <ns1:notificationCode>30</ns1:notificationCode>
```

```
    <ns1:callbackURL>http://hostname:port/myCallbackService</ns1:callbackURL>

    <ns1:jobOptions></ns1:jobOptions>

    </ns1:importBulkData>

</soap:Body>
```

Figure 5: Sample request payload for the Journals Import process

Sample Response from the importBulkData Operation

The importBulkData operation response contains the Request ID of the job loading data into the respective product interface table.

```
<env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="Body-Xm0FCudjyFrfJWAmQDXCvw22">
  <ns0:importBulkDataResponse
xmlns:ns0="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
      <result
xmlns="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/">252
9</result>
  </ns0:importBulkDataResponse>
</env:Body>
```

Figure 6: Response payload for the importBulkData web service operation

Operation: exportBulkData

The following table lists the parameters for the exportBulkData operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Job Name	Job package name, Job definition name, both comma separated.	IN	Yes	java.lang.String
Parameter List	ESS job parameters of the ESS job, comma separated. If the job does not have parameters, enter #NULL.	IN	Yes	java.lang.String
Notification Code	A two-digit number that determines how and when a notification is passed for the status of the export job. See the table below for the notification code values.	IN	No	java.lang.String
Callback URL	The callback URL of the web service you implemented to receive the ESS job status upon job completion.	IN	No	java.lang.String
Job Options	Optional parameters comma separated. To enable data file encryption, you must provide the following options: FileEncryption=PGPUNSIGNED or PGPSIGNED FA_ALIAS=<ERP Cloud Key Alias Name> CUSTOMER_ALIAS=<Customer Key Alias Name>	IN	No	java.lang.String

	Example: FileEncryption=PGPUNSIGNED,FA_ALIAS=ERP_CLOUD_KEY,CUSTOMER_ALIAS=CUSTOMER_ERP_KEY			
Response Code	The response code that returns the request identifier of the export job.	OUT		java.lang.Long

The following illustration highlights a sample request payload of the exportBulkData operation:

```
<soap:Body>
  <ns1:exportBulkData
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>

<ns1:jobName>oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader,InterfaceLoaderPurge</ns1:jobName>
    <ns1:parameterList>48,1001</ns1:parameterList>
    <ns1:jobOptions></ns1:jobOptions>
    <ns1:callbackURL>30</ns1:callbackURL>
    <ns1:notificationCode>http://hostname:port/myCallbackService</ns1:notificationCode>
  </ns1:exportBulkData>
</soap:Body>>
```

Figure 7: Sample request payload for the exportBulkData operation

Operation: getDocumentForDocumentId

The getDocumentForDocumentId operation downloads the job output file generated by the importBulkData operation or data file extracted by exportBulkData operation. This operation requires application user access and access to the AttachmentsRead role.

For more information on assigning a user with this access, see Appendix 1: Security Prerequisites to Download the Job Output File.

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Document ID	The UCM document ID from the callback response.	IN	Yes	java.lang.String
return	A list of elements, each containing the details of the downloaded files. The details include the document ID, file content, file name, content type, file title, author, security group, and account.	OUT		List<DocumentDetailsVORowImpl>

The following sample request payload illustrates the Journal Import process:

```
<soap:Body>
 <ns1:getDocumentForDocumentId
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
 <ns1:DocumentId>5900</ns1:DocumentId>
 </ns1:getDocumentForDocumentId>
```

```
</soap:Body>
```

Figure 8: Sample request payload for the getDocumentForDocumentId operation

Security Policy of the Oracle ERP Integration Service

The Oracle ERP Integration Service is secured using the following policy:

oracle/wss11_saml_or_username_token_with_message_protection_service_policy

When a client calls the service, the service must satisfy the message protection policy to ensure that the payload is transported with encryption, or sent over the SSL transport layer.

A client policy that can be used to meet this requirement is:

oracle/wss11_username_token_with_message_protection_client_policy

To use this policy, the message must be encrypted using a public key provided by the server. When the message reaches the server, it can be decrypted by the server's private key. A keystore is used to import the certificate, and it is referenced in the subsequent client code.

The public key can be obtained from the certificate provided in the service WSDL file. See the following figure for an example of a certificate that is Base64 encoded.

```
<wsdl:service name="ErpIntegrationService">
<wsdl:port name="ErpIntegrationServiceSoapHttpPort" binding="tns:ErpIntegrationServiceSoapHttp">
<soap:address location="
███████████████████████████████████████/publicFinancialCommonErpIntegration/ErpIntegrationService"/>
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
███████████████████████████████████████publicFinancialCommonErpIntegration/ErpIntegrationService
</wsa:Address>
<wsid:Identity xmlns:wsid="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
<dsig:X509Data>
<dsig:X509Certificate>
```
```
MIIC6DCCAdCgAwIBAgIGAVP2HjwHMA0GCSqGSIb3DQEBCwUAMHgxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdNeVN0YXRlMQ8wDQYDVQQHEwZNeVRvd24xFzA
VBgNVBAoTDk15T3JnYW5pemF0aW9uMRkwFwYDVQQLExBGT1IgVEVTVElORyBPTkxZMRIwEAYDVQQDEwlDZXJ0R2VuQ0EwHhcNMTYwNDA4MTM0NDQ5WhcNMj
EwNDA3MTM0NDQ5WjBTMRMwEQYKCZImiZPyLGQBGRYDY29tMRYwFAYKCZImiZPyLGQBGRYGb3JhY2xlMRIwEAYKCZImiZPyLGQBGRYCdXMxEDAOBgNVBAMTB
3NlcnZpY2UwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANaq50W6Og6lHiTkhY2aJM/EX/BnbQHHt7RAkrQGtTJbK+Jvp2Un91Nf4j84nZyuE3gaxYpn
I4DM5jSOAw7oKCddvHs3XItgszmeCRyZLQGi9bQQuV1t8rK+7zn4Y4IsnNgzJrWmJ7fx1ATVYR7vrSgyQclk1orK+l5wAj9AS1jPAgMBAAGjITAfMB0GA1U
dDgQWBBRH42KiqNY5aBgqzhAXg1gUqJbHyzANBgkqhkiG9w0BAQsFAAOCAQEAHnvQAIDjSW95xa+J/vJmQUoK8W55dQt1ifPiQTh9AHoMTiouVug3k4gpXg
556BvrPKmni+YY+cN33uvsuXkypTh0mQydmlldiu5CZh2TPiDPFcpAOfxgw56sLT5q9PTJZJPwX952E7llTksILTBMECFmF/ChKpoT1yEqjrO+F+0CpsqvV
wIubPgFSqB06EBayGNCeVpoSX68RwOlJs80go2QoNJ0DV6U7vu7C9h/IkuRrQxCFhUSGOZr348CkX7sLtcBhrlIB5hJEb6F5d98kBQxRxcPvNWLE4ClMx5H
nMJnXnIq8oZrigTJNUlRo4iLfL4uYYhiQGZrIZukuQluyQ==
```
```
</dsig:X509Certificate>
<dsig:X509IssuerSerial>
<dsig:X509IssuerName>
CN=CertGenCA, OU=FOR TESTING ONLY, O=MyOrganization, L=MyTown, ST=MyState, C=US
</dsig:X509IssuerName>
<dsig:X509SerialNumber>1460123089927</dsig:X509SerialNumber>
</dsig:X509IssuerSerial>
<dsig:X509SubjectName>CN=service, DC=us, DC=oracle, DC=com</dsig:X509SubjectName>
<dsig:X509SKI>R+GSoqjWOWgaqs4QF4NYFKiWx8s=</dsig:X509SKI>
</dsig:X509Data>
</dsig:KeyInfo>
</wsid:Identity>
</wsa:EndpointReference>
</wsdl:port>
```

Figure 9: Example of a certificate in the Oracle ERP Integration Service WSDL file

To use the key contained in this certificate, create a local Keystore and import the certificate into it. For more information, see Appendix 5: Testing Web Service using a Client Proxy.

Callback Web Service

In practice, customers create and host a callback web service to optimally leverage the callback capabilities provided by the Oracle ERP Integration Service for notification purposes. The customer callback web service must implement the onJobCompletion() operation. When a job completes, Oracle ERP Integration Service invokes the customer callback web service as defined in the request payload of supported operations with callback capabilities, such as the importBulkData operation.

For more information on Callback Service, see Appendix 11: Creating a Callback Web Service.

Callback Response in JSON Format

The callback response provides execution statuses and request IDs of all the applicable ESS jobs. For example, to import AP invoices, the following jobs are executed:

1. Load Interface File for Import

    i. Transfer File (upload file to UCM)

    ii. Load Data to Interface tables

2. Import Invoices

    i. Import Invoices Report

The response includes the UCM document ID of the output artifacts, such as the logs of each ESS jobs, and data from the interface and error tables. This ZIP file can be retrieved using the Operation: getDocumentForDocumentId.

The following callback response provides the Request ID and status of each of the jobs outlined above.

```
{
    "JOBS":
                [
                    {"JOBNAME":"Load Interface File for Import",
                     "JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
                     "DOCUMENTNAME":"apinvoiceimport.zip",
                     "REQUESTID":"2529",
                     "STATUS":"SUCCEEDED",
                     "CHILD":[
                                    {"JOBNAME":"Transfer File",
"JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
                                     "REQUESTID":"2530",
                                     "STATUS":"SUCCEEDED"},
                                    {"JOBNAME":"Load File to Interface",
"JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
                                     "REQUESTID":"2531",
                                     "STATUS":"SUCCEEDED"}
                               ]
                    },
                    {"JOBNAME":"Import Invoices",
                     "JOBPATH":"/oracle/apps/ess/financials/payables/invoices/transactions",
                     "REQUESTID":"2532",
                     "STATUS":"SUCCEEDED",
                     "CHILD":[
                                    {"JOBNAME":"Import Invoices Report",
"JOBPATH":"/oracle/apps/ess/financials/payables/invoices/transactions",
                                     "REQUESTID":"2533",
                                     "STATUS":"SUCCEEDED"}
                               ]
                    }

                ],
    "SUMMARYSTATUS":"SUCCEEDED",
    "DOCUMENTID":"23456"
}
```

Figure 10: Sample response from callback

Correcting Load Process Errors

The Load Interface File for Import process ends in error when the load of the data file fails for any individual row. The Load File to Interface child process ends as an error or warning. All rows that were loaded by the process are deleted and the entire batch of records is rejected.

Correcting Interface Data Errors

To correct errors:

1. Review the upload error logs.

2. Change any structural or formatting anomalies in the data.

3. Generate the ZIP file containing the CSV files using the respective import template.

4. Upload the corrected file to the UCM server and resubmit the Load Interface File for Import process.

5. Repeat these steps until the process successfully loads all the data.

Correcting Import Process Errors

If the import process fails with errors:

1. Review the errors in the import log.

2. Correct the error records using the applicable ADFdi correction spreadsheets.

For a list of import processes and their corresponding ADFdi correction spreadsheets, see Appendix 7: Error Handling for Import Jobs.

If auto purge is enabled in your import process, then you cannot use ADFdi. Use these steps:

1. Download the purge erroneous ZIP file from the File Import and Export page.

2. Select the erroneous data records from the interface file and correct them.

3. Follow the FBDI process to resubmit the corrected data.

Purging Interface and Error Tables

Data from the interface and error tables can be purged as part of the following processes:

1. Each File-Based Data Import (FBDI) process initiates a purge process by default. Following the completion of the import process, the erroneous data to be purged will first be extracted and uploaded to the Oracle WebCenter Content Server (UCM).

2. Customers also have the capability to manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed. This process supports the purge of interface data created from either FBDI or non-FBDI sources.

The purge backup file is stored and associated with the respective UCM import account for reference where needed. The file can either be downloaded using the Oracle ERP Integration Service or the File Import and Export page. This file is a consolidated ZIP file that contains the individual interface and error data files in a comma separated values (CSV) format.

For data correction, select and revise any erroneous data from the respective interface spreadsheet file, then upload the revised interface file again to execute the FBDI process.

For the processes outlined above, the existing inbound, outbound, and erroneous data files older than 30 days that are stored on the UCM server will automatically be purged for the applicable UCM account.

Operation: extractAndPurge

The extractAndPurge operation extracts data from the interface and error tables, uploads the relevant data file to UCM, then purges the respective data.

The purge file naming convention is as follows: ImportBulkData_<ImportJobName>_<LoadRequestId>.zip

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Request IDs	The request ID(s) of load jobs.	IN	Yes	java.lang.String
Notification Code	A two-digit number that determines how and when a notification is passed for the status of the import job. See the table below for the notification code values.	IN	Yes	java.lang.String
Callback URL	The callback URL of the web service you implemented to receive the ESS job status upon job completion.	IN	No	java.lang.String
Job Options	There are no additional job options for this operation.	IN	No	java.lang.String

The following table provides information on the notification codes:

Digit Position	Digit Value	Meaning
First digit	1	E-mail notification
	2	Bell notification
	3	Email and bell notification
Second digit	0	Send in any case (import failed or succeeded)
	1	Send on import success
	2	Send on import failure

The following sample request payload illustrates the extractAndPurge process:

```
<soap:Body>
  <ns1:extractAndPurge
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
     <ns1:requestIds>1234;1235;1236</ns1:requestIds>
     <ns1:notificationCode>30</ns1:notificationCode>
     <ns1:callbackURL>#NULL</ns1:callbackURL>
     <ns1:jobOptions></ns1:jobOptions>
</ns1:extractAndPurge>
</soap:Body>
```

Figure 11: Sample request payload for the extractAndPurge operation

To manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed, see Appendix 17: Purge - UI Based Approach.

Advanced Features

Securing the Inbound or Outbound Data File

Since inbound and outbound data files are transmitted over the Internet and often contain the company's sensitive information and financial transactions like journal entries, invoices, payments and bank records, data encryption is a critical and essential element in implementing your integrations with Oracle ERP Cloud. You can secure data files between Oracle ERP Cloud and your on premise applications or systems including Platform as a Service (PaaS) applications. This topic describes how to set up and use encryption keys for secure file transfer. After you perform this setup, you can encrypt and decrypt files and transfer them between your servers and Oracle ERP Cloud using import and export bulk data processes.

Oracle ERP Cloud supports Pretty Good Privacy (PGP) unsigned encryption with 1024 bits key size. There are two types of encryption keys:

1. Oracle ERP Cloud PGP Key

2. Customer PGP Key

Oracle ERP Cloud PGP Key

A customer uses the public key to encrypt the inbound file. The import bulk data process will use the private key to decrypt the file before starting the load and import process. This key can be generated using the Security Console.

Customer PGP Key

A customer uses the private key to decrypt exported files from the Oracle ERP Cloud. The export bulk process will use the public key to encrypt the outbound file. The customer can import their public key into the Oracle ERP Cloud using the Security Console.

Note: Customers may use different keys for a different Cloud pod or the same key on multiple Cloud pods.

For more information on managing PGP keys, see Appendix 14: Managing PGP Encryption Keys.

Enabling Encryption in the Import Process

A customer encrypts inbound data file using the cloud public key. Oracle ERP Cloud decrypts this file using a cloud private key before starting the load and import process. These are the following steps to enable encryption in the import process:

1. Encrypt the data ZIP file using an Oracle ERP Cloud public key. To encrypt inbound data file, see Appendix 15: How to Encrypt and Decrypt a Data File.

2. In the payload for the importBulkData operation, specify the following job options:

Options	Value
FileEncryption	PGPUNSIGNED or PGPSIGNED
FA_ALIAS	Oracle ERP Cloud Key Alias Name
CUSTOMER_ALIAS	Customer Key Alias Name

Example:

`<typ:jobOptions>FileEncryption=PGPUNSIGNED,FA_ALIAS=<ERP_CLOUD_KEY>,CUSTOMER_ALIAS=<CUSTOMER_KEY></typ:jobOptions>`

Note: Alias names are defined when you generate an Oracle ERP Cloud key or import a customer key.

Enabling Encryption in the Export Process

When enabled, Oracle ERP Cloud encrypts an extracted data file using a customer's public key and uploads the file to UCM. These are the following steps to enable encryption in the export process.

1. In the payload for the exportBulkData operation, specify the following job options:

Options	Value
FileEncryption	PGPUNSIGNED or PGPSIGNED
FA_ALIAS	Oracle ERP Cloud Key Alias Name
CUSTOMER_ALIAS	Customer Key Alias Name

Note: Alias names are defined when you generate an Oracle ERP Cloud key or import a customer key.

Decrypt the output file using the customer private key. To decrypt an outbound data file, see Appendix 15: How to Encrypt and Decrypt a Data File.

Job Property File for the Bulk Import Process

The Job Details parameter in the importBulkData operation includes the job definition and package names, as well as the job parameters of the imported object. To get the job package and definition name, see Viewing Details about Predefined Scheduled Processes: Procedure in the File-Based Data Import for Oracle Financials Cloud guide in the Oracle Help Center at http://docs.oracle.com . Use the following advanced options to specify the job details data associated with the importBulkData operation:

* Generate and add the Job Properties file to the data ZIP file

* Generate and upload the Job Properties file to the UCM applicable account for reusability

See Appendix 12: Creating a Job Property File for the importBulkData Operation for detailed information on how to generate the Job Property file.

Note
Parameter Precedence: 1. Payload Parameter File in the ZIP data file   2. Parameter File stored on UCM

Option 1: Job Property File as Part of the Data ZIP File

The following sample request payload illustrates the Journal Import process with the relevant parameter file, included together with the import data file in a ZIP file:

```
<soap:Body>
  <ns1:importBulkData
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>

  <ns1:document
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/">
  <ns2:Content>
UEsDBBQAAAAIAKSUz0guTciD4QAAAKwDAAAPAAAAR2xJbnRlcmZhY2UuY3N2ZJBS8NAEIXvgv9hf8A0mZmdDbu9tSR6OSK26jm0owRiIkk
9+O/dqDebSxc6h2UZvsfjPWZTvQABI/qcOLcEOx2P67qtu73Co773RzW7oe7GVx1ge1tGNHIoOQZYARIgIhARTp+fB+ZmcilcZj0AsZBkRW
RvPtvW/PpFo68PNQtTrbfmTg9vOizNczM2fWf+tg86NP1haUrdL9BDsvz/BCbvAs9mqDbl9dXJ0sxca2V1n9JaxIIjDiGzkSqYg0jATIrk9
Inyi5S3eko6OWudd/HoJobFCqFkwsnRE+VnNvcNUEsDBBQAAAAIAIeUz0g8JMczbAAAAH4AAAAdAAAASm91cm5hbHHNJbXBvcnRURVVNULnBy
b3BlcnRpZXZM1yDEKQjEMBuBd8A4eIBC7uDs4KEUE3wV+a6wP2qQkvsHbi4h822eO0oQxRrBE8GNWaJnRgquoOFqWexXn4VYdPbhY76Z0ssU
V7diH+Stj0fIU/2/8ejpcJ0rbXaIL3rg1CUq0z5nOX5v16gNQSwECFAAUAAAACACklM9ILk3Ig+EAAACsAwAADwAAAAAAAAAACAAAAAAA
AAR2xJbnRlcmZhY2UuY3N2UEsBAhQAFAAAAAgAh5TPSDwkzNsAAAAfgAAAB0AAAAAAAAAAgAAAADgEAAEpvdXJuYWxzSW1wb3J0VEVTTV
C5wcm9wZXJ0aWVzUEsFBgAAAAACAAIAiAAAALUBAAAAAA==</ns2:Content>
  <ns2:FileName>JournalsImportTEST_1234.zip</ns2:FileName>
  </ns1:document>
  <ns1:jobDetails></ns1:jobDetails>
  <ns1:notificationCode>30</ns1:notificationCode>
  <ns1:callbackURL>http://hostname:port/myCallbackService</ns1:callbackURL>
  <ns1:jobOptions></ns1:jobOptions>
  </ns1:importBulkData>
</soap:Body>
```

Figure 12: Sample request payload for the Journals Import process with the parameter file included with the import data in a ZIP file

The following sample job property file for Journals Import (JournalsImportTEST.properties) is included with the import data file in a ZIP file:

```
oracle/apps/ess/financials/generalLedger/programs/common,JournalImportLauncher,JournalsI
mportTEST,1061,Payables,1,ALL,N,N,N
```

Figure 13: Sample parameter file for Journals Import

See Appendix 12: Creating a Job Property File for the importBulkData Operation for detailed information on creating the job properties file.

Option 2: Upload the Job Properties File to UCM for Reuse

After the file is uploaded to the UCM applicable account, there are two options to reuse the file:

1. Add JobDetailFileName=<FileName.properies> in <jobOptions>

2. Follow file naming convention as defined in Appendix 12: Creating a Job Property File for the importBulkData Operation.

The following sample payload illustrates the Journal Import process with the job properties file uploaded to UCM. The parameter file should be uploaded using the specific UCM account associated with a particular import process:

```
<soap:Body>
  <ns1:importBulkData
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>

  <ns1:document
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/">
  <ns2:Content>
UEsDBBQAAAAIAKSUz0guTciD5gAAAKwDAAAPAAAAR2xJbnRlcmZhY2UuY3NtZLBasMwDIbvg72DHkB1JFkJcW8tyXbZwliz7hxabwSyZCT
dYW8/Z/SWFgZm/8EY+/sl9KOqfEVGIcoTlsQy1n46bZuu6Q8en/3HcPJQj00/vfkRd/dFQANHmpDDDRIjESEz03z5PfCqQpcsNTZHZFFWkw
X27qvr4NwP6u9PDysotzt48Md3P65h307t0MP59cmP7XBcQ+EPKwp1Yu1LOeE8dXLxb1ZZFbc31aXQ4FpqRfkYk1rAXMrinLGBykScqiOjW
fT0kfal/iG8zUvUylmb5mlYupkRtcqkRiV69Ej7Un9K7gdQSwECFAAUAAAACACklM9ILk3Ig+YAAACsAwAADwAAAAAAAABACAAAAAAAAA
R2xJbnRlcmZhY2UuY3N2UEsFBgAAAABAAEAPQAAABMBAAAAAA==</ns2:Content>
  <ns2:FileName>JournalsImportTEST_1234.zip</ns2:FileName>
  </ns1:document>
  <ns1:jobDetails></ns1:jobDetails>
  <ns1:notificationCode>30</ns1:notificationCode>
  <ns1:callbackURL>http://hostname:port/myCallbackService</ns1:callbackURL>
  <ns1:jobOptions>JobDetailFileName=JournalsImportTEST.properties</ns1:jobOptions>
  </ns1:importBulkData>
</soap:Body>
```

Figure 14: Sample request payload for the Journals Import process with the parameter file uploaded to UCM

Specifying Multiple Threads in Bulk Import

To increase the throughput when importing data, users can specify multiple threads in the import process. It supports a maximum of 10 threads for sequential processing and a maximum of 5 threads for parallel processing. After the data file is loaded in the interface table, the import process will start batch processing based on number of job parameters records defined in a property file and job option in the payload. The default option is sequential and the following property in <jobOptions> attribute could enable parallel processing:

<jobOptions>ExecutionMode=Parallel<jobOptions>

In a job property file, you must enter multiple records with parameter values that can import data in batches either sequentially or concurrently. In sequential pattern, an import process stops when a batch fails. The remaining batch processes will not be executed and callback will include all the details including the failed process.

The following is the sample property file of journals import where data file contains 3 different ledgers:

oracle/apps/ess/financials/generalLedger/programs/common,JournalImport
Launcher,GL,1061,Payables,**1**,ALL,N,N,N
oracle/apps/ess/financials/generalLedger/programs/common,JournalImport
Launcher,GL,1061,Payables,**2**,ALL,N,N,N

oracle/apps/ess/financials/generalLedger/programs/common,JournalImport

Launcher,GL,1061,Payables,**3**,ALL,N,N,N

ERP will import three ledgers either sequentially or concurrently depending on the "ExecutionMode" type. The job package and name must be same for all the records.

Optimized Management of Large Data Files

The Oracle ERP Integration Service provides the capability to attach data files instead of converting data files to base64 encoding. The attachment feature leverages the Message Transmission and Optimization Mechanism (MTOM) approach by reducing the request payload size as file content is not part of the payload (in base64 encoding). This process optimizes the handling of large files for both inbound and outbound processes. You need minor changes in your web service proxy code to enable MTOM support.

For more information about the MTOM changes, see Appendix 16: Large File Optimization (MTOM) Proxy Client Code Changes.

Appendix 1: Security Prerequisites to Download the Job Output File

ESS job and output files are placed in the Attachments Security group under the Oracle Universal Content Management server (Oracle WebCenter Content server). You must have access to the security group called Attachments to download the log file or the output file with the ERP Integration Service.

This access can be granted via the security role called AttachmentsUser.

Use the Security Console to grant access to the AttachmentsUser role. The Security Console can be accessed in the following ways:

- Use the Manage Job Roles or Manage Duties tasks in the Setup and Maintenance work area.
- Select **Navigator** - **Tools** - **Security Console**

Access to the Security Console is provided by the predefined IT Security Manager role.
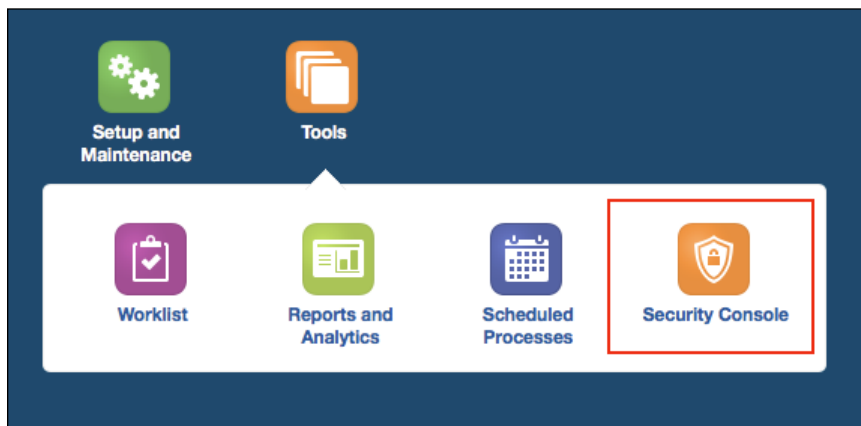
Figure 15: Accessing the Security Console from the Navigator

The role AttachmentsUser is inherited by the predefined Employee and Contingent Worker roles. You can verify this inheritance by querying the role AttachmentsUser from the Security Console, and use the Expand Toward **Users** and show the **Roles** option.

Figure 16: Verifying inheritance of AttachmentUser role

After reviewing the role inheritance of the AttachmentsUser role, review the users that are currently assigned the AttachmentsUser role.

You can verify role assignments to users by querying the role AttachmentsUser from the Security Console and use the Expand Toward **Users** and show **Users** option.

Figure 17: Review user assignments to AttachmentsUser role

In Figure 17 above, the user John.Reese have been assigned the AttachmentsUser role through the predefined Employee role.

Lastly, verify that the Attachments security group is listed in the UCM Search page.
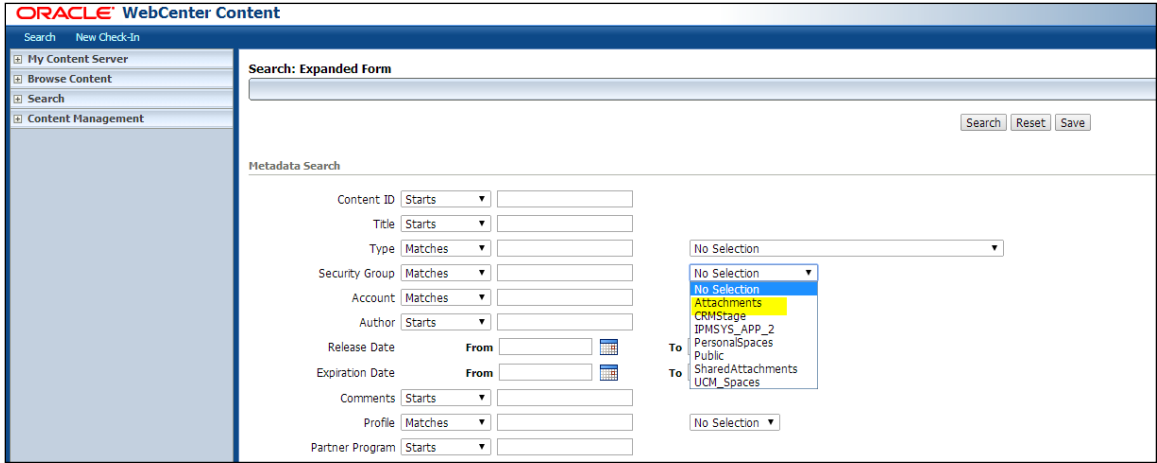
Figure 18: Search page for UCM to identify whether user has access to Attachments security group

Appendix 2: Sample Code for Preparing a Data File for Inbound and Outbound Flow

The following example illustrates sample code for preparing a data file for the inbound flow.

```
Sample File Name: utilEncodeBase.java

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import org.apache.commons.codec.binary.Base64;

public class utilEncodeBase {
    public utilEncodeBase() {
        super();
    }

 public static void main(String[] a) throws Exception {

        // Enter the filename as input
        File br = new File(a[0]);
        // Convert the file into Byte
        byte[] bytes = loadFile(br);

        // Call the api for Base64 encoding
        byte[] encoded = Base64.encodeBase64(bytes);
        String encStr = new String(encoded);
        // Print the file
        System.out.println(encStr);

    }   private static byte[] getByteArray(String fileName) {
        File file = new File(fileName);
        FileInputStream is = null;
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        int nRead;
        byte[] data = new byte[16384];
        try {
            is = new FileInputStream(file);
            while ((nRead = is.read(data, 0, data.length)) != -1) {
                buffer.write(data, 0, nRead);
            }
            buffer.flush();
        } catch (IOException e) {
            System.out.println("In getByteArray:IO Exception");
            e.printStackTrace();
        }
        return buffer.toByteArray();
    }   private static byte[] loadFile(File file) throws IOException {
        InputStream is = new FileInputStream(file);

        long length = file.length();
        if (length > Integer.MAX VALUE) {
            // File is too large
        }
        byte[] bytes = new byte[(int)length];

        int offset = 0;
        int numRead = 0;
        while (offset < bytes.length &&
                (numRead = is.read(bytes, offset, bytes.length - offset)) >=
                0) {
            offset += numRead;
        }
        if (offset < bytes.length) {
            throw new IOException("Could not completely read file " +
                                    file.getName());
```

```
        }
        is.close();
        return bytes;
    }
}
```

Figure 19: Sample code for the inbound data flow

The following example illustrates sample code for preparing a data file for the outbound flow.

```
Sample FileName: utilDecodeBase.java

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;

import org.apache.commons.codec.binary.Base64;

public class utilDecodeBase {
    public utilDecodeBase() {
        super();
    }


    public static void main(String[] a) throws Exception {
        System.out.println("Start");

        // Read the inputsteam
        String encStr = a[0];

        // Run the api to perform the decoding
      byte[] rbytes = Base64.decodeBase64(encStr.getBytes());

        // Put the location for the output file
        FileOutputStream os = new FileOutputStream("/tmp/Test1234.zip");
        os.write(rbytes);
        os.close();

    }
}
```

Figure 20: Sample code for the outbound data flow

Appendix 3: Predefined Target UCM Accounts

You can transfer data files to predefined accounts in the Oracle WebCenter Content server (UCM) that correspond to the interface table.

To find the UCM account:

1. Open the File Based Data Import guide for your cloud service.

2. Locate your respective import process. For example, **Journal Import**.

3. View the UCM account in the Details section.

Appendix 4: ESS Job Execution Status

The following table lists the execution statuses of the ESS jobs with descriptions. These statuses are returned by the getEssJobStatus operation.

ESS Job Execution Status	Description	User Action
COMPLETED	Request has completed.	This is an intermediary status and is returned when the child processes are generated. Check the status of the ESS job to ascertain whether the return status changes to SUCCEEDED or ERROR.
BLOCKED	Request is blocked by one or more incompatible requests.	Wait for the completion of the incompatible request and resubmit the process.
SUCCEEDED	Request completed and was successful.	Check the details of the completed process and proceed with any post processing.
ERROR	Request ran and resulted in error.	Download the details of the error and correct the data.
ERROR_AUTO_RETRY	Request ran, resulted in an error, and is eligible for automatic retry.	Resubmit the process after some time.
WARNING	Request ran and resulted in a warning.	Download the details of the process. Check the reason for the warnings and take the necessary action to correct the input data.
RUNNING	Request is processed.	No action.
CANCELED	Request was canceled.	Resubmit the request if required.

Appendix 5: Testing Web Service using a Client Proxy

Perform the following steps to test a web service operation using JDeveloper:

1. Import a new certificate in the keystore (for Internet Explorer).

2. Create a web service client proxy and add the OWSM policy.

3. Test the web service.

Steps to Import a New Certificate in the Keystore

1. Export the certificate from the browser to the file, using the following steps:

    i. Access the SSL URL for any web service using Microsoft Internet Explorer.

    ii. In Internet Explorer, click **Tools** > **Internet Options**.

    iii. On the **Content** tab, click **Certificates**. On the **Personal** tab, select the **Baltimore CyberTrust Root** certificate and click **View**. The certificate hierarchy appears; export the top two certificates (**Baltimore CyberTrust Root** and **Verizon Akamai SunServer CA G14-SHA1**).
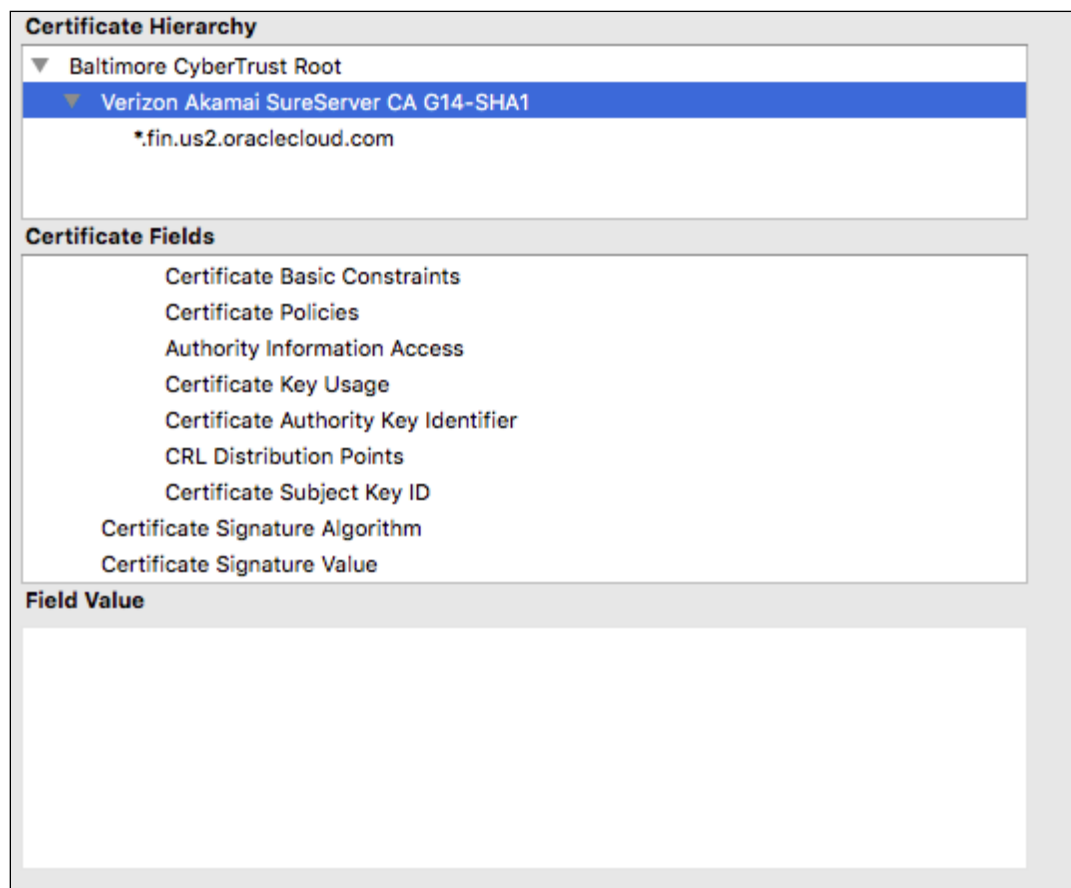
Figure 21: Select the certificate

iv.   On the **Certification Path** tab, select **Baltimore CyberTrust Root** and click **View Certificate**.

v.   On the **Details** tab, select **Copy to File**. The Certificate Export Wizard appears.

vi.   Click **Next** > **Next** and enter a name and location for the file you want to export.

vii.   Change the encoding to Base-64 and click **Next**.

viii.   Provide the file name and click **Finish**.

ix.   Repeat steps iv to viii for the **Verizon Akamai SunServer CA G14-SHA1** certificate.

When using other web browsers, perform similar steps. The navigation path may differ in the web browsers used.

2. Type the following command to import a certificate into keystore:
```
keytool -import -trustcacerts -file <filename> -alias <aliasname> -keystore default-
keystore.jks - storepass welcome1
```

3. Run the following command to verify if the trust store contains the imported certificates:

```
keytool -list -v -keystore <filename>  -storepass welcome1 | grep -i Verizon
```

Create a Proxy Client and Add the OWSM Policy

1. Create a new project and select **Web Services Proxy**.

2. Set the client style to **JAX-WS Style**.

3. Select the web service description, for example,
```
https://<Hostname>.<Domain Name>:<Port
No>/publicFinancialCommonErpIntegration/ErpIntegrationService?WSDL.
```

4. Select the **Copy WSDL Info Project** check box. Specify the default mapping options.

5. Specify the asynchronous method.

6. Select the **Do not generate any asynchronous methods** option.

7. Click **Finish**.

8. Once the proxy client code is generated, add the following variables:

- jksFPath: File location that has the certificate to add to the keystore. For example,
  ```
  D:\fintuilwdestapp\Project5\client.jks
  ```

- jksPassword: Password to access WSDL. For example, **Welcome1**.

- trustStore: Path where the certificates are stored, used during java installation by default.

- trustStorePassword: Password for truststore.

- Username: User name to sign in to the service.

- Password: Password for the user to sign in to the service. For example, **Welcome1**.

- endpointNonSSLURL: URL for the FinUtilService service.

- serviceName: Schema of the service used to add the policies.

- securityFeature: Policy used to add to the service.

> **Note**
>
> An example of a message protection policy is policy:oracle/wss_username_token_over_ssl_client.

9. Create the invokeServiceWithUsernameTokenMessageProtectionPolicy() method to add policy.

Test Upload File to UCM using Web Service

To test the file upload to the UCM server:

1. Create a sample payload associated with the uploadFileToUcm operation.

2. Create the method `invokeUpload` to call the operation uploadFileToUcm.

Export the Certificate

To export the certificate associated with the web service from the browser, invoke the end point URL for the web service `https://<hotaname>.<domainname>/publicFinancialCommonErpIntegration/ErpIntegrationService?WSDL`.

1. Copy the content from the XML element `dsig:X509Certificate`.

Figure 22: Sample content of the dsig:X509Certificate

2. To use the key contained in this certificate, create a local KeyStore and import the certificate into it:

2.1. Create a new file with any name you like. You must change the extension to .cer to indicate that it is a certificate file.

2.2. Using a text editor, open the file you just created and enter "----BEGIN CERTIFICATE----"on the first line.

2.3. In the next line, copy the Base64 encoded certificate from the service WSDL file to the newly created certificate file.

2.4. Add "-----END CERTIFICATE-----" on a new line and save the file. Now you have a certificate containing the public key from the server.

2.5. Open the command line and change the directory to $JAVA_HOME/bin. Use the following command to create a KeyStore and import the public key from the certificate:

keytool -import -file <Provide the path of the certification.cer file> -alias orakey - keypass welcome -keystore <Provide the path where the jks file needs to be created(including the file name)> -storepass welcome.

Figure 23: Sample certificate file (<Filename>.cer)

3. Add the variables to the proxy client code.

```
ErpIntegrationServiceSoapHttpPortClient.java
public class ErpIntegrationServiceSoapHttpPortClient
{
@WebServiceRef
private static ErpIntegrationService_Service ErpIntegrationService_Service;
private static final AddressingVersion WS_ADDR_VER = AddressingVersion.W3C;
// Add the additional variables
  private final String jksFPath = "D:\\fintuilwdestapp\\Project5\\client.jks";
  private final String jksPassword = "Welcome1";
  private final String trustStore = "C:\\ProgramFiles\\Java\\jdk1.7.0_51\\jre\\lib\\security\\cacerts";
  private final String trustStorePassword = "";
  private final String username = "finuser1";
  private final String password = "Welcome1";
  private String endpointNonSSLURL = "https://efops-rel9st1-cdrm1-external-fin.us.oracle.com/
publicFinancialCommonErpIntegration/ErpIntegrationService";
  private static final QName servicename = new
QName("http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/","ErpIntegra
tionService");
  private SecurityPolicyFeature[] securityFeature = new SecurityPolicyFeature[] { new
SecurityPolicyFeature("policy:oracle/wss_username_token_over_ssl_client_policy") };
  private ErpIntegrationService ErpIntegrationService;
// End add the additional variables
public static void main(String [] args)
   {  System.out.println("inside main");
     ErpIntegrationServiceSoapHttpPortClient f = new ErpIntegrationServiceSoapHttpPortClient();
     f.invokeServiceWithUsernameTokenMessageProtectionPolicy();
     String retStatus = f.invokeUpload();
     //long submittedJobId = f.invokeSubmitJob(retStatus);
     //f.invokeEssJobStatus(submittedJobId);
     //f.invokeDownloadESSJobExecDetails(submittedJobId);
  }
```

Figure 24: Sample proxy code with variables

4. Create the invokeServiceWithUsernameTokenMessageProtectionPolicy()method to add the policy.

```
public void invokeServiceWithUsernameTokenMessageProtectionPolicy() {
  System.out.println("inside invokeservice");
  URL wsdlDoc = null;
  try {
    wsdlDoc = new URL("https://efops-rel9st1-cdrm1-external-
fin.us.oracle.com/publicFinancialCommonErpItegration/ErpIntegrationService?wsdl");
  }catch(MalformedURLException e){
    e.printStackTrace();
  }
  System.setProperty("javax.net.ssl.trustStore", trustStore);
  System.setProperty("javax.net.ssl.trustStorePassword", trustStorePassword);
  ErpIntegrationService_Service = new ErpIntegrationService_Service(wsdlDoc, servicename);
  ErpIntegrationService =
ErpIntegrationService_Service.getErpIntegrationServiceSoapHttpPort(securityFeature);
  WSBindingProvider wsbp = (WSBindingProvider)ErpIntegrationService;
  Map<String, Object> requestContext = wsbp.getRequestContext();
  requestContext.put(BindingProvider.USERNAME_PROPERTY, username);
  requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);
  requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, endpointNonSSLURL);
  requestContext.put(ClientConstants.WSSEC_KEYSTORE_TYPE, "JKS");
  requestContext.put(ClientConstants.WSSEC_KEYSTORE_LOCATION, jksFPath);
  requestContext.put(ClientConstants.WSSEC_KEYSTORE_PASSWORD, jksPassword);
  System.out.println("Finished invokeservice");
}
```

Figure 25: Sample method code to add the policy

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns1:uploadFileToUcm
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/types/">
         <ns1:document
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/">
            <ns2:Content>UEsDBBQAAAAIAEhrrkQSDhHq5BLBQYAAAAAAQABAD0AAAATPQEAAAA=</ns2:Content>
            <ns2:FileName>TestUploadFileName.zip</ns2:FileName>
            <ns2:ContentType>zip</ns2:ContentType>
            <ns2:DocumentTitle>Sample File1</ns2:DocumentTitle>
            <ns2:DocumentAuthor>finuser1</ns2:DocumentAuthor>
            <ns2:DocumentSecurityGroup>FAFusionImportExport</ns2:DocumentSecurityGroup>
            <ns2:DocumentAccount>fin$/generalLedger$/import$</ns2:DocumentAccount>
         </ns1:document>
      </ns1:uploadFileToUcm>
   </soap:Body>
</soap:Envelope>
```

Figure 26: Payload for the uploadFileToUCM operation

```
private String invokeUpload()  {
    System.out.println("inside invokeupload");
    ObjectFactory objectFactory = new ObjectFactory();
    DocumentDetails documentDet = new DocumentDetails();
    String fileNameWithPath = "C:\\Users\\NGARLAPA\\Desktop\\Sample.zip";
    String fileName = "Sample.zip";
    String contentType = "zip";
    String title = "Journals Import";
    String ucmAccountInfo = "fin$/generalLedger$/import$";
    String ucmSecurityGroup = "FAFusionImportExport";
documentDet.setContent(objectFactory.createDocumentDetailsContent(getByteArray(fileNameWithPath)));
documentDet.setContentType(objectFactory.createDocumentDetailsContentType(contentType));
documentDet.setDocumentAccount(objectFactory.createDocumentDetailsDocumentAccount(ucmAccountInfo));
documentDet.setDocumentAuthor(objectFactory.createDocumentDetailsDocumentAuthor(username));
documentDet.setDocumentSecurityGroup(objectFactory.createDocumentDetailsDocumentSecurityGroup(ucmSecuri
tyGroup));documentDet.setDocumentTitle(objectFactory.createDocumentDetailsDocumentTitle(title));
documentDet.setFileName(objectFactory.createDocumentDetailsFileName(fileName));
 UploadFileToUcm uploadFileToUcm = new UploadFileToUcm();
 uploadFileToUcm.setDocument(documentDet);
UploadFileToUcmResponse retStatus = null;
    try {
       retStatus = ErpIntegrationService.uploadFileToUcm(uploadFileToUcm);
       System.out.println("File successfully Uploaded.Status is:" + retStatus.getResult());
        } catch (Exception e) {
         e.printStackTrace();
       }
   return retStatus.getResult();
 }
 private byte[] getByteArray(String fileName) {
    File file = new File(fileName);
    FileInputStream is = null;
    ByteArrayOutputStream buffer = new ByteArrayOutputStream();
    int nRead;
    byte[] data = new byte[16384];
    try {
      is = new FileInputStream(file);
      while ((nRead = is.read(data, 0, data.length)) != -1) {
        buffer.write(data, 0, nRead);
         }
      buffer.flush();
      } catch (IOException e) {
         System.out.println("In getByteArray:IO Exception");
         e.printStackTrace();
       }
    return buffer.toByteArray();
  }
```

Figure 27: Method invokeUpload() to invoke the uploadFileToUCM operation

Appendix 6: Automate Web Service Invocation Using JDeveloper 11

The automation approach for the web service invocation includes the following:

- Compile the web service client proxy project created in Appendix 5.

    - Add the following JDeveloper 11g jars to the deployment profile:

        - weblogic.jar : `$MW_HOME/wlserver_10.3/server/lib/weblogic.jar`

        - jrf.jar: `$MW_HOME/oracle_common/modules/oracle.jrf_11.1.1/jrf.jar`

- Create the deployment profile for the project:

    - Click **Project Properties.**

    - Click **Deployment**.

    - Click **New**.

    - Select **Archive Type** as JAR File and specify the name.

    - Select the **Include Manifest File** (META-INF/MANIFEST.MF) option and specify the value for Main Class.
      For example,
      ```
      oracle.apps.finacial.testUtil.proxy.client.ErpIntegrationServiceSoapHttpPortC
      lient
      ```

- Generate the jar file and execute the following command.
  For example, `java –classpath $CLASSPATH –jar <JAR File Name>`

Appendix 7: Error Handling for Import Jobs

To address errors generated during the import process, use the following methods:

- If an ADFdi correction spreadsheet is available, use the spreadsheet to correct the data in the interface table and resubmit the import process.

- If no correction spreadsheet is available, use the purge process to delete all the records from the interface table. Correct the data in the original data file and upload the file again using the correct UCM account.

Repeat the process until your data is successfully imported.

Error Handling Processes

The following table lists the existing error handling processes which may be used to address any errors encountered during the import process for each respective product interface table.

Import Process	Correction Spreadsheet	Steps
Import Payables Invoices	CorrectImportErrors.xlsx	In the Invoices work area, navigate to Correct Import Errors in the Tasks region.
Import AutoInvoice	ManageInvoiceErrors Spreadsheet.xlsx	In the Billing work area, navigate to the Review AutoInvoice Errors table. Click **Number of Errors**. Select the **Manage AutoInvoice Lines** spreadsheet.
Process Receipts through Lockbox	ManageLockboxErrors.xlsx	In the Receivables work area, navigate to Receivable Balances.
Fixed Asset Mass Additions Import	PrepareSourceLines.xlsx	In the Fixed Assets work area, navigate to Additions. Select **Pending Source Lines**.
Fixed Asset Mass Adjustments Import	UpdateMassFinancialTrans action.xlsm	In the Fixed Assets work area, navigate to Financial Transactions. Select **Pending Mass Financial Transactions**.
Fixed Asset Mass Retirements Import	UpdateMassRetirements.xls m	In the Fixed Assets work area, navigate to Retirements. Select **Pending Retirements**.
Fixed Asset Mass Transfers Import	UpdateMassTransfers.xlsm	In the Fixed Assets work area, navigate to Pending Mass Transfers. Select **Pending Mass Transfers**.
Journal Import	JournalCorrections.xlsx	In the Journals work area, navigate to Correct Import Errors in the Tasks region.

Appendix 8: Using XML Templates to Generate Data Files

The File Based Data Import guides in the Oracle Help Center (http://docs.oracle.com) include XML integration templates that you can use with Oracle Data Integrator (ODI) to generate import files from your external data. Oracle Data Integrator provides a solution for integrating complex data from a variety of sources into Oracle Fusion applications.

To use the XML templates and generate the import files, you must:

- Install and set up Oracle Data Integrator

- Create source and target models

- Create integration projects

Note

For Oracle Cloud implementations, you must upload the ZIP file to the content management repository in Oracle Cloud. For non-Cloud implementations, you can streamline the data integration process by installing the content management document transfer utility, which uses Oracle Data Integrator to transfer the ZIP file.

Installing and Setting Up Oracle Data Integrator

To use the XML templates for generating the import files:

1. Install Oracle Data Integrator.

   For more information about installing Oracle Data Integrator, see Oracle Fusion Middleware Installation Guide for Oracle Data Integrator.

2. Set up Oracle Data Integrator.

   For more information about setting up Oracle Data Integrator, see Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator.

Creating Source and Target Models

Create ODI models for both the source and target data stores. You determine the source models based on the system or technology of the external data that you need to import into your Oracle Fusion application. Create the target models by importing the XML files that you download from the Details section of the File Based Data Import guides in the Oracle Help Center (http://docs.oracle.com).

For more information on creating a reverse engineering model, see Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator.

Configuring Integration Projects

Create and configure an integration project by selecting the knowledge modules, creating the interfaces, and mapping the source and target data stores.

For more information on creating an integration project, see Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator.

Opening the XML Template

To prepare your data in Oracle Data Integrator, download the XML templates using the following steps:

1. Import the family-level template as a model folder.

2. Import the product-level template as a model folder within the family-level model folder.

3. Import the product template as a model within the product-level model folder.

4. Create the integration project.

5. Create the package.

6. Add and configure:

   - Integration projects

   - Content management document transfer utility

7. Execute the package. The package generates the CSV file and compresses it into a ZIP file.

Using XML Integration Templates to Generate Data Files

Use XML templates in Oracle Data Integrator to prepare your external data for the load and import process.

The File Based Data Import guides in the Oracle Help Center (http://docs.oracle.com) include three types of XML templates that you import as target models in your Oracle Data Integrator repository:

- Family level

- Product level

- Product

Family-Level XML Files

A family-level XML file is common to a group of product-level model folders and product models.

Consider the following points when you use family-level XML files:

- Use the family-level XML file to support assets in the family, for example, Oracle Fusion Financials or Oracle Fusion Human Capital Management.

- Import the family-level XML file into your Oracle Data Integrator repository prior to importing other XML files.

- Import one family-level XML file as a model folder for each family of products.

- Import each family-level XML file as a top-level model folder.

- Import the family-level XML file one time; it supports all subsumed product-level model folders.

- Select Synonym Mode Insert Update as the import type.

Product-Level XML Files

A product level XML file is common to a group of product models.

Consider the following points when you use product-level XML files:

- Use the product-level XML file to support assets in the product line, for example, Fixed Assets, General Ledger, or Payables.

- Import one product-level XML file as a model folder for each line of products.

- Import the product-level XML file as a model folder into your Oracle Data Integrator repository.

- Import the family-level XML file before you import product XML files.

- Import each product-level XML file as a mid-level model folder within the appropriate family-level model folder.

- Import the product-level XML file one time; it supports all subsumed product models.

- Select Synonym Mode Insert Update as the import type.

Product XML Files

A product XML file represents a specific interface table asset.

Consider the following points when you use product XML files:

- Import one product XML file as a model for each interface table or set of tables, for example, MassAdditions.

- Import the product XML file as a model into your Oracle Data Integrator repository after you import the product-level XML file.

- Import each product XML file as a model within the appropriate product-level model folder.

- Import each product XML file one time. The model is based on File technology.

- Select Synonym Mode Insert Update as the import type.

- After you import the product model, connect the model to the correct logical schema.

Creating Integration Projects That Generate Data Files for Import

When you use Oracle Data Integrator to generate the import data files from the external data sources, you must configure an integration project. Integration projects are collections of ODI components that provide the procedural details of an integration from a source to a target. The source is your external data and the target is the import data file that you load and import into your Oracle Fusion Applications.

To create your integration project, you configure the following components:

- Knowledge modules

- Integration interfaces

Knowledge Modules

Knowledge modules contain the information that Oracle Data Integrator requires to perform a specific set of tasks against a specific technology or set of technologies. For example, check knowledge modules ensure that constraints on the sources and targets are not violated, and integration knowledge modules load data to the target tables.

Consider the following points about knowledge modules:

- Knowledge modules that you import into your integration project depend on the source and target technologies, as well as other integration-specific rules and processes.

- Multiple types of knowledge modules exist in ODI.

- Use the **SQL File to Append** module to create the import data file.

Integration Interfaces

Integration interfaces contain the sets of rules that define the loading of data from one or more sources to the target.

Consider the following points about integration interfaces:

- The source is the datastore from your external data model.

- The target is the interface table datastore, which is the CSV file from your interface table model.

- After you set up the source and target datastores, map the target fields to the source fields, or map source field values to target fields or constants.

Appendix 9: Manage Inbound Flow Automation Steps with Separate Web Service Operations

Inbound data integration is achieved seamlessly using the importBulkData operation from the Oracle ERP Integration Service. However, for certain use cases, you may want to control the flow and orchestration using separate operations.

The following section describes how to import journals using the individual web service operations. The same process applies to all the supported FBDI objects.

Perform the following steps to control the orchestration flow using individual operations:

1. Generate the CSV file as previously outlined.

2. Use the following operations to control the orchestration flow:

    a. loadandImportData

    b. uploadFileToUcm

    c. submitESSJobRequest (load to interface tables)

    d. getEssJobStatus

    e. submitESSJobRequest (load from interface tables to application tables)

    f. getEssJobStatus

    g. downloadESSJobExecutionDetails

3. Prepare the payload to upload the file to the UCM server by invoking the web service.

    The following table lists the payload input parameters for the uploadFileToUcm operation:

Payload Parameter Name	Value	Comments
Content	<Output from the sample utilEncodeBase.java program>	Run the Base64 encoding sample program to provide the content value. This sample program creates a ZIP file as input and provides the content output.
ContentType	Zip	Generate the CSV file. The CSV file creates a ZIP file. Other content types include .txt, and .csv.
DocumentTitle	<Sample Journal Import>	
DocumentAuthor	<USER1>	
DocumentSecurityGroup	FAFusionImportExport	This is a fixed value and must not be changed.
DocumentAccount	fin$/generalLedger$/import$	To construct this value and add a dollar sign ($) as an escape character before a slash (/), see Appendix 3: Predefined Target UCM Accounts.

Use the service tester to call the web service with the payload. Check the response for the document ID. If there is an error, check the error code in the response.

The uploadFileToUcm operation uploads a file to the UCM server based on the document specified. This operation requires application user access.

The following table lists the parameters for the uploadFileToUcm operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Document	List of elements, each containing the details of the file to be uploaded. The details include the file content, file name, content type, file title, author, security group, and account. Document Attributes: • Content: File content uploaded to the UCM server. The value of the content tag is obtained by converting the file content into Base64 encoding. For a sample program for Base64 encoding, see Appendix 2: Sample Code for Preparing a Data File for Inbound and Outbound Flow. • FileName: Name of the file on the UCM server. • ContentType: Type of content uploaded such as .zip, .txt, or .csv. • DocumentTitle: Title of the file on the UCM server. • DocumentAuthor: Author of the document. • DocumentSecurityGroup: A fixed value used to import and export the documents on the UCM server from Oracle Fusion. The security group for all the import process is FAFusionImportExport. • DocumentAccount: Account under which the file is uploaded. For more information on the UCM account associated with the ESS process, see Appendix 3: Predefined Target UCM Accounts.	IN	Yes	java.lang.String
Return	Returns the document ID of the uploaded file.	OUT	No	java.lang.String

The following sample payload illustrates the Journal Import process:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:uploadFileToUcm
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/erpIntegrationService/types/">
            <ns1:document
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/erpIntegrationService/">
                <ns2:Content>UEsDBBQAAAAIAEhrrkQSDhHq5BLBQYAAAAAQABAD0AAAATPQEAAAA=</ns2:Content>
                <ns2:FileName>TestUploadFileName.zip</ns2:FileName>
                <ns2:ContentType>zip</ns2:ContentType>
                <ns2:DocumentTitle>Sample File1</ns2:DocumentTitle>
                <ns2:DocumentAuthor>finuser1</ns2:DocumentAuthor>
                <ns2:DocumentSecurityGroup>FAFusionImportExport</ns2:DocumentSecurityGroup>
                <ns2:DocumentAccount>fin$/generalLedger$/import$</ns2:DocumentAccount>
            </ns1:document>
        </ns1:uploadFileToUcm>
    </soap:Body>
</soap:Envelope>
```

Figure 28: Sample payload for the Journal Import process

4. Prepare the payload to load data from the file on the UCM server to the GL_INTERFACE table using the Load Interface File for Import ESS process.

The following table lists the payload input parameters for the submitESSJobRequest operation:

Payload Parameter Name	Value	Comments
jobPackageName	oracle/apps/ess/financials/ commonModules/shared/common/ interfaceLoader	
jobDefinitionName	InterfaceLoaderController	
paramList	15	Value of the Import Journals from the list of values.
paramList	<372750>	Document identifier associated with the file uploaded to the UCM server. The response payload in step 2 returns the document identifier value.
paramList	N	
paramList	N	

Use the service tester to call the web service with the payload. Check the response for the request ID of the ESS process.

The following table lists the parameters for the submitESSJobRequest operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
jobPackageName	ESS job package name.	IN	Yes	java.lang.String
jobDefinitionName	ESS job definition name.	IN	Yes	java.lang.String
paramList	List of parameters used to call the ESS job. The order of the parameters is maintained as per the list. When a parameter isn't passed, the corresponding entry in the list must be passed as "#NULL".	IN	No	java.util.List<java.lang.String>
return	Request ID is returned	OUT		java.lang.Long

5. Prepare the payload to verify the status of the request.

The following table lists the payload input parameters for the getEssJobStatus operation:

Payload Parameter Name	Value	Comments
requestID	<3727>	The request identifier for the ESS process.

Use the service tester to call the web service with the payload. Check the status of the ESS request. When the data uploads:

- Successfully, the execution status is SUCCEEDED.

- Fails, the execution status is ERROR.

Operation: getEssJobStatus

The getEssJobStatus operation obtains the execution status of the submitted ESS job. This operation requires application user access.

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
requestID	The request ID of the ESS job.	IN	Yes	java.lang.Long
return	Returns the current status of the ESS job. For complete list of the job statuses and description, see Appendix 4: ESS Job Execution Status.	OUT		java.lang.String

The following sample payload illustrates the Journal Import process:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:getESSJobStatus
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/erpIntegrationService/types/">
            <ns1:requestId>35961</ns1:requestId>
        </ns1:getESSJobStatus>
    </soap:Body>
</soap:Envelope>
```

Figure 29: Sample payload for the Journal Import process

6.   Prepare the payload to import the journals from the interface table to GL_JE_HEADERS and GL_JE_LINES using the Journal Import ESS process.

To map the ESS parameters to the payload parameters:

   i.   Navigate to the Scheduled Processes page.

   ii.   Click the **Schedule New Process** button.

   iii.   Find and select the **Import Journals** process.

   iv.   Submit the process by entering the required parameters.

   v.   Go to the Process Monitor and select the submitted process.

   vi.   In the Details region, expand the **Parameters** node. Review the ordered list of arguments and their values, and find the arguments IDs that are passed internally.

The payload input parameters for the submitESSJobRequest operation of ERP Integration Service are as follows:

Payload Parameter Name	Value	Comments
jobPackageName	/oracle/apps/ess/financials/generalLedger/programs/common	
jobDefinitionName	JournalImportLauncher	
paramList	1061	Data access set identifier
paramList	Expenses	Source
paramList	1	Ledger identifier
paramList	ALL	Group identifier
paramList	N	Post account errors to suspense account
paramList	N	Create summary journals
paramList	N	Import descriptive flexfields

Use the service tester to call the web service with the payload. Check the response for the status of the ESS job request.

7. Prepare the payload to verify the status of the request.

    The following table lists the payload input parameters for the getEssJobStatus operation of the ERP Integration Service:

Payload Parameter Name	Value	Comments
requestID	<372750>	The request identifier for the ESS process.

    Use the service tester to call the web service with the payload. Check the response for the status of the ESS request.

8. Prepare the payload to retrieve the log or output file associated with the request.

    The following table lists the payload input parameters for the downloadESSJobExecutionDetails operation of the ERP Integration Service:

Payload Parameter Name	Value	Comments
requestID	<372750>	The request identifier for the ESS process.

    Use the service tester to call the web service with the payload. Check the response payload for this ESS request.

Response Payload

Run the Base64 decode sample program to convert the Content value from the response payload to 212913.zip file. For the sample program, see Appendix 2: Sample Code for Preparing a Data File for Inbound and Outbound Flow.

```
<env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="Body-M3Y1Wo0PrQ91vGVfDENtkQ22">
        <ns0:downloadESSJobExecutionDetailsResponse
xmlns:ns0="http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/types/">
            <ns2:result xmlns:ns0="http://xmlns.oracle.com/adf/svc/types/"
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/"
xmlns:ns2="http://xmlns.oracle.com/apps/financials/commonModules/shared/ErpIntegrationService/types/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns1:DocumentDetails">
            <ns1:Content>UEsDBBQACAAIAPK68UQAAAAAAAAAAAAAAAKAAAAMjEyOTEzLmxvZz3PwYrDIBCA4btPMQ8gOJoa
6N5CY6CnDd28gCSTRTCa6gS6b99Sy16/ww8/PWg+mL5+pu42iZ4qh+Q55ARDiAR9KDRzLn+gasys
aKtnja3y+779FtWN4/Q9qpAq+zSTolpVWRdltDlro/LB4kb341WFaw9vbf7lkhPTgz8sYfWxkoR0
xChhcK3u0Wrd2guesHF4aixaHBA76lonQRsJBo2Yik91pfLqbXskpkXQZ8qlRTwBUEsHCNkrs2Wz
AAAA4QAAAFBLAQIUABQACAAIAPK68UTZK7NlswAAAOEAAAAKAAAAAAAAAAAAAAAAAAAAAAyMTI5
MTMubG9nUEsFBgAAAAABAAEAOAAAAOsAAAAAA==</ns1:Content>
                <ns1:FileName xsi:nil="true"/>
                <ns1:ContentType>zip</ns1:ContentType>
                <ns1:DocumentTitle>ESS_L_212913</ns1:DocumentTitle>
                <ns1:DocumentAuthor>FIN_USER1</ns1:DocumentAuthor>
                <ns1:DocumentSecurityGroup>Attachments</ns1:DocumentSecurityGroup>
                <ns1:DocumentAccount xsi:nil="true"/>
                <ns1:DocumentName>212913.zip</ns1:DocumentName>
            </ns2:result>
        </ns0:downloadESSJobExecutionDetailsResponse>
    </env:Body>
```

Figure 30: Response payload for the downloadESSJobExecutionDetails operation

Appendix 10: Manage Outbound Flow Automation Steps with Separate Web Service Operations

This section describes an alternate way to automate outbound data integration using individual web service operation. The Extract Receivables Transactions flow illustrates the outbound data integration flow with Oracle ERP Cloud. In this example, a batch of transactions are extracted from the system and sent to the customers. The transactions are extracted in the output file of the Print Receivables Transaction ESS process.

Use the following operations to control the orchestration of the outbound flow:

1. submitESSJobRequest

2. getEssJobStatus

3. downloadESSJobExecutionDetails

Flow Steps Details

1. Prepare the payload to call the Print Receivables Transactions ESS process.
   To map the ESS parameters and the values for preparing the payload, see Step 5 in the inbound flow example.

   The payload input parameters for the submitESSJobRequest operation are as follows:

Payload Parameter Name	Value	Comments
jobPackageName	oracle/apps/ess/financials/receivables/transactions/shared	
jobDefinitionName	PrintReceivablesTransaction	
paramList	204	Business Unit Identifier
paramList	NEW	Transactions to Print
paramList	TRX_NUMBER	Order By
paramList	#NULL	Batch Name
paramList	#NULL	Transaction Class
paramList	#NULL	Transaction Type
paramList	#NULL	Customer Class
paramList	#NULL	From Customer
paramList	#NULL	To Customer
paramList	#NULL	From Customer Account Number
paramList	#NULL	To Customer Account Number
paramList	#NULL	From Transaction Number
paramList	#NULL	To Transaction Number
paramList	#NULL	Installment Number
paramList	#NULL	From Print Date

paramList	#NULL	To Print Date
paramList	#NULL	Open Invoices Only
paramList	Default Invoice Template	Invoice Template Name
paramList	Default Credit Memo Template	Credit Memo Template Name
paramList	Default Debit Memo Template	Debit Memo Template Name
paramList	Default Chargeback Template	Chargeback Template Name
paramList	No	Itemized Tax by Line

Important

You must pass "#NULL" in the payload for the null value in the request payload.

Use the service tester to call the web service with the payload. Check the response for the request ID of the ESS process.

2. Prepare the payload to verify the status of the request.

The following table lists the payload input parameter for the getEssJobStatus operation of the ERP Integration Service:

Payload Parameter Name	Value	Comments
requestID	<372750>	The request identifier for the ESS process.

Use the service tester to call the web service with the payload. Check the response for the status of the ESS request.

Operation: getEssJobStatus

The getEssJobStatus method obtains the execution status of the ESS job. This operation requires application user access.

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
requestID	The request ID of the ESS job.	IN	Yes	java.lang.Long
return	Returns the current status of the ESS job. For a complete list of job statuses and descriptions, see Appendix 4: ESS Job Execution Status.	OUT		java.lang.String

The following sample payload illustrates the Journal Import process:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
     <soap:Body>
        <ns1:getESSJobStatus
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/erpIntegrationService/types/">
             <ns1:requestId>35961</ns1:requestId>
        </ns1:getESSJobStatus>
     </soap:Body>
</soap:Envelope>
```

Figure 31: Sample payload for the Journal Import process

3. Prepare the payload to retrieve the log file or the output file associated with the request.

The following table lists the payload input parameter for the downloadESSJobExecutionDetails operation of the ERP Integration Service:

Payload Parameter Name	Value	Comments
requestID	<372750>	The request identifier for the ESS process.

Use the service tester to call the web service with the payload. Check the response for the log or output file associated with the ESS request. For more information on preparing the log or output file using the output from the response payload, see Appendix 2: Sample Code for Preparing a Data File for Inbound and Outbound Flow.

Appendix 11: Creating a Callback Web Service

In practice, customers will create and host a callback web service to optimally leverage the callback capabilities provided by Oracle ERP Integration Service for notification purposes. The customer callback web service must implement the onJobCompletion() operation. When a job completes, Oracle ERP Integration Service invokes the customer callback web service as defined in the request payload of supported operations with callback capabilities, such as the importBulkData operation.

Note

The customer callback web service triggers when the last job in the payload is executed, irrespective of whether the job completes successfully or fails.

Sample Callback Response

```
{
  "JOBS":
        [
           {"JOBNAME":"Load Interface File for Import",
            "JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
            "DOCUMENTNAME":"apinvoiceimport.zip",
            "REQUESTID":"2529",
            "STATUS":"SUCCEEDED",
            "CHILD":[
                       {"JOBNAME":"Transfer File",
                        "JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
                        "REQUESTID":"2530",
                        "STATUS":"SUCCEEDED"},
                       {"JOBNAME":"Load File to Interface",
                        "JOBPATH":"/oracle/apps/ess/financials/commonModules/shared/common/interfaceLoader",
                        "REQUESTID":"2531",
                        "STATUS":"SUCCEEDED"}
                    ]
           },
           {"JOBNAME":"Import Invoices",
            "JOBPATH":"/oracle/apps/ess/financials/payables/invoices/transactions",
            "REQUESTID":"2532",
            "STATUS":"SUCCEEDED",
            "CHILD":[
                       {"JOBNAME":"Import Invoices Report",
                        "JOBPATH":"/oracle/apps/ess/financials/payables/invoices/transactions",
                        "REQUESTID":"2533",
                        "STATUS":"SUCCEEDED"}
                    ]
           }

        ],
  "SUMMARYSTATUS":"SUCCEEDED"

}
```

Figure 32: Sample callback response

Callback Service Implementation Sample Code

The following example illustrates a sample Java code for retrieving the details from the response payload:

```java
public void onJobCompletion(OnJobCompletion params) {
    String strPayload = null;


    strPayload = params.getResultMessage();

    JSONObject jsonResponse = new JSONObject(strPayload);

    String xmlStr = XML.toString(jsonResponse);

  // Use the xml parser api to retrieve the value of the node

try {

        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

        InputSource is = new InputSource();

        is.setCharacterStream(new StringReader(xmlStr));

        Document doc = dBuilder.parse(is);

        System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

        if (doc.hasChildNodes()) {

                NodeList nodeList = doc.getChildNodes();

                for (int count = 0; count < nodeList.getLength(); count++) {

                        Node tempNode = nodeList.item(count);

                        if (tempNode.getNodeType() == Node.ELEMENT_NODE) {

                                if (tempNode.getNodeName().equals("DOCUMENTID")) {

                                        System.out.println("DOCUMENTID: " + tempNode.getTextContent());

                                }

                                if (tempNode.getNodeName().equals("SUMMARYSTATUS")) {

                                        System.out.println("SUMMARYSTATUS: " + tempNode.getTextContent());

                                }

                                if (tempNode.getNodeName().equals("JOBS") && tempNode.hasChildNodes()) {

                                        NodeList jobList = doc.getChildNodes();

                                        for (int c = 0; c < jobList.getLength(); c++) {

                                                Node job = jobList.item(c);

                                                printJobDetails(job);
```

```
                                                    //printing child

                                                    if (job.hasChildNodes()) {

                                                        NodeList nodeList2 = job.getChildNodes();

                                                        for (int count2 = 0; count2 < nodeList2.getLength(); count2++) {

                                                            Node tempNode2 = nodeList2.item(count2);

                                                            if (tempNode2.getNodeType() == Node.ELEMENT_NODE &&
tempNode2.hasChildNodes()) {

                                                                NodeList nodeList3 = tempNode2.getChildNodes();

                                                                for (int count3 = 0; count3 < nodeList3.getLength(); count3++) {

                                                                    printJobDetails(nodeList3.item(count3););

                                                                }

                                                            }

                                                        }

                                                    }

                                            }

                                    }

                            }

                    }

  } catch (Exception e) {

        e.printStackTrace();

}

private void printJobDetails(Node job) {

        if (job.hasChildNodes()) {

                NodeList nodeList = job.getChildNodes();

                for (int count = 0; count < nodeList.getLength(); count++) {

                        Node tempNode = nodeList.item(count);

                        if (tempNode.getNodeType() == Node.ELEMENT_NODE && !tempNode.hasChildNodes()) {

                                if (tempNode.getNodeName().equals("JOBNAME")) {

                                        System.out.println("JOBNAME: " + tempNode.getTextContent());
```

```
                }

                if (tempNode.getNodeName().equals("JOBPATH")) {

                        System.out.println("JOBPATH: " + tempNode.getTextContent());

                }

                if (tempNode.getNodeName().equals("STATUS")) {

                        System.out.println("STATUS       : " + tempNode.getTextContent());

                }

                if (tempNode.getNodeName().equals("REQUESTID")) {

                        System.out.println("REQUESTID: " + tempNode.getTextContent());

                }

                if (tempNode.getNodeName().equals("DOCUMENTNAME")) {

                        System.out.println("DOCUMENTNAME: " + tempNode.getTextContent());

                }

            }

        }

    }

}
```

Figure 33: Sample code for the callback service

Callback Web Service Security

If you have enabled SSL in your callback web service, you must use the CA certificate in SOA cloud service or ICS. The SSL certificate used in the callback API must be present in both Oracle ERP Cloud and SOA cloud service or ICS.

Note: Oracle ERP Cloud does not import the self-signed SSL certificate to Keystore. Where the callback web service is deployed, use the CA certificate in the PaaS application.

PaaS or On-Premise Security Configuration

Oracle ERP cloud implements Oracle Web Service Manager (OWSM) to secure web services. The security policy for callback is: oracle/wss_saml_token_bearer_client_polic. This mandates that the callback web service in your PaaS or on-premise implementation must be secured through compatible OWSM server policy: oracle/wss_saml_bearer_or_username_token_service_policy.

This is a sample SAML assertion from Oracle ERP Cloud when invoking the callback service:

```xml
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" env:mustUnderstand="1">

    <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" MajorVersion="1" MinorVersion="1" AssertionID="SAML-fCdgV0BqwQDU97xq6frjRw22" IssueInstant="2016-05-24T01:06:22Z" Issuer="www.oracle.com">

    <saml:Conditions NotBefore="2016-05-24T01:06:22Z" NotOnOrAfter="2016-05-24T01:11:22Z"/>

    <saml:AuthenticationStatement AuthenticationInstant="2016-05-24T01:06:22Z" AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">

      <saml:Subject>

        <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">FINUSER1</saml:NameIdentifier>

        <saml:SubjectConfirmation>

          <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</saml:ConfirmationMethod>

        </saml:SubjectConfirmation>

      </saml:Subject>

    </saml:AuthenticationStatement>

    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">

      <dsig:SignedInfo>

        <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

        <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>

        <dsig:Reference URI="#SAML-fCdgV0BqwQDU97xq6frjRw22">

          <dsig:Transforms>

            <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>

            <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

          </dsig:Transforms>

          <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

          <dsig:DigestValue>IqAsJkAm490B8xQLk3+ztDPLggw=</dsig:DigestValue>

        </dsig:Reference>

      </dsig:SignedInfo>
<dsig:SignatureValue>DqM35PHNh4RKvWiZ/QOBYieWl0a9sKk0eFYSISL2NRqK7gLNdH8mTUT3HlBl282b3FsJrEgK25cuV1f2suquyEzOeGUQTd71hue
W4xbXg5bjUNAQ+RXjdP3TUr8wy9+Ftv1s2tobZSXyfpxmHDDcsgOEP2MltkItjLKRyCDlHTrqUxBJDW+g5KvwgNhjadA/1ShlBeyA4uv2X3uxa/IyDGr5h82P
6v+jUIXtnDjZW0lDphdjZOXx+y943tYPx8SYDaIaf20lb19lHI10nwm76nzo7gU/MAxS2d5uf1YsbH15NN2tD7amwpSznRTt8OgTwgru27XcFVQ+0FGjA4O2
6Q==</dsig:SignatureValue>          <dsig:KeyInfo>

        <dsig:X509Data>
```

<dsig:X509Certificate>MIIDbDCCAlSgAwIBAgIGAVL8VT1fMA0GCSqGSIb3DQEBCwUAMHgxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdNeVN0YXRlMQ8
wDQYDVQQHEwZNeVRvd24xFzAVBgNVBAoTDk15T3JnYW5pemF0aW9uMRkwFwYDVQQLExBGT1IgVEVTVElORyBPTkxZMRIwEAYDVQQDEwlDZXJ0R
2VuQ0EwHhcNMTYwMjIwMDEzOTUwWhcNMjEwMjE4MDEzOTUwWjBTMRMwEQYKCZImiZPyLGQBGRYDY29tMRYwFAYKCZImiZPyLGQBGRYGb3JhY
2xlMRIwEAYKCK8tzPmS9ielPXDmLnIlaF+NvYf2YsQzurFxkzJ37PPeOpImr5bj83gyTKzsnRLETN6j1cQUtCARa+QPuBYw9c2Y4gkoFSyKBS3nRG5ulJmFafoB
3SBBfX93126V7rZBjZ8QrzsZhATTZQijs2a7s/X2hNTuGheIWqyerlUrek5PErkPY7eYFijmwn5pcSokUV10Py6dxw+A==</dsig:X509Certificate>

             <dsig:X509IssuerSerial>

              <dsig:X509IssuerName>CN=CertGenCA, OU=FOR TESTING ONLY, O=MyOrganization, L=MyTown, ST=MyState,
C=US</dsig:X509IssuerName>

              <dsig:X509SerialNumber>1455932390751</dsig:X509SerialNumber>

             </dsig:X509IssuerSerial>

             <dsig:X509SubjectName>CN=service, DC=us, DC=oracle, DC=com</dsig:X509SubjectName>

             <dsig:X509SKI>kFgAwGeiKu8lIiJe62UKOE2V0X8=</dsig:X509SKI>

           </dsig:X509Data>

         </dsig:KeyInfo>

       </dsig:Signature>

     </saml:Assertion>

Figure 34: Sample SAML assertion of the callback service

These are the sample steps in Weblogic to receive callback from the Oracle ERP Cloud:

1. The Weblogic instance must support issuer www.oracle.com and the LDAP directory must have the saml:NameIdentifier username. In this sample, it is FINUSER1.

2. The SAML assertion must also contain a digital signature that must be imported in the Weblogic OWSM keystore.

Appendix 12: Creating a Job Property File for the importBulkData Operation

You can generate the job property file for each job to further simplify the request payload of the importBulkData operation.

Generating Job Properties

The job properties file must be in CSV format. These are the following columns in the proper order:

1. Job package name

2. Job definition name

3. Inbound ZIP data file prefix name

4. ESS parameters (or arguments) – could be none or many as defined by job name

To get the job package and definition name, see Viewing Details about Predefined Scheduled Processes in the File-Based Data Import for Oracle Financials Cloud guide in the Oracle Help Center at http://docs.oracle.com.

This is an example of a journal import where the ZIP data file name is: my_journal_import_123456.zip.

/oracle/apps/ess/financials/generalLedger/programs/common,JournalImportLauncher,my_journal_import,#NULL,#NULL,Balance Transfer,#NULL,1,#NULL,N,N,N

The file name could be any name, but the extension must be ".properties".

Delivering Job Property File

Once the job property file is generated for the respective job, it can be bundled with the ZIP file that contains the data file(s). The job property file must have the extension ".properties".

You can reuse the same job property file for similar imports by uploading the file to the respective UCM account from the Oracle Fusion Applications UI, or using the uploadFileToUcm operation.

Reusing the Job Property File Naming Convention from the UCM Account

The job property file name must be defined with respective to ZIP file name. For example:

ZIP file name is "SS_GL_ERP_<SEQUENCE or TIMESTAMP>.zip

The parameter file name must be "SS_GL_ERP.properties".

Reusing the Job Property File – Custom Name from the UCM Account

A parameter file with a custom name can be uploaded to the respective UCM account. To reuse the parameter file, you must provide the file name in the request payload as follows:

`<ns1:jobOptions> `**`JobDetailFileName`**`=<fileName.properties></ns1:jobOptions>`

Appendix 13: Manual Inbound (Import) Steps

Transferring Data Files to Oracle WebCenter Content Server

After you generate the ZIP file that contains the CSV data import file, transfer the ZIP file to the Oracle WebCenter Content Server.

Use any of the following methods to transfer the file:

- File Import and Export page in Oracle Fusion Applications: Manual flow

- Oracle WebCenter Content Server home page

File Import and Export

Use the File Import and Export page to access the WebCenter Content Repository. For example, each Oracle ERP Cloud instance connects to a single Oracle WebCenter Content server for content management.

For more information about the use and administration of content management, see:

- Oracle WebCenter Content Server User's Guide

- Oracle WebCenter Content Server System Administrator's Guide

References for Using Content Management

The File-Based Data Import guides in the Oracle Help Center (http://docs.oracle.com) provide the objects to upload and download, including templates for external data integration. For general access to content management, including the metadata and manage accounts, use the Oracle WebCenter Content Server's standard service user interface.

The following table provides additional resources for more information:

Topic	Resource
Content server	Oracle WebCenter Content User's Guide for Content Server
Creating WebCenter content accounts	Oracle WebCenter Content System Administrator's Guide for Content Server
Naming accounts used in import and export processes	Files for Import and Export section
Programmatic upload and download to content management	Oracle WebCenter Content System Administrator's Guide for Content Server
Roles	Oracle Fusion Applications Common Security Reference Manual

Managing Files for Import and Export

Import data or export data out of Oracle ERP Cloud using content and processes repositories. Integration specialists stage data for import and export. Application administrators run processes to import data in the content repositories to the application transaction tables or retrieve data exported from the applications.

Managing files for import and export involve the following:

- Using the File Import and Export page

- Interacting with the content management server

Using the File Import and Export Page

The File Import and Export page enables you to upload or download content from the document repository of Oracle WebCenter Content Management. The search criteria section is limited to the minimum metadata of content management records needed for file import and export.

To access the File Import and Export page:

1. From the Navigator, click **Tools**.

2. Click **File Import and Export**.

The maximum file size using the File Import and Export page is 2 GB.

Interacting with Content Management

When you use the File Import and Export page, you are assigned one or more accounts in the content management server. These accounts organize and secure access to the content items.

Interaction between the File Import and Export page and Oracle WebCenter Content server requires securing content in an account. Oracle provides predefined accounts in the Oracle WebCenter Content server.

File import and export include the following concepts:

- Security

- Searching records

- Accessing content in a new account

- Account names

- Deleting files

- Uploading for import

- Downloading for export

- File size

Security

Use the File Import and Export Management duty role to access the File Import and Export page. This duty role is included in the predefined role hierarchy for integration specialist roles and product family administrator roles.

The files in Oracle WebCenter Content Management server are associated with an account. The users who have the access privileges to a specific account can work with content items that belong to that account.

> **Note**
>
> You can only upload and download files to and from the content management server that is associated with the accounts that you are entitled to access.

The Oracle WebCenter Content Management server does not support trailing slashes (/) as part of the account name. Account names are appended with a dollar sign ($) to ensure each account is unique. Account names are dynamic so that if they overlap (one name is completely contained in another, longer name, such as the US and US Sales), each account is considered as discrete by access grants. Security such as virus scanning is handled by the underlying integrated content management.

Searching Records

A record in the Oracle WebCenter Content Management server contains metadata used to access the file. When you run a scheduled process to completion on a file, the record for the file includes a process ID.

Accessing Content in a New Account

When you create a new account in the Oracle WebCenter Content Management server and the content server is not restarted, the access to the content in the new account from the File Import and Export page is delayed until the policy store is updated.

Account Names

If you create custom accounts for importing or exporting data, use the following naming conventions for the account:

- To avoid partial string matching, do not include a slash (/) at the beginning or use a dollar sign ($) at the end.

- Use a dollar sign and slash ($/) as a separator in the hierarchical structure.

The File Import and Export page transforms account names by removing the dollar signs ($). For example, fin$/journal$/import$ displays as fin/journal/import.

The Remote Intradoc Client (RIDC) HTTP Command-Line Interface (CLI) transforms the account name you specify without the dollar signs ($) to one that includes them. For example, fin/journal/import becomes fin$/journal$/import$ in the Oracle WebCenter Content Management server.

You must transfer files to these predefined accounts in content management that correspond to the interface table or assets of interest. For a list of the target accounts in each interface table, see Appendix 3: Predefined Target UCM Accounts.

Deleting Files

You can delete one file at a time in the File Import and Export page. To delete multiple files simultaneously from the content repository, use the standard service page in the Oracle WebCenter Content Management server.

Uploading for Import

To create a record, you must specify an account and the file. The account you specify determines which import process is used.

You can upload any file formats that can be parsed by the content repository, such as any MIME or content types. However, the format uploaded must conform to the requirements of the import process, such as a CSV file for the Load Interface File for Import process.

Downloading for Export

The export data processes create files in the content management server. Records in the search results table of the File Import and Export page provide links to the files for download.

Load Interface File for Import Process

The Load Interface File for Import process loads external setup or transaction data from the data file on the content management server to the relevant product interface table(s). You can run this process from the Scheduled Processes page on a recurring basis as needed.

Before running this process, you must:

1. Prepare your data file.

2. Transfer the data file to the content management server.

The following table describes the parameters for this process:

Parameter	Description
Import Process	Select the target import process.
Data File	Choose the data file from the choice list.

Importing Data

The final destination for your external data is the product application data table(s) of your Oracle ERP Cloud application.

Importing data into the application tables involves the following:

- Loading data into the product interface table(s)

- Finding and submitting the applicable import process

Loading Data into Interface Tables

Interface tables are intermediary tables that store your data temporarily while the application validates format and structure. Run the Load Interface File for Import scheduled process to load data from the data file into the interface table that corresponds to the template that you use to prepare the data.

To load your data into interface tables, submit the Load Interface File for Import scheduled process using the following steps:

1. From the Navigator, click **Tools**.

2. Click **Scheduled Processes**.

3. Click the **Schedule New Process** button.

4. Search and select the Load Interface File for Import job.

5. On the Process Details page:

    i. Select the target import process.

    ii. Enter the data file name.

6. Submit the process.

If the process is successful, the status is SUCCEEDED and the process populates the interface tables. If the process isn't successful, the status is ERROR. For more information on correcting load errors, see the Correcting Load Process Errors

The Load Interface File for Import process ends in error when the load of the data file fails for any individual row. The Load File to Interface child process ends as an error or warning. All rows that were loaded by the process are deleted and the entire batch of records is rejected.

Correcting Interface Data Errors

To correct errors:

6. Review the upload error logs.

7. Change any structural or formatting anomalies in the data.

8. Generate the ZIP file containing the CSV files using the respective import template.

9. Upload the corrected file to the UCM server and resubmit the Load Interface File for Import process.

10. Repeat these steps until the process successfully loads all the data.

Correcting Import Process Errors

If the import process fails with errors:

3. Review the errors in the import log.

4. Correct the error records using the applicable ADFdi correction spreadsheets.

For a list of import processes and their corresponding ADFdi correction spreadsheets, see Appendix 7: Error Handling for Import Jobs.

If auto purge is enabled in your import process, then you cannot use ADFdi. Use these steps:

4. Download the purge erroneous ZIP file from the File Import and Export page.

5. Select the erroneous data records from the interface file and correct them.

6. Follow the FBDI process to resubmit the corrected data.

Purging Interface and Error Tables

Data from the interface and error tables can be purged as part of the following processes:

3. Each File-Based Data Import (FBDI) process initiates a purge process by default. Following the completion of the import process, the erroneous data to be purged will first be extracted and uploaded to the Oracle WebCenter Content Server (UCM).

4. Customers also have the capability to manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed. This process supports the purge of interface data created from either FBDI or non-FBDI sources.

The purge backup file is stored and associated with the respective UCM import account for reference where needed. The file can either be downloaded using the Oracle ERP Integration Service or the File Import and Export page. This file is a consolidated ZIP file that contains the individual interface and error data files in a comma separated values (CSV) format.

For data correction, select and revise any erroneous data from the respective interface spreadsheet file, then upload the revised interface file again to execute the FBDI process.

For the processes outlined above, the existing inbound, outbound, and erroneous data files older than 30 days that are stored on the UCM server will automatically be purged for the applicable UCM account.

Operation: extractAndPurge

The extractAndPurge operation extracts data from the interface and error tables, uploads the relevant data file to UCM, then purges the respective data.

The purge file naming convention is as follows: ImportBulkData_<ImportJobName>_<LoadRequestId>.zip

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Request IDs	The request ID(s) of load jobs.	IN	Yes	java.lang.String
Notification Code	A two-digit number that determines how and when a notification is passed for the status of the import job. See the table below for the notification code values.	IN	Yes	java.lang.String
Callback URL	The callback URL of the web service you implemented to receive the ESS job status upon job completion.	IN	No	java.lang.String
Job Options	There are no additional job options for this operation.	IN	No	java.lang.String

The following table provides information on the notification codes:

Digit Position	Digit Value	Meaning
First digit	1	E-mail notification
	2	Bell notification
	3	Email and bell notification
Second digit	0	Send in any case (import failed or succeeded)
	1	Send on import success
	2	Send on import failure

The following sample request payload illustrates the extractAndPurge process:

```
<soap:Body>
  <ns1:extractAndPurge
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
      <ns1:requestIds>1234;1235;1236</ns1:requestIds>
      <ns1:notificationCode>30</ns1:notificationCode>
      <ns1:callbackURL>#NULL</ns1:callbackURL>
      <ns1:jobOptions></ns1:jobOptions>
</ns1:extractAndPurge>
</soap:Body>
```

Figure 11: Sample request payload for the extractAndPurge operation

To manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed, see Appendix 17: Purge - UI Based Approach.

 section.

Note

The data file remains in the content repository after the process ends.

Finding and Submitting the Import Process

To import your data into the application tables:

1. From the Navigator, click **Tools**.

2. Click **Scheduled Processes**.

3. Click the **Schedule New Process** button.

4. Search and select the import process for the target application tables.

5. On the Process Details page, select the process that corresponds to the data that you're importing. For example, **Journal Import**. If you prepared your data using the spreadsheet template, select the process shown in the Overview section of the spreadsheet.

6. Submit the process.

If the process is successful, the status is SUCCEEDED. The data in the interface tables is validated and the successful records are imported into the relevant Oracle ERP Cloud main application table(s). If the process isn't successful, the status is ERROR. For more information on correcting import errors, see the Correcting Load Process Errors

The Load Interface File for Import process ends in error when the load of the data file fails for any individual row. The Load File to Interface child process ends as an error or warning. All rows that were loaded by the process are deleted and the entire batch of records is rejected.

Correcting Interface Data Errors

To correct errors:

11. Review the upload error logs.

12. Change any structural or formatting anomalies in the data.

13. Generate the ZIP file containing the CSV files using the respective import template.

14. Upload the corrected file to the UCM server and resubmit the Load Interface File for Import process.

15. Repeat these steps until the process successfully loads all the data.

Correcting Import Process Errors

If the import process fails with errors:

5. Review the errors in the import log.

6. Correct the error records using the applicable ADFdi correction spreadsheets.

For a list of import processes and their corresponding ADFdi correction spreadsheets, see Appendix 7: Error Handling for Import Jobs.

If auto purge is enabled in your import process, then you cannot use ADFdi. Use these steps:

7. Download the purge erroneous ZIP file from the File Import and Export page.

8. Select the erroneous data records from the interface file and correct them.

9. Follow the FBDI process to resubmit the corrected data.

Purging Interface and Error Tables

Data from the interface and error tables can be purged as part of the following processes:

5. Each File-Based Data Import (FBDI) process initiates a purge process by default. Following the completion of the import process, the erroneous data to be purged will first be extracted and uploaded to the Oracle WebCenter Content Server (UCM).

6. Customers also have the capability to manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed. This process supports the purge of interface data created from either FBDI or non-FBDI sources.

The purge backup file is stored and associated with the respective UCM import account for reference where needed. The file can either be downloaded using the Oracle ERP Integration Service or the File Import and Export page. This file is a consolidated ZIP file that contains the individual interface and error data files in a comma separated values (CSV) format.

For data correction, select and revise any erroneous data from the respective interface spreadsheet file, then upload the revised interface file again to execute the FBDI process.

For the processes outlined above, the existing inbound, outbound, and erroneous data files older than 30 days that are stored on the UCM server will automatically be purged for the applicable UCM account.

Operation: extractAndPurge

The extractAndPurge operation extracts data from the interface and error tables, uploads the relevant data file to UCM, then purges the respective data.

The purge file naming convention is as follows: ImportBulkData_<ImportJobName>_<LoadRequestId>.zip

The following table lists the parameters for this operation:

Parameter Name	Description	Parameter (In/Out)	Mandatory	Type
Request IDs	The request ID(s) of load jobs.	IN	Yes	java.lang.String
Notification Code	A two-digit number that determines how and when a notification is passed for the status of the import job. See the table below for the notification code values.	IN	Yes	java.lang.String
Callback URL	The callback URL of the web service you implemented to receive the ESS job status upon job completion.	IN	No	java.lang.String
Job Options	There are no additional job options for this operation.	IN	No	java.lang.String

The following table provides information on the notification codes:

Digit Position	Digit Value	Meaning
First digit	1	E-mail notification
	2	Bell notification
	3	Email and bell notification
Second digit	0	Send in any case (import failed or succeeded)
	1	Send on import success
	2	Send on import failure

The following sample request payload illustrates the extractAndPurge process:

```
<soap:Body>
  <ns1:extractAndPurge
xmlns:ns1="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/"
>
      <ns1:requestIds>1234;1235;1236</ns1:requestIds>
      <ns1:notificationCode>30</ns1:notificationCode>
      <ns1:callbackURL>#NULL</ns1:callbackURL>
      <ns1:jobOptions></ns1:jobOptions>
</ns1:extractAndPurge>
</soap:Body>
```

Figure 11: Sample request payload for the extractAndPurge operation

To manage the purge process directly from the Scheduled Processes page by launching the Purge Interface Tables process as needed, see Appendix 17: Purge - UI Based Approach.

section.

Note

For more information on the process used for data prepared using the spreadsheet template, see the Instructions and CSV Generation tab of the spreadsheet template.

Appendix 14: Managing PGP Encryption Keys

Managing PGP Certificates

Certificates establish keys for the encryption and decryption of data that Oracle Cloud applications exchange with other applications. The Oracle Fusion Applications Security Console is an easy-to-use administrative interface that you access by selecting **Tools → Security Console** on the home page or from the Navigator. Use the Certificates page in the Security Console functional area to work with PGP certificates.

A PGP certificate consists of a public key and a private key. The Certificates page displays one record for each certificate. Each record reports these values:

- Type: For a PGP certificate, **Public Key** is the only type.

- Private Key: A check mark indicates that the certificate's private key is present. For either certificate format, the private key is present for your own certificates (those you generate in the Security Console). The private key is absent when a certificate belongs to an external source and you import it via the Security Console.

- Status: For a PGP certificate, the only value is **Not Applicable**. (A PGP certificate has no status.)

- For each certificate, click the button at the end of the row to display a menu of actions appropriate for the certificate. Alternatively, to view details for a certificate, select its name ("alias"). Actions include:

    o Generating certificates

    o Importing and exporting PGP certificates

    o Deleting certificates

Generating Certificates

For a PGP certificate, one operation creates both the public and private keys. From the Certificates page, select the **Generate** option. In the Generate page, select the certificate format **PGP**, and enter values appropriate for the format.

For a PGP certificate, these values include:

- An alias (name) and passphrase to identify the certificate uniquely.

- The algorithm by which the keys are generated, DSA or RSA.

- A key length – select **1024**.
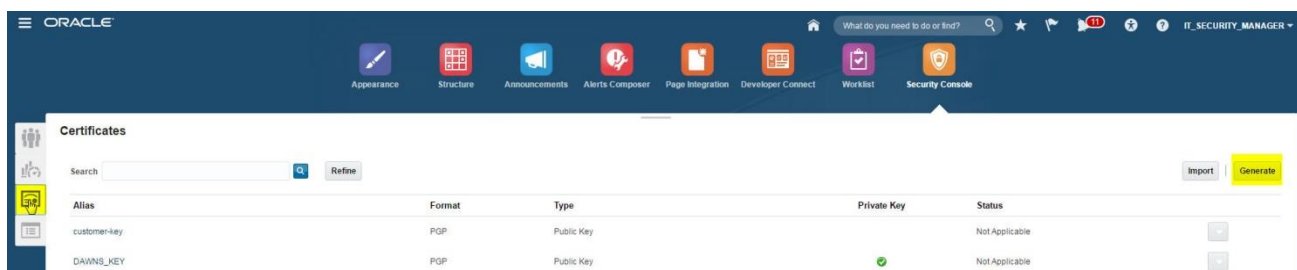
Figure 35: Security Console Certificates main page

Figure 36: PGP Generate Certificates dialog window

Importing and Exporting PGP Certificates

For a PGP certificate, you export the public and private keys in separate operations. You can import only public keys. The assumption is that you will import keys from external sources, which will not provide their private keys to you.

To export an Oracle Fusion public key:

1. From the Certificates page, select the menu in the row of the certificate that you want to export. Alternatively, open the details page for that certificate and select its **Actions** menu.

2. In either menu, select **Export**, then **Public Key** or **Private Key**.

3. If you selected **Private Key**, provide its passphrase. (The public key does not require one.)

4. Select a location for the export file. By default, this file is called [alias]_pub.asc or [alias]_priv.asc

To import a Customer's PGP public key:

1. On the Certificates page, click the **Import** button.

2. In the Import page, select **PGP** and specify an alias (which does not need to match the alias of the file you are importing).

3. Browse for the public key file, and then select **Import and Close**.

The Certificates page displays a record for the imported certificate, with the **Private Key** cell unchecked.
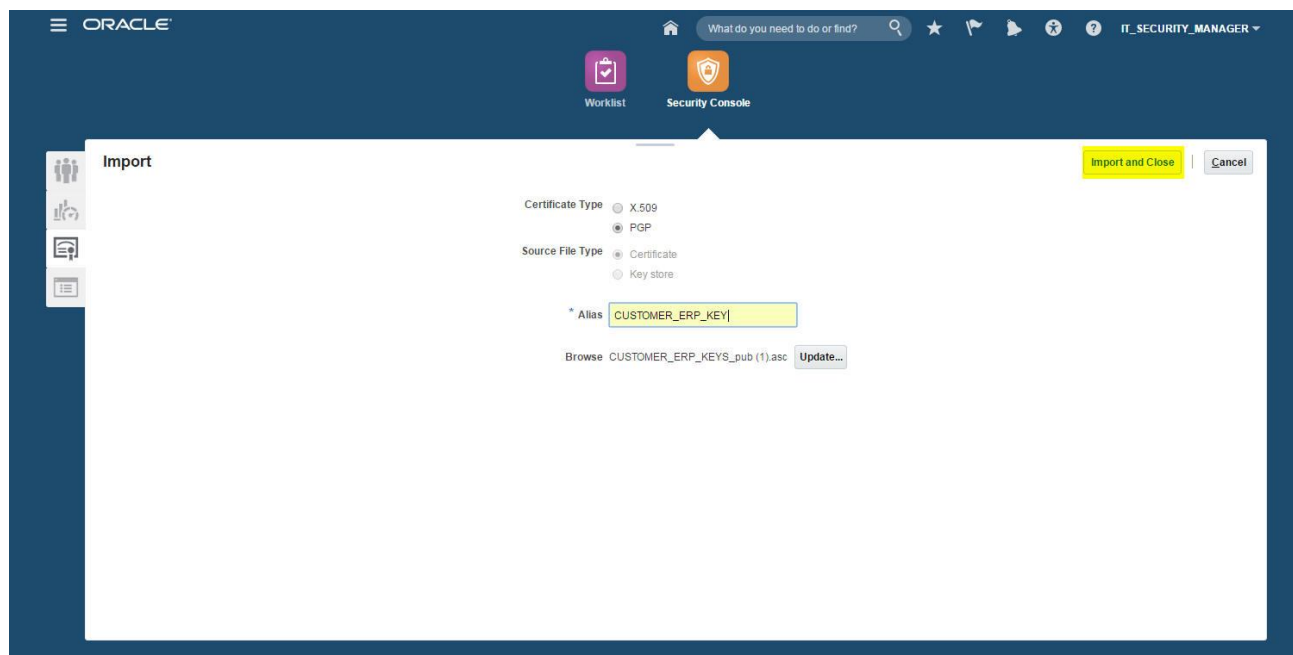
Figure 37: PGP Import Certificate page

Deleting Certificates

You can delete PGP certificates.

In the Certificates page, select the menu in the row of the certificate that you want to delete. Alternatively, select the **Actions** menu in the details page for that certificate. In either menu, select **Delete**, then review and accept the resulting warning message as appropriate.

Appendix 15: How to Encrypt and Decrypt a Data File

Encrypt an Inbound Data File from your Linux On-Premise System

In Linux, you can use "gpg" to encrypt a data file. After PGP keys are generated, you must import the Oracle ERP Cloud public key as follows:

```
gpg --import <MY_ERP_KEY_pub.asc>

###Verify the imported key using this command
gpg --list-keys
```

Figure 38: Sample command to import PGP public key

Once the public key is imported, use the following command to encrypt your inbound data file:

```
gpg --cipher-algo=AES -r=<alias> --encrypt <my data file>.zip
```

Figure 39: Sample command to encrypt ZIP file

The encrypted file will be renamed as <my_data_file>.zip.gpg.

Decrypt an Outbound Oracle ERP Cloud Data File in your Linux On-Premise System

To decrypt an outbound Oracle ERP Cloud data file, you must first import a customer's private key as follows:

```
gpg --allow-secret-key-import --import <my private.asc>

###Verify the imported key using this command
gpg --list-keys
```

Figure 40: Sample command to import PGP private key

Once a customer's private key is imported, use the following command to decrypt your outbound data file:

```
gpg --decrypt <EncryptedFileName> > <DecryptedFileName>
```

Figure 41: Sample command to decrypt file

Appendix 16: Large File Optimization (MTOM) Proxy Client Code Changes

The following example illustrates sample Java code changes for MTOM attachment:

```java
//Example for uploadToFileUcm – similar process for importBulkData

/* Extend your code for MTOM and the <content> will be generated as follows:

<Content xmlns="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" xmime:contentType="application/octet-stream"<xop:Include
xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:6098af03-52ff-4a40-9665-c89085ac2f3a"></Content>

*/

    private static BinaryElementPathInfo prepareAttachmentInfo(){

        Map<String, List<String>> xpaths= new HashMap<String, List<String>>();

        List<String> paths = new ArrayList<String>();

        paths.add("//typ:uploadFileToUcm/typ:Document/erp:Content");

        paths.add("//typ:uploadFileToUcm/typ:Document/erp:Content/xop:Include");

        xpaths.put("GenericRequest", paths);

        Map<String,String> prefixes = new HashMap<String, String>();

        prefixes.put("typ", TYP_NAMESPACE);

        prefixes.put("erp", ERP_NAMESPACE);

        prefixes.put("xop", XOP_NAMESPACE);

        BinaryElementPathInfo elemInfo = new BinaryElementPathInfo(xpaths,prefixes);

        return elemInfo;

    }
private static String ERP_NAMESPACE="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/";

private static String XOP_NAMESPACE="http://www.w3.org/2004/08/xop/include";

 private static String TYP_NAMESPACE="http://xmlns.oracle.com/apps/financials/commonModules/shared/model/erpIntegrationService/types/";

public static void setupMtomAttachments(SOAPMessage requestMessage, BinaryElementPathInfo binaryElementPathInfo

            ) throws SOAPException {

if (requestMessage == null)

            return;

if (binaryElementPathInfo == null || binaryElementPathInfo.getXpathExpressions() == null ||

            binaryElementPathInfo.getXpathExpressions().isEmpty())

            return;

SOAPBody body = requestMessage.getSOAPBody();
```

```java
List<Element> binElemList = getAllBinaryElements(body, binaryElementPathInfo);

if (binElemList == null)

        return;

for (Element theElm : binElemList) {

        final OracleSOAPElement theSOAPElem;

        if (theElm instanceof OracleSOAPElement)

                theSOAPElem = (OracleSOAPElement)theElm;

        else {

                WsMetaFactory jrfMetaFactory = WsMetaFactory.newInstance(ImplType.JRF);

                SOAPFactory fact = jrfMetaFactory.createSOAPFactory();

                theSOAPElem = (OracleSOAPElement)fact.createElement(theElm);

                theElm.getParentNode().replaceChild(theSOAPElem, theElm);

                        }

        theSOAPElem.setDataHandler(new DataHandler(new DataSource() {

        public InputStream getInputStream() throws IOException {

        {

                java.io.InputStream is = new FileInputStream(new File("C:\\temp\\test.zip"));

                return is;

        }

        }

        public OutputStream getOutputStream() throws IOException {

                throw new IOException();

        }

        public String getContentType() {

                return "application/octet-stream";

        }

        public String getName() {

                return theSOAPElem.getLocalName();

        }
```

```
}));

        } //for every binary element

                requestMessage.setProperty(MessageImpl.PACKAGING_STYLE, MessageImpl.MTOM);

                requestMessage.setProperty(MessageImpl.ATTACHMENT_STYLE_PACKAGING,"true");

                requestMessage.setProperty(OracleSOAPMessage.MTOM_THRESHOLD, 1024);

        }
```

Figure 42: Sample code snippet to enable MTOM

Appendix 17: Purge - UI Based Approach

In addition to the automated approach, interface and error data can also be purged through the Oracle ERP Cloud user interface. These are the steps to execute the purge process through the user interface:

1. From the Oracle ERP Cloud home page, select **Tools → Scheduled Processes** from the Navigator.

2. Click **Schedule New Process.**

3. Search for and select the **Purge Interface Tables** process name.

4. Enter the data as defined in the table below.

Parameter Name	Description	Mandatory
Purge Process Intent	There are three options: 1. File-based data import – Purge FBDI data. 2. Maintenance – Extend the number of days for the auto purge process to extract data and upload to UCM. The default setting is 30 days. 3. Other - Purge Non-FBDI data.	Yes
Import Process	Select the applicable import process name.	Yes
Load Request ID	Enter the Load Request ID or enable an ID range.	Yes
Extract Data	Yes/No. The default value is No. If the value is Yes, the process will extract all the data being selected for purge and upload the data to UCM for future audit reference.	No

Purge FBDI Object Data using a Single Load Request ID

Figure 43: Sample of a purge using a load request ID

Purge FBDI Object Data using a Range of Load Request IDs

Figure 44: Sample of a purge using a range of load request IDs

Purging Non-FBDI Data

Non-FBDI data can be purged by selecting the Purge Process Intent as **Other** and providing the Import Request ID.

Figure 45: Sample of purging non-FBDI data

Hardware and Software, **Engineered to Work Together**