IFS Aviation Maintenance

Time Sheet API Guide

ICAM 26.2



Limited Availability Release

Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

1 Introduction....4
About CAMO Integrations....4
The CAMO Time Sheet API....5

2 API scenarios....6
Time sheet querying scenarios....6
Requesting completed work hours....6
Time sheet operational scenarios....7
Updating completed work hours....7

3 Setting up the Time Sheet API....8
License requirement....8

4 Operational service messages....9
update-completed-work-hours-request-1.0....9
Sample messages....10
Message contents....10
Mandatory elements....10
Errors and exceptions....10
Mapping elements to CAMO....11
update-completed-work-hours-response-1.0....11

5 Querying service messages....12
get-completed-work-hours-request-1.1....12
Sample messages....12
Message contents....12
get-completed-work-hours-response-1.1....13
Sample messages....13
Message contents....13
Mapping elements to the CAMO GUI....13

6 Integration framework....16
Operational and querying service messages....16
SOAP request message structure....18

Limited Availability Release

SOAP response message structure..... 19
Notification service messages..... 21
Configuring Maintenix for the external system's web service.....22
SOAP request message structure - notification service.....24
SOAP response message structure - notifications.....26
Setting up the IFS Maintenix Message Ordering Processor..... 26
Message streams.....26
Adding new message streams..... 26
Configuring message order timing.....27
Sequencing inbound messages.....28
Managing integration messages in Maintenix..... 29
Viewing integration messages..... 29
Sending messages manually within Maintenix.....31
Send message manually.....32
Edit and resend an integration message.....33
Understanding message errors.....33
Archive integration messages.....34
Logging.....34

Limited Availability Release

1 Introduction

About CAMO Integrations

CAMO Application Programming Interfaces (APIs) support logical integrations with external business applications. You can also use CAMO APIs for partial implementations, where your organization uses CAMO in conjunction with an existing Enterprise Resource Planning (ERP) application, or for phased implementations, where automation supported by the existing ERP is transferred to CAMO over time.

APIs simplify the task of integration, but do not deal with error handling between the integration layer of CAMO and the external system. Messages are designed to describe the data in a flexible way, but message invocation is left open to customization. Timing considerations are particularly important when dealing with Web services that support a request-based integration as opposed to trigger-based integration.

There are three types of messaging scenarios available with the CAMO APIs:

1. *Operational Service Messaging Scenario*: In this scenario, the external system sends an inbound message to CAMO containing the information that CAMO needs to function correctly. Upon the successful receipt of a message, CAMO sends an outbound response message for confirmation.
2. *Querying Service Messaging Scenario:* In this scenario, the external system sends an inbound message to query information from CAMO. CAMO returns an outbound message containing the requested information.
3. *Notification Service Messaging Scenario*: In this scenario, CAMO sends an outbound message containing specific information to an external system upon established events in CAMO.

Limited Availability Release

Note Not all APIs support all messaging scenarios.

Depending on the integration, there are different framework options.

Older adapter-type integrations can be configured to use a SOAP web service or JMS framework, with or without using the IFS Maintenix Integration Framework as a gateway in the integration architecture.

The IFS Maintenix Integration Framework is not required to use any integration, but it is an option for a number of APIs.

Some newer integrations are accessed by sending JSON requests to a publicly exposed HTTP endpoint.

The CAMO Time Sheet API

You can use the Time Sheet API to facilitate integrations with an external time sheet system. The Time Sheet API contains two sets of messages: one set of querying messages to retrieve labor information from CAMO, and one set of operational messages to update completed labor information in CAMO.

Use the querying message service to retrieve information about labor recorded against a task. This information can form the basis of a person's time sheet in an external system. CAMO provides the following information about each labor record:

- A short description of the task or fault.
- The account code associated with that task or fault.
- The number of hours that the person worked.

An identification key that you can use to update the labor information. You can use the operational message service to correct labor records in CAMO. When using an external time sheet system, a person may correct their hours worked in the external system. You must pass that information back to CAMO to ensure the that the labor records remain synchronized in both systems.

Limited Availability Release

2

API scenarios

This section describes the scenarios that are supported by the API.

Each of the supported scenarios contain the following sections:

- **Goal**: A short statement that defines the purpose of the scenario.
- **Summary**: A more detailed description of the scenario with additional context for the integration.
- **Preconditions**: A list of conditions that must be met before the scenario can be implemented.
- **Messaging**: A basic description of the messaging required for the scenario.
- **Result**: The outcome of the successful execution of the scenario.

Time sheet querying scenarios

The Time Sheet API supports the following querying service scenarios.

Requesting completed work hours

Goal: To obtain information about labor completed in CAMO.

Summary: To integrate with an external time sheet system, an external system sends a message to CAMO, requesting the details of all labor rows completed within a specified period of time. The list of completed labor rows includes the labor records that have been stopped, and separate records for inspection and certification labors.

Preconditions:

- The start time is provided.
- The end time is provided.

Limited Availability Release

Messaging: The external system sends an inbound *get-completed-work-hours-request-1.1* message. CAMO returns a *get-completed-work-hours-response-1.1* outbound message.

Result: CAMO returns all completed labor rows.

Time sheet operational scenarios

The Time Sheet API supports the following operational service scenarios.

Updating completed work hours

Goal: To update the start time, end time, and actual hours of a completed labor row in CAMO.

Summary: A completed labor row in CAMO is incorrect or out-of-date because it has been altered in an external system. The external system sends a message to CAMO to update the actual hours, start date, and end date of the labor row in CAMO.

Preconditions:

- The specified labor is already completed in CAMO.

Messaging: The external system sends an inbound *update-completed-work-hours-request-1.0* message. CAMO returns an update-completed-work-hoursresponse- 1.0 (p.31) outbound message.

Result: The labor is updated according to the details provided in the message.

Limited Availability Release

3

Setting up the Time Sheet API

The following chapter discusses setup and configuration issues for the Time Sheet API.

License requirement

To use the Time Sheet API, you must have the TIMESHEET_ADAPTER feature name added to your CAMO `license.xml` file. Contact your CAMO representative if you have any questions about licensing.

Limited Availability Release

4

Operational service messages

This section provides detailed information for each of the API operational messages. For each inbound message, the following information is provided:

- **Sample Messages:** A description of each of the sample messages that complete the supported scenarios.
- **Message Contents:** Information to help you understand how to use the message.
- **Mandatory Elements:** A table providing information about which elements are mandatory or data-specific in each of the supported scenarios. The following keys are used:
- • **M** indicates that the element is mandatory.
  - **D** indicates that the element, if provided, depends on the existence of corresponding data or reference terms in CAMO.
  - **U** indicates that the element, if provided, must be unique within CAMO
- **Errors and Exceptions:** A list of errors and exceptions that can be encountered when sending messages to CAMO.
- **Mapping Elements to the CAMO GUI:** A table providing the corresponding field in CAMO for each of the elements contained in the message.

For each outbound message, a brief explanation is provided. Response messages are generally reduced to a few key elements that confirm the details of the successful inbound message. For more specific information about each of the response messages, review the XML schemas.

update-completed-work-hours-request-1.0

The `update-completed-work-hours-request` inbound message is sent by external systems to update labor rows in CAMO.

The `update-completed-work-hours-request-1.0.xsd` schema contains annotations describing each element.

Limited Availability Release

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Updating_Completed_Work_Hours.xml`: This message corresponds to the *Updating completed work hours* scenario. In this message, the details of an update to a labor row are passed to CAMO.

Message contents

Use the `update-completed-work-hours-request` inbound message to update the actual hours, start date, and end date for a completed labor row in CAMO. To identify the appropriate labor row for update, you must provide the labor ID. The labor ID is a unique identifier that is assigned to the labor when the labor information is returned to the external system in the `get-completed-work-hours-response` outbound message.

In the `update-completed-work-hours-request` message, dates are specified as YYYY-MMDDThh: mm:ssTZD (e.g 2013-04-05T00:00:00).

Mandatory elements

The following table displays the mandatory elements for the `update-completed-work-hours-request` message in each supported scenario.

The following keys are used:

- **M** indicates that the element is mandatory.
- **D** indicates that the element, if provided, depends on the existence of corresponding data or reference terms in CAMO.
- **U** indicates that the element, if provided, must be unique within CAMO.

Element	Scenarios
	Updating completed work hours
<completed-work-hours> <labor-id>	M D
<completed-work-hours> <hours>	M
<completed-work-hours> <start-date>	M
<completed-work-hours> <end-date>	M

Errors and exceptions

The following is a list of errors and exceptions that can occur when processing a message:

- Invalid XML message formatting.
- Elements marked as mandatory are not provided.
- Elements that must correspond to existing data or reference terms in CAMO are invalid.

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the `update-completed-work-hours-request` message elements.

Element	Corresponding entry in CAMO GUI
<completed-work-hours> <labor-id>	N/A
<completed-work-hours> <hours>	Fault Definition Details page, History tab, Labor History group box, Actual Hrs column.
<completed-work-hours> <start-date>	Fault Definition Details page, History tab, Labor History group box, Start Time column.
<completed-work-hours> <end-date>	Fault Definition Details page, History tab, Labor History group box, End Time column.

update-completed-work-hours-response-1.0

The `update-completed-work-hours-response` outbound message is returned to the external system upon a successful `update-completed-work-hours-request` inbound message.

The `update-completed-work-hours-response-1.0.xsd` schema contains annotations describing each element.

Limited Availability Release

5

Querying service messages

The following chapter provides detailed information for each of the API querying messages. For each message, the following information is provided:

- A description of the message processing.
- A description of each of the sample messages that complete the supported scenarios.
- For the response messages, a table with the corresponding fields in CAMO for each element.

get-completed-work-hours-request-1.1

The get-completed-work-hours-request inbound message is sent by the external system. CAMO returns the requested information in a get-completed-work-hours-response outbound message.

The `get-completed-work-hours-request-1.1.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Requesting_Completed_Work_Hours.xml`: This message corresponds to the *Requesting completed work hours* scenario. In this message, the date range is specified for the labor information.

Message contents

The `get-completed-work-hours-request` message specifies the start and end dates, which form the date range for the request for completed labor information.

Limited Availability Release

get-completed-work-hours-response-1.1

CAMO sends a `get-completed-work-hours-response` outbound message following a `get-completed-work-hours-request` inbound message from the external system.

The `get-completed-work-hours-response-1.1.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Requesting_Completed_Work_Hours-Response.xml`: This message corresponds to the *Requesting completed work hours* scenario. In this message, the information requested in `Requesting_Upcoming_Part_Requests.xml` is returned to the external system.

Message contents

The `get-completed-work-hours-response` message returns the start and ends dates, as specified in the `get-completed-work-hours-request` inbound message, and the details of each completed (historic) labor row that was completed within the date range. The message also includes the labor ID, which you can use to update the labor row using the `update-completed-work-hours-request` inbound message.

In CAMO, jobs that are stopped are divided into two new labor rows: one historic labor row to capture the completed work, and one active labor row to capture the work that has not yet been performed. The `get-completed-work-hours-response` message returns only the historic labor row, with the job-stop code, and not the active labor row.

For labor rows that require inspection and/or certification, each are treated as separate labor records in the `get-completed-work-hours-response` message.

In the `update-completed-work-hours-request` message, dates are specified as YYYY-MMDDThh: mm:ssTZD (e.g 2013-04-05T00:00:00).

If an error occurs while processing an `update-completed-work-hours-request` message, the entire message is rejected, and the response message lists all errors encountered.

Mapping elements to the CAMO GUI

The following table displays the corresponding field in the CAMO user interface for each of the `get-completed-work-hours-response` message elements.

Limited Availability Release

Element	Corresponding entry in CAMO GUI
<start-date>	N/A.
<end-date>	N/A.
<completed-labor> <task-barcode>	Task Details page, Barcode field.
<completed-labor> <task-name>	Task Details page, page title.
<completed-labor> <task-code>	
<completed-labor> <account-code>	Task Details page, Task Information tab, Issue To Account field.
<completed-labor> <labor> <id>	N/A.
<completed-labor> <labor> <user> <username>	User Details page, Username field.
<completed-labor> <labor> <user> <hr-code>	User Details page, HR Code field.
<completed-labor> <labor> <skill>	Task Details page, Task Execution tab, Labor group box, Skill column.
<completed-labor> <labor> <role-type>	
<completed-labor> <labor> <hours>	Task Details page, History tab, Labor History group box, Actual Hrs column.
<completed-labor> <labor> <start-date>	Task Details page, History tab, Labor History group box, Start Time column.
<completed-labor> <labor> <end-date>	Task Details page, History tab, Labor History group box, End Time column.
<completed-labor> <labor> <job-stop-code>	Work Capture page, Reason field.
<completed-labor> <inventory-id> <part-no-oem>	Task Details page, Inventory field.
<completed-labor> <inventory-id> <manufacturer-code>	Inventory Details page, Manufacturer field.
<completed-labor> <inventory-id> <serial-number>	Task Details page, Inventory field.
<completed-labor> <aircraft-registration-code>	Task Details page, Aircraft field.

Limited Availability Release

Element	Corresponding entry in CAMO GUI
<completed-labor> <assembly-id>	Assembly Details page.
<completed-labor> <ata-chapter>	

Limited Availability Release

6

Integration framework

External system developers who are developing software to integrate their system with Maintenix, can use Maintenix Adapters. You have the option to use the adapters to integrate directly with Maintenix or to include the IFS Maintenix Integration Framework in the integration framework.

Maintenix Adapters use a Web service to integrate with an external system. The clients and servers exchange XML (Extensible Markup Language) messages that follow the SOAP (Service Oriented Architecture Protocol) version 1.1 standard.

The external system can access the Web services using SoapScope or another Web service tool.

You can find a copy of the WSDL (Web Services Description Language) request definition (Request.wsdl) in the Maintenix download directory, in the am-docs\Integrators and Developers\ Integration Resources folder, or at the following URL: `http://<username>:<password>@<servername>:7001/mxcorews/services/RequestWS?wsdl`

External systems can interact with Maintenix using the following methods:

- Operational Service Messages are used to send data to Maintenix. Maintenix processes the data, and then returns information about the success or failure of the message.
- Querying Service Messages are used to retrieve information from Maintenix.
- Notification Service Messages are generated by Maintenix, and sent to a Web service on the external system. These messages notify the external system when important events occur within Maintenix.

Operational and querying service messages

Use operational service messages when your external system must send information to Maintenix for processing. An example of this type of messaging is when an external system must tell Maintenix that a flight occurred or that a work package was completed.

Limited Availability Release

Use the querying service messages when your external system must query or extract information from Maintenix. An example of this type of messaging is when an external system needs to know the status of an aircraft in Maintenix, or when a finance system needs to know the list of financial transactions that have occurred in Maintenix.

All operational and querying service messages are invoked and handled in the same way. The external system calls the following generic HTTP Web service: `http://<username>:<password>@<servername>:<port>/mxcorews/services/RequestWS`

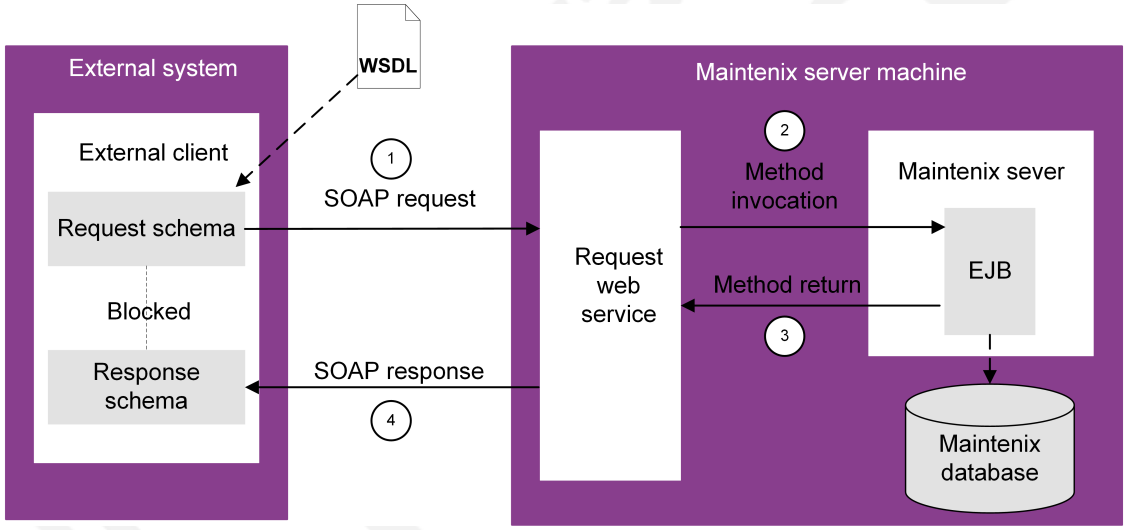
The same Web service is called, regardless of the data that is passed.

Maintenix then looks at the content of the HTTP request's <soap:Body> element to determine the logic that it should execute. Maintenix processes the data and returns an HTTP response where the <soap:Body> element describes whether or not the action was successful, and the details of any failure that might have occurred. As with all HTTP requests, the external client is blocked until it receives the response from the Web service.

See related topics on the format of the requests.

Integration scenario

The following diagram and integration scenario describe a typical Web service work flow for both the operational and querying service messages.



Operational and querying message web service

1. The client running on the external system composes a SOAP request message for the request Web service running on the Maintenix server machine. See example in the "SOAP request message structure" topic.

Note The thread in the external client is blocked until it receives the SOAP response.

2. The request Web service receives the SOAP request, and then invokes a method on an Enterprise Java Bean (EJB) in the Maintenix server.
3. The Maintenix server EJB processes the request, saves the results to the Maintenix database, and then returns the results to the request Web service.
4. The request Web service composes a SOAP response that contains the information requested by the external client (or a failure indication), and then sends it to the external client. The content of this request is specific to the event that occurred, and is documented in the associated Maintenix adapter guide. See example in the "SOAP response message structure" topic.

SOAP request message structure

The request Web service is the entry point into Maintenix for all operational and querying service messages.

Regardless of which adapter message is being called, external systems always call the single, generic SOAP-based "RequestWS" URL: `http://<username>:<password>@<servername>:<port>/mxcorews/services/RequestWS`

As specified in the Request.wsdl definition, the SOAP request message must include the <mode> element within the <Header> element of the message. The <mode> must be set to `sync`, which is used to indicate that the service must be processed, and that a response must be generated immediately.

You must also include a <request> element within the <Body> element of the SOAP message. Within the <request> element, you must enter XML-structured content that is based on the XSD of the adapter message that you want to invoke. Maintenix will look at the contents within the <request> element, and call the proper service based on the element name and namespace.

You can find the specific XSD files as well as sample messages for the content of the <request> element in the Maintenix download directory, `am-docs\Integrators and Developers` folder.

Note The sample messages included in the documentation folder do not contain the SOAP envelope.

The following is a sample SOAP request message that an external system might generate for an operational or querying message:

```
<?xml version='1.0' ?>
<soapenv:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
```

```
<soapenv:Header>
    <head:mode xmlns:head="http://xml.mxi.com/xsd/integration/
header/1.1">sync</head:mode>
</soapenv:Header>

<soapenv:Body>
    <req:request xmlns:req="http://xml.mxi.com/xsd/integration/
request/1.1">
        <!--The contents of the request message begins here.-->
        <!--The following example is taken from the aircraft-status-
request message-->
        <aircraft-status-request
          xmlns="http://xml.mxi.com/xsd/core/flight/aircraft-status-
request/1.0"
          xmlns:dt="http://xml.mxi.com/xsd/core/mxdomaintypes/2.0">
            <aircraft-identifier>
                <dt:barcode>I0005QVM</dt:barcode>
            </aircraft-identifier>
        </aircraft-status-request>
        <!--The contents of the request message ends here.-->
    </req:request>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP response message structure

Once the service is complete, Maintenix will return a SOAP-based XML message to the external system. This response message always includes a <response> element within the <Body> element of the SOAP message.

The <response> element has the following attributes:

- msg_id: The Maintenix-generated ID used to identify the particular message. It can be used for troubleshooting.
- msg_status: This is set to COMPLETE if the service was successful, or ERROR if it failed.

If the service is successful, then the <response> element contains an XML structure that is based on the XSD of the adapter message that was invoked. You can find the specific XSD files as well as sample messages for the content of the <request> element in the Maintenix download directory, am-docs\Integrators and Developers folder.

Note The sample messages included in the documentation folder do not contain the SOAP envelope.

The following is a sample SOAP response message that Maintenix might generate in response to an operational or querying request message:

```
<?xml version='1.0' ?>
<soapenv:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
```

```
   <soapenv:Body>
      <resp:response xmlns:resp="http://xml.mxi.com/xsd/integration/
response/1.1"
       msg_id="100024"
       msg_status="COMPLETE">
         <!--The contents of the response message begins here.-->
         <!--The following example is taken from the flight-
identifier-response message-->

         <flight-identifier-response
          xmlns="http://xml.mxi.com/xsd/core/flight/flight-identifier-
response/1.0">
            <external-key>0101BNAATL19990101</external-key>
            <flight-id>
               <flight-name>Duty</flight-name>
               <scheduled-departure-date>1999-01-01 03:38:00</
scheduled-departure-date>
               <departure-airport>ATL</departure-airport>
            </flight-id>
            <internal-identifier>4650:63699</internal-identifier>
         </flight-identifier-response>

         <!--The contents of the response message ends here.-->
      </resp:response>
   </soapenv:Body>

</soapenv:Envelope>
```

If the service is unsuccessful and there are errors, the <response> element contains an XML structure that describes the error. The XML message always includes an <errors> element within the <response> element. The <errors> element contains one or more <error> elements. Each <error> element describes the details of the failure that was encountered. Each <error> element contains the following child elements:

- <message>: A description of the problem that was encountered.
- <stacktrace>: A detailed trace of the exact line of code within Maintenix that generated the error. It is used for troubleshooting when communicating with your product support representative.

The following is a sample SOAP response message that Maintenix might generate in response to an unsuccessful operational or querying request message:

```
<?xml version='1.0' ?>
<soapenv:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

   <soapenv:Body>
      <resp:response
       msg_id="100031"
       msg_status="ERROR"
       xmlns:resp="http://xml.mxi.com/xsd/integration/response/1.1">
```

```
         <ns1:errors
          xmlns="http://xml.mxi.com/xsd/integration/error/1.1"
          xmlns:ns1="http://xml.mxi.com/xsd/integration/error/1.1"

 class="com.mxi.mx.core.adapter.flight.messages.flight.publish.
PublishingException"
          key="unknown"
          msg_id="100031">

            <ns1:error>
               <ns1:message>Unable to find valid inventory</
ns1:message>
               <ns1:stacktrace>
                  <!
[CDATA[<com.mxi.mx.core.adapter.flight.messages.flight.publish
                        .PublishingException: Unable to find valid
 inventory for aircraft
                        at
 com.mxi.mx.core.adapter.flight.messages.flight.publish

 .FlightPublishingService.generateAircraftStatus(Unknown Source)
                        at
 sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
                        ]]>

               </ns1:stacktrace>
            </ns1:error>
         </ns1:errors>
      </resp:response>
   </soapenv:Body>
</soapenv:Envelope>
```

Notification service messages

Notification service messages are used to alert an external system that something happened within Maintenix. For example an external system can be notified whenever a fault is raised, or a purchase order is updated.

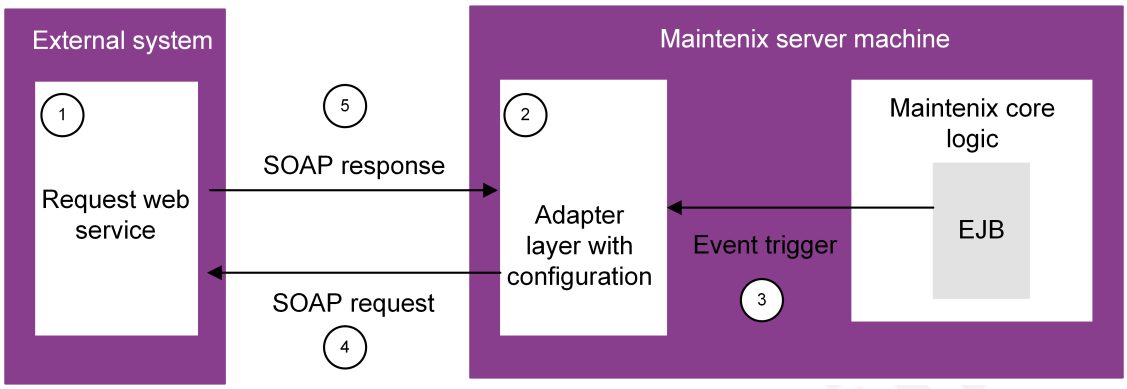
To set up the notification service messages, the external system must expose an HTTP Web service for Maintenix to call. You must also configure Maintenix, providing the external system's Web service details (URL, username, password, etc.), as well as the message that the external system is expecting (namespace, root_name). When the specified event occurs within Maintenix, Maintenix will generate a request to the external system's Web service, providing the message information.

The external system's Web service can provide a response to Maintenix's HTTP request. Maintenix will acknowledge the response message, but there is no logic that occurs within Maintenix, regardless of whether the external system processed the message successfully or not.

Limited Release

Integration scenario

The following diagram and integration scenario describe a typical notification message Web service workflow:



Notification message web service

1. A Web service is developed on the client that can listen to the HTTP requests generated by Maintenix. You can build this Web service using any tool or programming language, as long as it can listen to HTTP requests.
2. Maintenix is configured for the destination Web services on the external system.

> **Note** When configuring Maintenix, the NAMESPACE and ROOT_NAME depend on the event that is being monitored, and is documented in the associated Maintenix adapter guide.

3. The triggering event occurs within Maintenix.
4. Maintenix composes a SOAP request message for the external system's Web service. The content of this request is specific to the event that occurred, and is documented in the associated Maintenix adapter guide.
5. The external system provides a response message, which is acknowledged by Maintenix.

Configuring Maintenix for the external system's web service

To configure the notification service, you must add a new row into Maintenix's INT_DEST_LOOKUP table for each separate instance of a notification messaging service. In the INT_DEST_LOOKUP table, you specify the external system's Web service details (URL, username, password, etc.), as well as the message that the external system is expecting (namespace, root_name).

You have the option to create multiple rows for the same notification message service. This enables you to send multiple messages each time a single triggering event occurs. For example,

you can have multiple external systems (different URL/USERNAME/PASSWORD values), or you can receive different XSD versions for the same triggering event (different NAMESPACE values).

The following table describes each column that you must populate in the INT_DEST_LOOKUP table.

Column	Data Type	Mandatory	Description
SUBSCRIPTION_DB_ID	Integer	Yes	This is the database identifier of this entry. This should be set to the DB_ID provided at time of database creation.
SUBSCRIPTION_ID	Integer	Yes	This is a unique value used to identify this configuration entry.
NAMESPACE	Char(100)	Yes	This is the namespace of the notification message's XML Schema (XSD). It is used to identify which schema will be generated when the triggering event occurs. You can find the name of the notification message in the Notification Service Messages section of the Maintenix Adapter guide. For example: http://xml.mxi.com/xsd/core/ flight/fault-updated-event/1.0.
ROOT_NAME	Char(100)	Yes	This is the name of the notification message. It is used to identify which message the external system is listening for. You can find the name of the notification message in the Notification Service Messages section of the Maintenix Adapter guide. For example: fault-updated-event.
URL	Char(100)	Yes	This is the URL of the external system's Web service. For example: http:// www.client.com:8080/service.
USERNAME	Char(80)	No	This is the username that Maintenix uses to login to the external system's Web service.
PASSWORD	Char(100)	No	This is the password that Maintenix uses to login to the external system's Web service.
HIST_BOOL	1 or 0	No	This value is always set to 0.

Limited Availability Release

Column	Data Type	Mandatory	Description
STYLESHEET_PATH	Char(100)	No	This value is not needed and can be left empty.
DESTINATION	Char(20)	Yes	This value is always set to WS.

SOAP request message structure - notification service

When a triggering event occurs within Maintenix, Maintenix generates a SOAP-based XML message, and then sends it to the external system's Web service URL. The XML message always includes a <response> element within the <Body> element of the SOAP message.

The <response> element has the following attributes:

- msg_id: The Maintenix-generated ID used to identify the particular message. It can be used for troubleshooting.
- msg_status: This is set to COMPLETE

The <response> element contains an XML structure based on the XML Schema (XSD) of the notification message, as defined by the NAMESPACE specified in Maintenix's INT_DEST_LOOKUP table. You can find the specific XSD files, as well as sample messages for the content of the <response> element, in the Maintenix download directory, am-docs\Integrators and Developers folder.

Note The sample messages included in the documentation folder do not contain the SOAP envelope.

The following is a sample SOAP request message that Maintenix might generate for a notification message:

```
<?xml version='1.0' ?>
<soapenv:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

   <soapenv:Body>
      <resp:response xmlns:resp="http://xml.mxi.com/xsd/integration/
response/1.1"
       msg_id="100024"
       msg_status="COMPLETE">

         <!--The contents of the message begins here.-->
         <!--The following example is taken from the work-package-
updated-event message-->

         <work-package
```

```
          xmlns="http://xml.mxi.com/xsd/core/flight/work-package-
updated-event/1.0"
          xmlns:dt="http://xml.mxi.com/xsd/core/mxdomaintypes/2.0">
            <work-package-no>WO-T0000LDD</work-package-no>
            <name>A-CHECK (A Check)</name>
            <aircraft>
               <aircraft-identifier>
                  <dt:barcode>I0005QVM</dt:barcode>
                  <inventory-id>
                     <dt:oem-serial-number>203</dt:oem-serial-number>
                     <dt:oem-part-number>
                        <dt:oem-part-number>ACME505</dt:oem-part-
number>
                        <dt:manufacturer-id>
                           <dt:manufacturer-name>Acme</
dt:manufacturer-name>
                           <dt:manufacturer-code>ABI</dt:manufacturer-
code>
                        </dt:manufacturer-id>
                     </dt:oem-part-number>
                  </inventory-id>
                  <dt:aircraft-identifier>CFDRH</dt:aircraft-
identifier>
               </aircraft-identifier>
               <aircraft-status>
                  <capability />
                  <operating-status>In Repair</operating-status>
               </aircraft-status>
            </aircraft>
            <barcode>T0000LDD</barcode>
            <status />
            <work-type />
            <type-subtype>CHECK</type-subtype>
            <location>
               <airport-code>ORD</airport-code>
               <scheduled-location-code>ORD/HGR-2/TRK-4</scheduled-
location-code>
               <scheduled-location-name>Track-4</scheduled-location-
name>
               <scheduled-location-type>TRACK</scheduled-location-
type>
            </location>
            <scheduled-date>
               <start-date>2002-04-13 06:30:50</start-date>
               <end-date>2002-04-13 02:30:47</end-date>
            </scheduled-date>
            <actual-date />
            <external-vendor>
               <vendor-name />
               <vendor-code />
            </external-vendor>
            <delay />
         </work-package>

         <!--The contents of the message ends here.-->
```

```
      </resp:response>
   </soapenv:Body>

</soapenv:Envelope>
```

SOAP response message structure - notifications

The external system's Web service can provide a response to Maintenix's HTTP request. Maintenix will then acknowledge the response message, but there is no logic that occurs within Maintenix, regardless of whether the external system processed the message successfully or not.

The following is a sample SOAP response message that an external system might generate for a notification message:

```
<?xml version="1.0" encoding="UTF-8"?>
<en:Envelope xmlns:en="http://schemas.xmlsoap.org/soap/envelope/"/>
```

Setting up the IFS Maintenix Message Ordering Processor

The IFS Maintenix Message Ordering Processor provides message-ordering capabilities at the application layer. This means that applications which use CAMO APIs are responsible for proper message ordering.

> **Note** If the external system is unable to add sequence numbers to its messages, then message ordering is not possible using the IFS Maintenix Integration Framework.

Message streams

The IFS Maintenix Message Ordering Processor supports multiple message streams. A message stream is a logical sequence of messages which must be processed in a specified order. Messages in a particular stream need to be processed in relation to other messages in a stream, but not in relation to messages outside of the stream. Note that not all messages necessarily belong to a message stream.

Adding new message streams

Before ordered messages can be sent to a message stream in the Core Transaction Engine, a row describing the message stream must be added to the INT_MESSAGE_ORDER table. Create a row with the following information:

Parameters to be set in the INT_MESSAGE_ORDER

Column Name	Column Type	Description
stream_cd	VARCHAR2(16)	Primary key that uniquely represents the ordered message stream.
stream_sdesc	VARCHAR2(80)	Short description of the ordered message stream.
seq_number_last_processed	NUMBER(20)	Last sequence number that was processed in this ordered message stream. Initialize this value to 0
seq_number_next	NUMBER(20)	Next sequence number that should be processed in the ordered message stream. Initialize this value to 1

Configuring message order timing

In the UTL_CONFIG_PARM table, the MESSAGE_ORDER_SLEEP_TIME and MESSAGE_ORDER_TIMEOUT configuration parameters control the timing associated with message ordering.

Note The MESSAGE_ORDER_SLEEP_TIME and MESSAGE_ORDER_TIMEOUT configuration parameters shared by all message streams created in an instance of CAMO.

Configuration Parameters

Configuration Parameter	Configuration Type	Description
MESSAGE_ORDER_TIMEOUT	INTEGRATION	Maximum amount of time (in seconds) that the IFS Maintenix Message Ordering Processor will wait before skipping the next expected message.

Configuration Parameter	Configuration Type	Description
MESSAGE_ORDER_ SLEEP_TIME	INTEGRATION	The amount of time (in milliseconds) that the IFS Maintenix Message Ordering Processor will wait before repeating its cycle of acquiring a database lock.

The MESSAGE_ORDER_SLEEP_TIME configuration parameter allows the message ordering to be throttled as necessary. A higher sleep time value will reduce the frequency of queries to the database, but will also reduce the processing speed of ordered messages. The value should be tuned appropriately for the specific solution required by the customer.

If an ordered message is delayed in its delivery from an external system, processing of further messages from the ordered stream is halted until the missing message is delivered. However, without intervention, a message that is never delivered due to corruption or some other error emanating from the external system could permanently halt the stream. In order to avoid this, the MESSAGE_ORDER_TIMEOUT configuration parameter specifies a timeout interval for the missing message to arrive. This will ensure that message processing will never cease to function completely due to missing messages.

Note If an expected message appears only after the timeout has expired, and appears after another message that had a higher sequence number, then it will never be processed. If the message must still be processed in this scenario, it will have to be processed manually or resubmitted with a higher sequence number.

Sequencing inbound messages

In order for a message to be processed by the IFS Maintenix Message Ordering Processor, it needs to specify the message stream to which it belongs, as well as its sequence number in the stream.

The following variables should be included in the SOAP message or JMS properties:

Message Property	Property Type	Description
msg_stream	VARCHAR2(16)	The name of the message stream you would like to send the message to; stream_cd from the INT_MESSAGE_ORDER table

Message Property	Property Type	Description
msg_id	NUMBER(20)	The sequence number of the message; the message will be processed when this property matches seq_number_next in the INT_MESSAGE_ORDER table [minimum of 5 characters]

In addition, the following should be included as a JMS property:

Message Property	Property Type	Description
MX_SEQUENTIAL	Boolean	Indicates that the message is part of a message stream; should be set to `true` for all ordered messages

Managing integration messages in Maintenix

Maintenix supports the searching, viewing, and archiving of all Maintenix Adapter messages. You can also send individual messages through the Maintenix GUI to help test and debug your messaging integration.

Viewing integration messages

You can view all integration messages in the Maintenix GUI. Use the Message Search page to retrieve a list of messages. Use the Message Details page to view specific information about a message.

Use the Message Search page to return a list of messages. For each message that is returned in your search, Maintenix provides the following information:

Message element	Description
Queue ID	This number is generated by Maintenix.
Message ID	This number or code is the unique identifier of the message. The Message ID may be specified in the message itself, or else it is generated by Maintenix automatically. Click the Message ID hyperlink to go to the Message Details page, where you can find specific information about the message.

Limited Availability Release

Message element	Description
Stream	The code of the stream to which the message belongs (applicable to Message Ordering Framework). This is null if the message does not belong to a stream.
Status	The status of the message sent to Maintenix. The status can be one of the following: • AWAIT: The message is still processing. • ERROR: The message encountered an error while processing. • SUCCESS: The message was processed successfully.
Message Type	The type of integration framework message.
Start Time	The time that the message is received and processing begins.
Archived	If this check box is selected, the message is archived.

From the Message Search page, you can link to message details pages that provide additional information including the following:

Inbound tab

Item	Description
Delivery Code	The restrictions on message processing. The delivery code can be one of the following: • DEFAULT: There are no restrictions on message processing. • FIFO: The message must be processed in sequence by the order received. • ORDERED: The message must be processed through the Message Ordering Processor.
Message Source	The technology by which an external system sent the message to Maintenix. The message source code can be one of the following: • ws: The message was received through a web service. • jms: The message was received through a JMS queue.
Root Namespace	The namespace of the root element of the XML message.
Message	The contents of the XML message.

Limited Availability Release

Outbound tab

Item	Description
Root Name	The name of the root element of the XML message.
Root Namespace	The namespace of the root element of the XML message.
Message	The contents of the XML message.

Errors tab: The Errors tab appears for messages that contain errors.

Results tab:When a message is processed successfully, the Results tab lists the details for any of the following actions that are performed by Maintenix:

- Creating a work package.
- Updating a work package.
- Creating a task.
- Updating a task.
- Creating a fault.

Note Maintenix uses the Alerts framework to populate the Results tab. The Alert Engine must be active, and the EMA alert types must be enabled for the Results tab to be populated.

Notification tab: The Notification tab lists all notification log entries related to the integration message. All notifications sent to a subscriber through the event listener service are listed here.

Process tab: The Process tab lists all process log entries related to the integration message.

Sending messages manually within Maintenix

You can send integration messages to Maintenix manually through the Maintenix GUI as if they are being sent from an actual external system.

Use the Echo and Sleep functionalities when sending a message through the Maintenix GUI to test the framework. The Echo function returns a response immediately, while the Sleep function waits for the number of seconds specified in the message before returning a response.

Note "xmlns" is the default name space used by the parser to retrieve objects (elements) and values from the request message. If the request message contains elements that are defined in other name spaces, these XML elements must be prefixed with the name space. Otherwise, the parser cannot retrieve the element value correctly, and Maintenix business logic may fail.

Limited Availability Release

If the request message declares a namespace URL as xmlns:mxdomain="http://xml.mxi.com/xsd/core/mxdomaintypes/1.0, then elements defined in xmlns:mxdomain should be encoded as illustrated in the following example:

```
<mxdomain:contact>
    <mxdomain:username>jdoe</mxdomain:username>
</mxdomain:contact>
```

Send message manually

To test and debug your integration, you can send an XML message manually from within Maintenix. Maintenix processes the message and provides details, including any errors that occurred, on the Message Details page.

Prerequisite

You must have the ACTION_SEND_MESSAGE permission enabled.

Steps

1. Select the **Send Integration Message** menu item.
2. On the **Send Message** page, provide the following information:
   - Choose one of the following:
     - **Asynchronous**: If this check box is selected, the message will be processed asynchronously. Asynchronous operation mode is recommended for requests that may require more than 60 seconds of processing by the Maintenix server.
     - **Sequential**: If this check box is selected, the messages are processed in order, such that the processing of the next sequential message will occur only after the previous sequential message has been processed.
   - **Transport**: The manner in which the message will be sent to Maintenix. Choose one of the following:
     - Web Service: The message will be sent to the Maintenix request Web service.
     - Remote Method Invocation: The integration framework will be invoked directly from within Maintenix, bypassing the Web service; when asynchronous, the message will be placed directly on the inbound integration queue, and when synchronous, the message will be processed immediately.
   - **Base URL**: The URL that specifies the server from which the message will be sent; this field is read-only and will always identify the server to which you are currently logged in.
   - **Username**: Enter your username.
   - **Password**: Enter your password.
   - **Sample Message**: If you are testing the framework, click either the Echo button or the Sleep button to populate the Body field with a pre-formatted test message.

Limited Availability Release

- **Body**: Enter your XML message. For information on the message formatting and schema details, refer to the appropriate Maintenix API guide.

3. Click **OK**.

Edit and resend an integration message

If you encounter an error when manually sending an integration message into Maintenix, you can correct the error in your XML and resend it.

Prerequisite

You must have the ACTION_RESEND_MESSAGE permission.

Steps

1. Go to the **Message Details** page.
2. Click **Edit and Resend**.
3. On the **Send Message** page, edit the XML message.
4. Click **OK**.

Understanding message errors

When a message is sent, the Message Details page returns a list of any errors on the Error tab. You can use the error details to troubleshoot the error.

Maintenix tracks the following types of errors:

- Integration framework errors
- Business logic errors

An integration framework error results from an improperly formatted message. Such errors can include XML tagging errors, namespace destination errors, and errors resulting from unlicensed namespace references.

When an integration error occurs, Maintenix displays a description of the error, and a stack trace of the Maintenix component that detected the error.

A business logic error is an error in the data contained in the XML message. Business logic errors include references to entities that do not exist in Maintenix, such as unknown locations, or unknown part numbers.

When a business logic error occurs, Maintenix displays a description of the error and an error element path that identifies the element that contained the problematic data.

Limited Availability Release

Archive integration messages

You cannot delete integration message records through the Maintenix GUI. To facilitate message searching, old integration messages can be archived. You can then filter out archived messages from your message searches.

Prerequisite

You must have the ACTION_ARCHIVE_MESSAGE permission enabled.

Steps

1. Go to the **Message Search** page.
2. Enter search criteria and click **Search**.
3. In the **Messages Found** list, select one or more messages and click **Archive**.

Logging

Integration events, exceptions, and responses are logged in the CAMO log files. The log files use standard formats that you can configure to work with your own log processing and monitoring tools.

The primary log file is `maintenix.log`. This file uses Log4j format.

HTTP requests are also logged in `weblogic-access.log` which you have the option to monitor in real time.

For information on configuring logging, see the *Maintenix Installation and Configuration Guide*.

Limited Availability Release

ABOUT IFS

IFS develops and delivers enterprise software for customers around the world who manufacture and distribute goods, maintain assets, and manage service-focused operations. The industry expertise of our people and solutions, together with commitment to our customers, has made us a recognized leader and the most recommended supplier in our sector. Our team of 3,500 employees supports more than 10,000 customers world-wide from a network of local offices and through our growing ecosystem of partners.

#forthechallengers

ifs.com

WHERE WE ARE

AMERICAS

+1 888 437 4968

ASIA PACIFIC

+65 63 33 33 00

EUROPE EAST

+48 22 577 45 00

EUROPE CENTRAL

+49 9131 77 340

UK & IRELAND

+44 1494 428 900

FRANCE, BENELUX AND IBERICA

+33 3 89 50 72 72

MIDDLE EAST AND AFRICA

+971 4390 0888

NORDICS

+46 13 460 4000

Limited Availability Release

COPYRIGHT © 2026 INDUSTRIAL AND FINANCIAL SYSTEMS, IFS AB. IFS AND ALL IFS PRODUCTS AND SERVICES NAMES ARE TRADEMARKS OF IFS. ALL RIGHTS RESERVED. THIS DOCUMENT MAY CONTAIN STATEMENTS OF POSSIBLE FUTURE FUNCTIONALITY FOR IFS'S PRODUCTS AND TECHNOLOGY. SUCH STATEMENTS ARE FOR INFORMATION PURPOSES ONLY AND SHOULD NOT BE INTERPRETED AS ANY COMMITMENT OR REPRESENTATION. THE NAMES OF ACTUAL COMPANIES AND PRODUCTS MENTIONED HEREIN MAY BE THE TRADEMARKS OF THEIR RESPECTIVE OWNERS.