IFS Aviation Maintenance

Diagnostics API Guide

ICAM 26.2



Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

IoT Gateway API v

1 Introduction 6

About CAMO Integrations 6

The CAMO Diagnostics API 7

How you can use the Diagnostics API 7

Loading fault baseline information 7

Loading failure effects from aircraft diagnostic systems 7

Loading complete faults into CAMO 8

Loading possible faults into CAMO 8

Understanding faults and failure effects in CAMO 8

Faults 8

Fault definitions 8

Failure effects 9

Failure effect definitions 9

Creating faults based on failure effects 9

2 API scenarios 11

Fault integration scenarios 11

Raising a fault and leaving it open 11

Raising a fault with a logbook reference 12

Raising a fault from a fault definition 12

Raising a possible fault 13

Raising a possible fault from a fault definition 13

Updating a fault 14

Creating a fault definition 14

Updating a fault definition 14

Creating a failure effect definition 15

Updating an existing failure effect definition 15

Adding a failure effect to a flight 15

Marking a fault definition as obsolete 16

3 Setting up the Diagnostics API 17

License requirement 17

Configuration parameters 17

4 Operational service messages 19
faults-1.0 19
Sample messages 20
Message contents 20
Mandatory elements 20
Errors and exceptions 22
Mapping elements to CAMO 23
faults-response-1.0 24
fault-definition-1.0 24
Sample messages 24
Message contents 25
Mandatory elements 25
Errors and exceptions 26
Mapping elements to CAMO 26
fault-definition-response-1.0 27
failure-effect-1.0 28
Sample messages 28
Message contents 28
Mandatory elements 28
Errors and exceptions 29
Mapping elements to CAMO 29
failure-effect-definition-1.0 30
Sample messages 30
Message contents 30
Mandatory elements 31
Errors and exceptions 31
Mapping elements to CAMO 31
failure-effect-response-1.0 32
mark-as-obsolete-1.0 32
Sample messages 32
Message contents 33
Mandatory elements 33
Errors and exceptions 33
Mapping elements to CAMO 33
mark-as-obsolete-response-1.0 34

5 Integration framework 35

Operational and querying service messages 35

SOAP request message structure 37

SOAP response message structure 38

Notification service messages 40

Configuring Maintenix for the external system's web service 41

SOAP request message structure - notification service 43

SOAP response message structure - notifications 45

Setting up the IFS Maintenix Message Ordering Processor 45

Message streams 45

Adding new message streams 45

Configuring message order timing 46

Sequencing inbound messages 47

Managing integration messages in Maintenix 48

Viewing integration messages 48

Sending messages manually within Maintenix 50

Send message manually 51

Edit and resend an integration message 52

Understanding message errors 52

Archive integration messages 53

Logging 53

IoT Gateway API

The Diagnostics API is also referred to as the IoT (Internet of Things) Gateway API.

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.

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 Diagnostics API

The CAMO Diagnostics API provides integration capabilities for on-board and off-board aircraft diagnostics reporting systems and the exchange of maintenance-significant data between external systems and CAMO.

Integrating diagnostics data from external systems into the core CAMO database means that corrective maintenance is planned with immediate access to parts, tools and labor.

How you can use the Diagnostics API

The following high-level scenarios describe the supported situations where you can use the Diagnostics API to integrate your external system with CAMO.

Loading fault baseline information

You can use the Diagnostics API to load your fault baseline information into CAMO. This is usually done on a one-time basis, but using the Diagnostics API's messaging to complete this task can save all of the manual entries that would otherwise be necessary.

Using the Diagnostics API, you can create your fault definitions (with associated corrective tasks), and your failure definitions. You can also link you failure definitions to the appropriate fault definitions.

Loading failure effects from aircraft diagnostic systems

You can use the Diagnostics API to load failure effects into CAMO from an aircraft diagnostics reporting system. CAMO will track and manage this data against the fault baseline that you have created.

All failure effects that have a corresponding failure effect defined in CAMO will be recorded against the appropriate inventory. Failure effects that are not defined in CAMO will be dropped.

When the failure effects defined in a fault definition for an aircraft have all been reported, CAMO will automatically create a fault to respond to the failed system, provided that the **manual-raise** element of the fault definition is set to false.

Loading complete faults into CAMO

You can use the Diagnostics API to load whole faults into CAMO from an external system. This is helpful if you are using and external system for your planning, but still need to import relevant data into CAMO so that CAMO can performs all of its own planning and reporting functions correctly. Also, you may be using an external system to log faults, but need to import this data into CAMO so that CAMO can plan work packages properly.

Loading possible faults into CAMO

You can use the Diagnostics API to load possible faults into CAMO. This is similar to loading faults into CAMO, except that loading possible faults requires a continuation of the workflow into CAMO because a possible fault must be examined and accepted before it is elevated to a fault in CAMO.

Possible faults can only be loaded through the Diagnostics API.

Understanding faults and failure effects in CAMO

The following sections provide a brief introduction to faults and fault definitions, failure effects and failure effect definitions, and the relationships that they form in CAMO. For more information on any of these concepts, refer to the CAMO User Guide.

Faults

Non-routines, Findings, and Discrepancies are terms used in the aviation maintenance domain to imply the identification of an aircraft component that is not working properly. CAMO uses the term Fault as a synonym to all of these industry terms. In other words, Faults are tasks created upon identifying a malfunctioning aircraft component.

Fault definitions

A fault definition is a template that you can create for faults. Fault definitions contain the baseline information required to create fault. Some of the benefits of using fault definitions are:

- fault definitions can be used to populate many of the fields when creating a new fault
- fault definitions standardize common faults
- fault definitions can be set up with failure effect definitions to create faults automatically when failure effects are reported

Failure effects

Failure effects represent the symptoms of a problem observed during flight. Failure effects are defined against an aircraft, and apply to the assembly in general. The list of failure effects defined on the assembly are then available to assign to any fault definition on the assembly.

You can only raise failure effects in CAMO using the Diagnostics API.

Failure effect definitions

Before a failure effect can be reported to CAMO using the Diagnostics API, you must first define the failure effect in CAMO.

A list of failure effects that together represent a (likely) fault can be provided on the fault definition. Consider the following example:

Fault Definition: Failed Battery

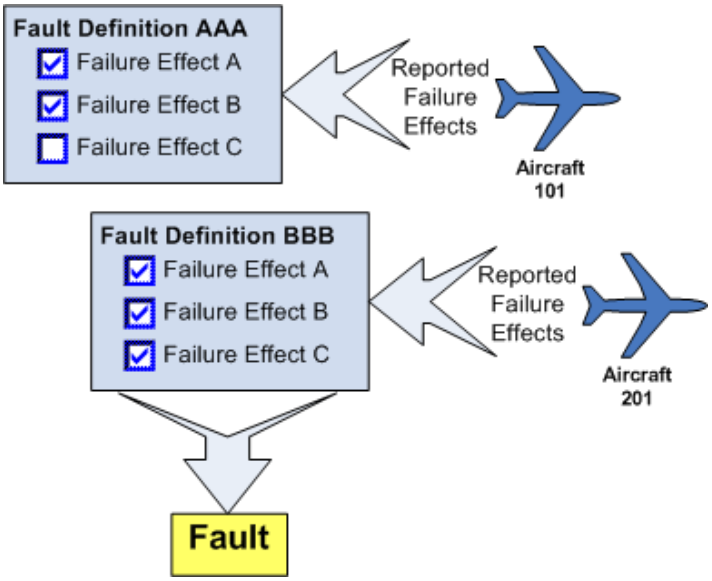
- Failure Effect A: visible corrosion on battery
- Failure Effect B: cracked battery casing

In this example, visible corrosion and a cracked casing are the two symptoms that define a failed battery.

Creating faults based on failure effects

You can use CAMO to automatically create faults from failure effects reported with the Diagnostics API. To do this, assign a list of failure effect definitions to your fault definition.

As failure effects on aircraft are reported through the Diagnostics API, CAMO compares the information to the available fault definitions for the aircraft. When all failure effects for a fault definition have been reported, the information is compared to the available fault definitions for the aircraft and CAMO creates a new fault.



Automatic fault creation

To use the automatic fault creation in CAMO the manual-raise field on the fault definition must be set to TRUE, and the auto-raise field on the failure effect message sent must be set to TRUE.

Faults that are created automatically from failure effects are raised as possible faults. Possible faults must first be examined and accepted before they are elevated to faults.

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.

Fault integration scenarios

Inbound diagnostic messages are intended for the identification of faults detected in an aircraft during flight and the supporting functions (fault definitions and failure effects).

Raising a fault and leaving it open

Goal: To raise a new logbook fault from an existing fault definition, and attach it to the appropriate inventory item within CAMO.

Summary: A fault has been detected in an aircraft or assembly, and it must be recorded in CAMO.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: If there is a work package, in the IN WORK status on this inventory item, then the fault is added to this work package; otherwise, the fault is left as an open fault to be added to a future

work package. Faults can optionally be unassigned by configuring the setting <assign-to-workpackage> to FALSE.

Raising a fault with a logbook reference

Goal: To raise a new logbook fault with a logbook reference, and attach it to the appropriate inventory item within CAMO.

Summary: A fault has been detected in an aircraft or assembly, and it must be recorded in CAMO with a logbook reference.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.
- A unique logbook reference must be specified.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: If there is a work package, in the IN WORK status on this inventory item, then the fault is added to this work package; otherwise, the fault is left as an open fault to be added to a future work package. Faults can optionally be unassigned by configuring the setting <assign-to-workpackage> to FALSE.

Raising a fault from a fault definition

Goal: To raise a new logbook fault from an existing fault definition, and attach it to the appropriate inventory item within CAMO.

Summary: You want to raise a fault has been detected in an aircraft or assembly, and the fault matches a previously defined fault definition. Values recorded on the fault definition will populate many of the values in the possible fault (and eventually the fault) that you create. If you specify values that are different from the fault definition, you will override the values provided by the fault definition. This is useful when you would like to raise a common fault, but the severity is that you want to record on the fault is different than the severity on the fault definition.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.
- The fault definition identified in the message must exist in CAMO.
- The fault definition identified in the message must be associated with the assembly or aircraft you are raising the fault for.
- No other active fault exists against the same assembly or aircraft using the same fault definition.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: If there is a work package, in the IN WORK status on this inventory item, then the fault is added to this work package; otherwise, the fault is left as an open fault to be added to a future work package. Faults can optionally be unassigned by configuring the setting `<assign-to-workpackage>` to FALSE.

Raising a possible fault

Goal: To raise a possible fault, and attach it to the appropriate inventory item within CAMO.

Summary: You would like to raise a possible fault. A possible fault is similar to a fault, except that a possible fault can have no corrective tasks associated with it. A possible fault must be examined, and marked as accepted in the CAMO GUI, before it is elevated to logbook fault. You cannot mark a possible fault as accepted through the Diagnostics API.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.
- No corrective action is specified.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: The fault is raised as a possible fault.

Raising a possible fault from a fault definition

Goal: To raise a possible fault from an existing fault definition, and attach it to the appropriate inventory item within CAMO.

Summary: You would like to raise a possible fault from an existing fault definition. Values recorded on the fault definition will populate many of the values in the possible fault (and eventually the fault) that you create. If you specify values that are different from the fault definition, you will override the values provided by the fault definition. This is useful when you would like to raise a common fault, but the severity is that you want to record on the fault is different than the severity on the fault definition.

A possible fault is similar to a fault, except that a possible fault can have no corrective tasks associated with it. A possible fault must be examined, and marked as accepted in the CAMO GUI, before it is elevated to logbook fault. You cannot mark a possible fault as accepted through the Diagnostics API.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.
- The fault definition identified in the message must exist in CAMO.
- The fault definition identified in the message must be associated with the assembly or aircraft you are raising the possible fault for.

- No corrective action is specified in the message.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: The fault is raised as a possible fault from the existing fault definition.

Updating a fault

Goal: To update an existing fault in CAMO.

Summary: You would like to modify the details of a fault that has already been created in CAMO. Any of the following details can be updated: name of the fault, severity, fault type, found on date, found by human resource code, flight found on, flight stage, flight safety impact, external usage reference, log book reference, usage snapshot and corrective task code.

Preconditions:

- The assembly or aircraft identified as being faulty must already exist within CAMO.
- The fault identified in the message must exist in CAMO.

Messaging: The external system sends an inbound *faults-1.0* message. CAMO returns a *faults-response-1.0* outbound message.

Result: The specified details of the fault are updated.

Creating a fault definition

Goal: To create a new fault definition in CAMO.

Summary: You want to create a fault template in CAMO, which you can then use to raise faults. Faults created using the fault definition will be populated with values from the fault definition automatically.

Preconditions:

- The assembly for which the fault definition will be associated must exist in CAMO.

Messaging: The external system sends an inbound *fault-definition-1.0* message. CAMO returns a *fault-definition-response-1.0* outbound message.

Result: A new fault definition is created in CAMO.

Updating a fault definition

Goal: To updates a fault definition in CAMO.

Summary: You want to update the details of an existing a fault definition in CAMO.

Preconditions:

- The assembly for which the fault definition will be associated, must exist in CAMO.

- The fault definition identified in the message must exist in CAMO.
- The fault definition must be active and not marked as obsolete.

Messaging: The external system sends an inbound *fault-definition-1.0* message. CAMO returns a *fault-definition-response-1.0* outbound message.

Result: The details of the fault definition in CAMO are updated.

Creating a failure effect definition

Goal: To create a new failure effect definition based on the information in the message.

Summary: Before you can load failure effects from an aircraft into CAMO, each failure effect must be defined in CAMO. Use this scenario to create a failure effect definition.

Preconditions:

- The assembly identified in the message must exist in CAMO.

Messaging: The external system sends an inbound *failure-effect-definition-1.0* message. CAMO returns a *failure-effect-response-1.0* outbound message.

Result: A new failure effect is defined for the assembly as specified in the message.

Updating an existing failure effect definition

Goal: To update an existing failure effect definition based on the information in the message.

Summary: You want to update a failure effect definition in CAMO.

Preconditions:

- The assembly identified in the message must exist in CAMO.
- The failure effect code must exist in CAMO.

Messaging: The external system sends an inbound *failure-effect-definition-1.0* message. CAMO returns a *failure-effect-response-1.0* outbound message.

Result: The failure effect identified in the message is updated.

Adding a failure effect to a flight

Goal: To add a failure effect with a flight in CAMO.

Summary: A failure effect is identified during a flight, and the information must be sent to CAMO, which tracks and manages failure effects. When the failure effects defined in a fault definition for an aircraft have all been reported, CAMO will automatically create a fault to respond to the failed system.

Preconditions:

- The failure effect must be defined in CAMO.

- The flight stage that the failure effect was detected on must exist in CAMO.

Messaging: The external system sends an inbound *failure-effect-1.0* message. CAMO returns a *failure-effect-response-1.0* outbound message.

Result: A new failure effect is logged against the flight. If the failure effect completes the list of failure effects assigned to a fault definition, a new fault is created in CAMO.

Marking a fault definition as obsolete

Goal: To mark a fault definition record as obsolete.

Summary: The failure effects defined by the fault definition are no longer applicable, or the assembly associated with the fault definition is no longer used. When a fault definition is marked as obsolete, it can no longer be used or updated; therefore, this scenario is useful when you want to ensure that a fault definition is no longer used.

Preconditions:

- The fault definition identified in the message must exist in CAMO.

Messaging: The external system sends an inbound *mark-as-obsolete-1.0* message. CAMO returns a *mark-as-obsolete-response-1.0* outbound message.

Result: The fault definition is marked as obsolete in CAMO.

3

Setting up the Diagnostics API

The following section discusses setup and configuration issues for the Diagnostic API.

License requirement

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

Configuration parameters

The UTL_CONFIG_PARM table contains the parameters that configure the API functionality as described in the table below.

API configuration parameters

Parameter	Description
INCOMING_ADAPTER _XML_VALIDATION	This parameter controls the schema validation of incoming XML messages. By default, schema validation is enabled. To disable schema validation, set this configuration parameter to FALSE.

Parameter	Description
OUTGOING_ADAPTER _XML_VALIDATION	This parameter controls the schema validation of outgoing XML messages. By default, schema validation is enabled. To disable schema validation, set this configuration parameter to FALSE.

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.

faults-1.0

The faults inbound message is sent by external systems to create or update faults in CAMO. The `faults-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Raising_Fault_and_Leave_Open.xml`: This message corresponds to the *Raising a fault and leaving it open* scenario. In this message, a fault is raised against the specified inventory.
- `Raising_Fault_with_Logbook_Reference.xml`: This message corresponds to the *Raising a fault with a logbook reference* scenario. In this message, a fault is raised against the specified inventory. A unique Logbook reference entry is specified.
- `Raising_Fault_From_Fault_Definition.xml`: This message corresponds to the *Raising a fault from a fault definition* scenario. In this message, a new fault is raised from the fault definition created in Creating_A_Fault_Definition.xml.
- `Raising_Possible_Fault.xml`: This message corresponds to the *Raising a possible fault* scenario. In this message, a new possible fault is raised.
- `Raising_Possible_Fault_From_Fault_Definition.xml`: This message corresponds to the *Raising a possible fault from a fault definition* scenario. In this message, a new possible fault is raised from the fault definition created in `Creating_A_Fault_Definition.xml`.
- `Updating_Fault.xml`: This message corresponds to the *Updating a fault* scenario. In this message, the sample `Raising_Fault_and_Leave_Open.xml` message is updated. The `Fault Severity` and `Found By` entries are changed.

Message contents

To create a new fault in CAMO, the <operation> element must be set to `LOG_AND_LEAVE_OPEN`, to update an existing fault, the <operation> element must be set to `UPDATE`.

In each `faults` message, you must provide either the <aircraft> or the <flight-identifier> .

If the possible fault element is not used it is assumed false. If the element is used it must be set to true or false.

Use the <usage-snapshots> element to provide the usage information recorded at the time of the fault. In CAMO, usage snapshots are used to provide additional information only, and are not used in any calculations.

The <assign-to-workpackage> element is optional. If a value is not present for that element, or the value is TRUE, when the fault is created in CAMO, it will be assinged to an existing IN WORK work package on the same inventory. If the value is present and set to FALSE, the the fault will be created and not assigned to a work package.

Mandatory elements

The following table displays the mandatory elements for the faults 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					
	Raising a fault and leave it open	Raising a fault with a logbook reference	Raising a fault from a fault definition	Raising a possible fault	Raising a possible fault from a fault definition	Updating a fault
<operation>	M	M	M	M	M	M
<fault-name>	M	M		M	M	
<inventory-item> <assembly-code>	M D	M D	M D	M D	M D	M D
<inventory-item> <assembly-bom-code>	M D	M D	M D	M D	M D	M D
<inventory-item> <assembly-pos-code>	D	D	D	D	D	D
<external-identifier>	M U	M U	M U	M U	M U	M D
<fault-severity>	M D	M D	D	M D	M D	D
<found-on-date>	M	M	M	M	M	M
<aircraft>	M D	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<failure-type>						
<fault-definition-code>			M D		M	
<found-by>	D	D	D	D	D	D

[1] Either the <aircraft> or the <flight-identifier> is mandatory in the message.

Element	Scenarios					
	Raising a fault and leave it open	Raising a fault with a logbook reference	Raising a fault from a fault definition	Raising a possible fault	Raising a possible fault from a fault definition	Updating a fault
<flight-identifier>	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<flight-stage>	D	D	D	D	D	D
<flight-safety-impact>						
<possible-fault>				M	M	
<usage-snapshots>						
<logbook-reference>	U	M U	U	U	U	U
<corrective-task-code>	D	D	D			D
<internal-identifier>						
<description>						
<assign-to-workpackage>						

Errors and exceptions

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

- The fault definition specified and the assembly and config slot of the failed inventory do not match.
- The aircraft or assembly specified is locked in CAMO.
- An active task or following task based on the specified fault definition already exists for the aircraft or assembly.
- Another active task already exists.
- The fault definition specified is obsolete.
- Logbook reference provided, but it is not unique.

- External identifier not unique.

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the *faults* message elements.

Element	Corresponding entry in CAMO GUI
<operation>	N/A.
<fault-name>	Inventory Details page, Open tab, Open Faults sub-tab, Fault Name column.
<inventory-item> <assembly-code>	Inventory Details page, Details tab, Assembly field.
<inventory-item> <assembly-bom-code>	Inventory Details page, Details tab, Config Slot field.
<inventory-item> <assembly-pos-code>	Config Slot Details page, Positions tab, Position Name column.
<external-identifier>	N/A.
<fault-severity>	Inventory Details page, Open tab, Open Faults sub-tab, Severity column.
<found-on-date>	Inventory Details page, Open tab, Open Faults sub-tab, Found on Date column.
<aircraft>	Inventory Details page, Details tab.
<failure-type>	Inventory Details page, Open tab, Open Faults sub-tab, Failure Type column.
<fault-definition-code>	Inventory Details page, Open tab, Open Faults sub-tab, Fault Name column.
<found-by>	Task Details page, Fault Information tab, Found By field.
<flight-identifier>	N/A.
<flight-stage>	Task Details page, Fault Information tab, Flight Stage field.
<flight-safety-impact>	Task Details page, Fault Information tab, Flight Safety Impact field.

Element	Corresponding entry in CAMO GUI
<possible-fault>	Inventory Details page, Possible Faults tab.
<usage-snapshots>	N/A.
<logbook-reference>	Task Details page, Details group box, Logbook Reference field.
<corrective-task-code>	Task Details page, Corrective Action group box.
<internal-identifier>	N/A.
<description>	Task Details page, Description group box.
<assign-to-workpackage>	N/A.

faults-response-1.0

The `faults-response` outbound message is returned to the external system upon a successful faults inbound message.

The `faults-response-1.0.xsd` schema contains annotations describing each element.

Each `faults-response` message contains the `internal-identifier` generated by CAMO. The `internal-identifier` can be used to identify the fault in subsequent messages.

fault-definition-1.0

The `fault-definition` inbound message is sent by external systems to create or update a fault definition record in CAMO.

The `fault-definition-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Creating_A_Fault_Definition.xml`: This message corresponds to the *Creating a fault definition* scenario. In this message, a new fault definition is created.
- `Updating_A_Fault_Definition.xml`: This message corresponds to the *Updating a fault definition* scenario. In this message, the fault definition created in the `Creating_A_Fault_Definition.xml` sample message is updated.

Message contents

When you send an inbound fault-definition message, CAMO checks whether the fault definition already exists in CAMO. If the fault definition does not exist, CAMO creates a new one. If the fault definition does exist, CAMO uses the information provided in the message to update the existing fault definition. When updating a fault definition, if you provide MTBF and MTBR values, those values are updated in the fault definition. If you do not provide MTBF and MTBR values, then the existing values in the fault definition remain unchanged.

When updating a fault definition, you may also provide failure effects, fault suppressions and item links. If you provide data that already exists in the fault definition, the duplicated data is ignored. If you provide failure effects, fault suppressions and item links that are not currently assigned to the fault definition, CAMO assigns the new data, but does NOT unassign any data. This means that you can use the fault-definition message to assign new failure effects, fault suppressions and item links; but you cannot use the fault-definition message to unassign any that already exist.

Set the `<manual-raise>` element to false if you would like CAMO to automatically create the fault when all of the failure effects associated with the fault definition have been reported.

Mandatory elements

The following table displays the mandatory elements for the `fault-definition` 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	
	Creating fault definition	Updating fault definition
<fault-name>	M	M
<fault-code>	M	M D
<fault-priority>	M D	M D
<fault-severity>	M D	M D
<fault-description>	M	M
<inventory-item> <assembly-bom-code>	M D	M D

Element	Scenarios	
	Creating fault definition	Updating fault definition
<inventory-item> <assembly-code>	M D	M D
<inventory-item> <assembly-pos-code>	D	D
<failure-category>	D	D
<deferral-class>	D	D
<deferral-reference>		
<mean-time-between-failures>		
<mean-time-between-removals>		
<trouble-shooting-task-definition>	D	D
<failure-effects>	D	D
<fault-suppression-codes>	D	D
<ietm-links>	D	D
<manual-raise>		

Errors and exceptions

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

- Fault definition specified is obsolete.

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the `fault-definition` message elements.

Element	Corresponding entry in CAMO GUI
<fault-name>	Fault Definition Details page, Identification group box, Name field.

Element	Corresponding entry in CAMO GUI
<fault-code>	Fault Definition Details page, Identification group box, Code field.
<fault-priority>	Fault Definition Details page, Details tab, Priority field.
<fault-severity>	Fault Definition Details page, Details tab, Severity field.
<fault-description>	Fault Definition Details page, Details tab, Description group box.
<inventory-item> <assembly-bom-code>	Fault Definition Details page, Identification group box, Assembly field.
<inventory-item> <assembly-code>	Fault Definition Details page, Identification group box, Config Slot field.
<inventory-item> <assembly-pos-code>	Config Slot Details page, Positions tab, Position Name column.
<failure-category>	Fault Definition Details page, Details tab, Failure Category field.
<deferral-class>	Fault Definition Details page, Details tab, Deferral Class field.
<deferral-reference>	Fault Definition Details page, Details tab, Deferral Ref field.
<mean-time-between-failures>	
<mean-time-between-removals>	
<trouble-shooting-task-definition>	Fault Definition Details page, Details tab, Troubleshooting Task field.
<failure-effects>	Fault Definition Details page, Failure Effects tab.
<fault-suppression-codes>	Fault Definition Details page, Suppression tab.
<ietm-links>	Fault Definition Details page, IETM tab.
<manual-raise>	Fault Definition Details page, Details tab, Manually Raised field.

fault-definition-response-1.0

The `fault-definition-response` outbound message is returned to the external system upon a successful `fault-definition` inbound message.

The `fault-definition-response-1.0.xsd` schema contains annotations describing each element.

Each `fault-definition-response` message contains the internal-identifier generated by CAMO. The internal-identifier can be used to identify the fault definition in subsequent messages.

failure-effect-1.0

The `failure-effect` inbound message is sent by external systems to add a failure effect to a flight.

The `failure-effect-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Adding_Failure_Effect_To_Flight.xml`: This message corresponds to the *Adding a failure effect to a flight* scenario. In this message, the specified details of the failure effect are added to the flight. The primary key of the fight is identified by the `internal-identifier` element.

Message contents

In order to add a failure effect to a flight, the failure effect must be defined in CAMO first. You can add the same failure effect to the same flight multiple times.

Mandatory elements

The following table displays the mandatory elements for the failure-effect 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 Adding a failure effect to flight
<external-identifier>	M* D U

Element	Scenarios
	Adding a failure effect to flight
<internal-identifier>	M*
<found-on>	M
<effect-name-code> <effect-code>	M D
<effect-name-code> <effect-name>	M D
<assembly-code>	M D
<failure-effect-type-code>	M D
<flight-stage>	M D
<description>	
<auto-raise>	

* Either the external-identifier or the internal-identifier is mandatory.

Errors and exceptions

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

- external identifier not unique
- found on date set in future

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the `failure-effect` message elements.

Element	Corresponding entry in CAMO GUI
<external-identifier>	N/A.
<internal-identifier>	N/A.
<found-on>	Flight Details page, Faults tab, Failure Effects group box, Date column.
<effect-name-code> <effect-code>	Failure Effect Details page, Details tab, Name field.

Element	Corresponding entry in CAMO GUI
<effect-name-code> <effect-name>	Failure Effect Details page, Details tab, Code field.
<assembly-code>	Failure Effect Details page, Details tab, Assembly field.
<failure-effect-type-code>	Flight Details page, Faults tab, Failure Effects group box, Effect Type column.
<flight-stage>	Flight Details page, Faults tab, Failure Effects group box, Flight Stage column.
<description>	Flight Details page, Faults tab, Failure Effects group box, Info column.
<auto-raise>	N/A

failure-effect-definition-1.0

The `failure-effect-definition` inbound message is sent by external systems to create or update a fault definition record in CAMO.

The `failure-effect-definition-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Creating_Failure_Effect_Definition.xml`: This message corresponds to the *Creating a failure effect definition* scenario. In this message, a new failure effect definition is created and assigned to the specified fault definition.
- `Updating_Failure_Effect_Definition.xml`: This message corresponds to the *Updating an existing failure effect definition* scenario. In this message, the failure effect definition created in the Creating_Failure_Effect_Definition.xml sample message is updated.

Message contents

When you send an inbound `failure-effect-definition` message, CAMO checks whether failure effect is already defined in CAMO. If the failure effect definition does not exist, CAMO creates a new one. If the failure effect definition does exist, CAMO uses the information provided in the message to update the existing failure effect definition.

If you provide a fault definition in your message, the failure effect is assigned to that fault definition. If you are updating an existing failure effect definition, you may use the fault definition element to assign the failure effect to a new fault definition, but the failure effect will NOT be unassigned from any previous fault definitions.

Mandatory elements

The following table displays the mandatory elements for the failure-effect-definition 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	
	Creating failure effect definition	Updating existing failure effect definition
<effect-name>	M	M
<effect-code>	M	M D
<effect-type>	M D	M D
<effect-severity>	M D	M D
<assembly-code>	M D	M D
<description>		
<fault-definition>	D	D

Errors and exceptions

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

- Fault Definition specified does not exist in CAMO.
- Fault Definition specified is linked to another assembly.

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the `failure-effect-definition` message elements.

Element	Corresponding entry in CAMO GUI
<effect-name>	Failure Effect Details page, Details group box, Code field.
<effect-code>	Failure Effect Details page, Details group box, Name field.
<effect-type>	Failure Effect Details page, Details group box, Type field.
<effect-severity>	Failure Effect Details page, Details group box, Severity field.
<assembly-code>	Failure Effect Details page, Details group box, Assembly field.
<description>	Failure Effect Details page, Description group box.
<fault-definition>	Failure Effect Details page, Fault Definitions tab.

failure-effect-response-1.0

The `failure-effect-response` outbound message is returned to the external system upon a successful `failure-effect` or `failure-effect-definition` inbound message.

The `failure-effect-response-1.0.xsd` schema contains annotations describing each element.

Each `failure-effect-response` message contains the `internal-identifier` generated by CAMO. The `internal-identifier` can be used to identify the failure effect in subsequent messages.

mark-as-obsolete-1.0

The `mark-as-obsolete` inbound message is sent by external systems to mark a fault definition record in CAMO as obsolete.

The `mark-as-obsolete-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Marking_A_Fault_Definition_as_Obsolete.xml`: This message corresponds to the *Marking a fault definition as obsolete* scenario. In this message, a specific fault definition is identified, which CAMO will mark as obsolete.

Message contents

Each `mark-as-obsolete` message that you send must identify a fault definition record that exists in CAMO. If you send a message identifying a fault definition that does not already exist in CAMO, you will receive an error.

If you send a message identifying a fault that is already marked as obsolete, the message will process successfully, but no records will be changed in CAMO.

Mandatory elements

The following table displays the mandatory elements for the `mark-as-obsolete` 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
	Marking a fault definition as obsolete
<failure-mode-code>	M D
<failure-mode-name>	M D

Errors and exceptions

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

- Code or Name does not exist already.

Mapping elements to CAMO

The following table displays the corresponding field in the CAMO user interface for each of the `mark-as-obsolete` message elements.

Element	Corresponding entry in CAMO GUI
<failure-mode-code>	Fault Definition Details page, Code field.
<failure-mode-name>	Fault Definition Details page, Name field.

mark-as-obsolete-response-1.0

The `mark-as-obsolete-response` outbound message is returned to the external system upon a successful `mark-as-obsolete` inbound message.

The `mark-as-obsolete-response-1.0.xsd` schema contains annotations describing each element.

Each `mark-as-obsolete-response` message contains the `internal-identifier` generated by CAMO. The `internal-identifier` can be used to identify the fault definition in subsequent messages.

5

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.

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`

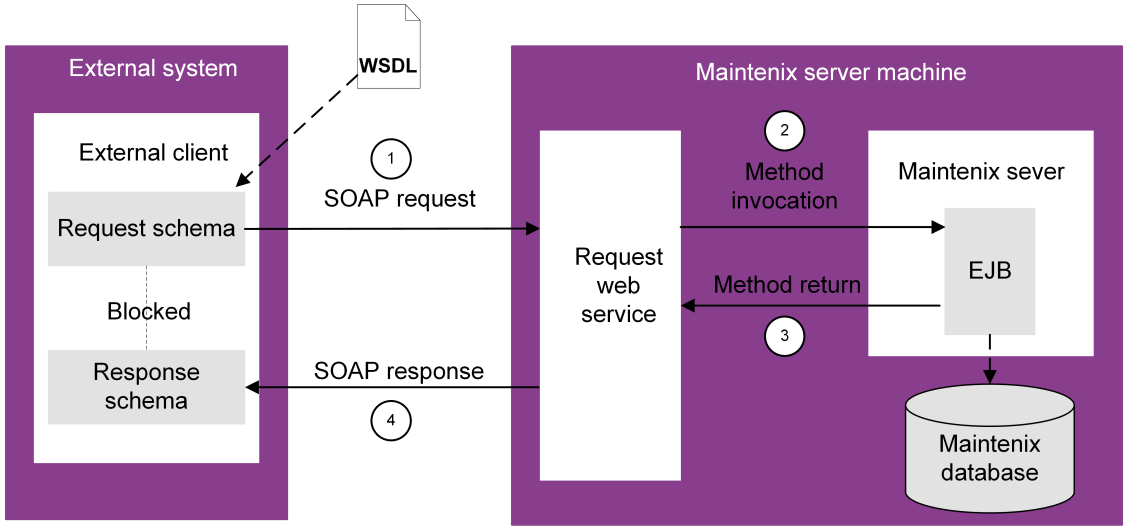
The same Web service is called, regardless of the data that is passed.

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

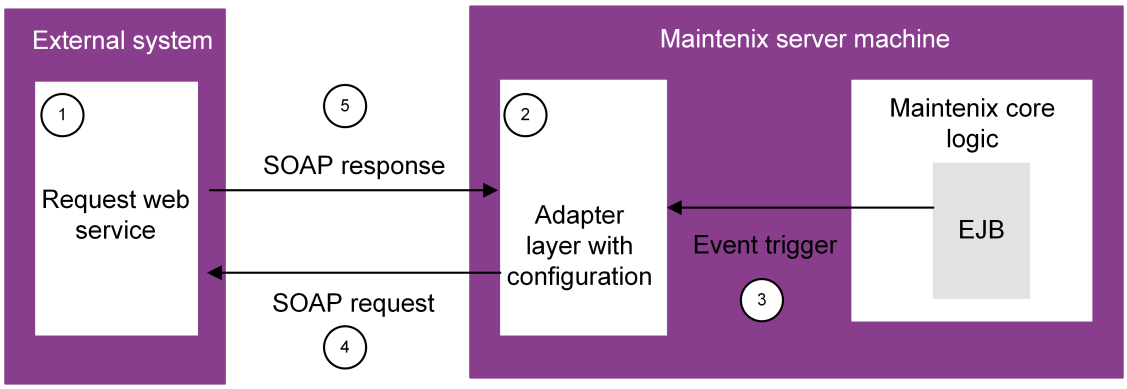
Notification service messages are used to alert an external system that something happened within Maintenix. For example an external system can be notified whenever a fault is raised, or a purchase order is updated.

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.

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.

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.

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.

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.

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.

- **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.

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*.

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

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.