IFS Aviation Maintenance

Flight API Guide

ICAM 26.2



Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

1 Introduction....7
About CAMO Integrations....7
The CAMO Flight API....8
How you can use the Flight API....8
Loading flight information from external systems....8
Notifying external systems of significant maintenance activities....8
Understanding flights in CAMO....9
Understanding usage in CAMO....9
Flight status....9
Time zones....11

2 API scenarios....12
Operational service scenarios for flights....12
Creating a planned flight....12
Creating a planned flight leg....13
Creating a historical flight leg....13
Changing an aircraft for a flight leg....13
Updating the estimated departure time....14
Updating the estimated arrival time....14
Assigning or updating the departure gate....15
Assigning or updating the arrival gate....15
Assigning or updating the arrival airport....15
Creating a flight disruption....16
Updating a flight disruption....16
Canceling a planned flight....16
Replanning a canceled flight leg....17
Recording an aircraft leaving the gate....17
Recording an aircraft returning to the gate....17
Recording an aircraft take-off....17
Recording an aircraft turning back or diverted to a new destination....18
Recording an aircraft landing....18
Recording an aircraft arrival at the gate....18
Canceling an existing flight leg....19
Marking a flight leg in error....19
Completing a flight leg....20

Operational service scenarios usage and measurement..... 20
Assigning new usage totals to inventory..... 20
Incrementing the usage on inventory..... 21
Recording an out of sequence usage..... 21
Negating an existing usage record..... 22
Recording a flight measurement..... 22
Querying service scenarios..... 22
Obtaining a flight identifier..... 22
Obtaining an aircraft status..... 23
Obtaining the earliest maintenance deadline for the aircraft..... 23
Obtaining the next due maintenance event..... 24
Obtaining aircraft capabilities..... 24
Notification service scenarios..... 24
Publishing aircraft fault information..... 25
Publishing inventory with updated deadlines..... 25
Publishing aircraft work package information..... 25
Publishing updated aircraft capabilities..... 26
Recovery steps for changing the aircraft of a departed flight..... 26

3 Setting up the Flight API..... 27
License requirement..... 27
Configuration parameters..... 27
Alerts..... 28

4 Operational service messages..... 29
flights-2.0..... 29
Sample messages..... 30
Message contents..... 31
Dates..... 31
Arrival and departure gates..... 32
Flight status transitions..... 32
Updating existing flights..... 33
Adding measurements..... 35
Adding and editing usage..... 36
Adding and editing capability requirements..... 37
Mandatory elements..... 38
Errors and exceptions..... 45

Mapping elements to CAMO UI....45
flights-response-2.0....47
flight-disruption-1.0....47
Sample messages....47
Message contents....47
Mandatory elements....48
Errors and exceptions....49
Mapping elements to CAMO UI....49
flight-disruption-responses-1.0....50
measurement-1.0....50
Sample messages....50
Message contents....50
Mandatory elements....50
Errors and exceptions....51
Mapping elements to CAMO UI....51
measurements-response-1.0....52
usage-1.2....52
Sample messages....52
Message contents....52
Mandatory elements....54
Errors and exceptions....55
Mapping elements to CAMO UI....56
usages-response-1.0....56
negate-usage-1.0....57
Sample messages....57
Message contents....57
Mandatory elements....57
Errors and exceptions....57
negate-usage-response-1.0....58

5 Querying service messages....59

aircraft-status-request-1.0....59
Sample messages....59
aircraft-status-response-1.0....59
Sample messages....60
Mapping elements to CAMO UI....60
earliest-deadlines-request-1.2....60

Sample messages....60
earliest-deadlines-response-1.2....61
Sample messages....61
flight-identifier-request-1.1....61
Sample messages....61
flight-identifier-response-1.1....61
Sample messages....62
Mapping elements to CAMO UI....62
next-due-maintenance-task-request-1.0....62
Sample messages....62
next-due-maintenance-task-response-1.0....63
Sample messages....63
aircraft-capabilities-request-1.0....63
Sample messages....63
aircraft-capabilities-response-1.0....64
Sample messages....64

6 Notification service messages....65

fault-updated-event-1.0....65
Sample messages....66
Message triggers....66
Mapping elements to CAMO UI....66
inventory-deadlines-updated-event-1.0....67
Sample messages....67
Message triggers....68
Mapping elements to CAMO UI....68
work-package-updated-event-1.2....68
Sample messages....69
Message triggers....69
Mapping elements to CAMO UI....69
aircraft-capabilities-updated-event-1.0....70
Sample messages....71
Message triggers....71
Mapping elements to CAMO UI....71

7 Integration framework....72

Operational and querying service messages....72

SOAP request message structure....74
SOAP response message structure....75
Notification service messages....77
Configuring Maintenix for the external system's web service....78
SOAP request message structure - notification service....80
SOAP response message structure - notifications....82
Setting up the IFS Maintenix Message Ordering Processor....82
Message streams....82
Adding new message streams....82
Configuring message order timing....83
Sequencing inbound messages....84
Managing integration messages in Maintenix....85
Viewing integration messages....85
Sending messages manually within Maintenix....87
Send message manually....88
Edit and resend an integration message....89
Understanding message errors....89
Archive integration messages....90
Logging....90

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 Flight API

The Flight API relays flight planning, flight tracking, and aircraft usage information to CAMO, and notifies flight operations of significant maintenance activities.

How you can use the Flight API

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

Loading flight information from external systems

You can use the Flight API to load flight planning and tracking information. You can load any related inventory usage using the Flight API.

Flying hours and flight cycles loaded into the CAMO database through the Flight API can be used to track the usage of your assets and plan maintenance requirements. CAMO produces maintenance schedules based on the availability and location of the aircraft and maintenance facility provisions, while optimizing the on-ground time of any aircraft. If flight scheduling is done in CAMO, the schedule can be revised quickly when flights are delayed or cancelled. Having the flight schedule available in the CAMO database also means that CAMO can plan maintenance while ensuring that there are enough aircraft available to meet the flight schedule.

Notifying external systems of significant maintenance activities

You can use the notification and querying services provided with the Flight API to report significant information to external systems such as: the aircraft status, upcoming maintenance deadlines, fault activity, and work package activity.

Understanding flights in CAMO

In CAMO, each flight is a single flight event from one location to another location. CAMO uses the concept of a master flight to create a relationship between flight legs. For example:

Flight 1234 from YOW to YVR with stops at YYZ and YWG. CAMO records each flight leg as a complete flight.

1. Flight 1234 from YOW to YYZ.
2. Flight 1234 from YYZ to YWG.
3. Flight 1234 from YWG to YVR.

To maintain the flight relationship, you can assign each of the flights to a common master flight number. For more information on flights, refer to the *CAMO User Guide*.

Understanding usage in CAMO

In CAMO, usage is the amount of time that an item has operated, or the number of times that an item has undergone a specific situation. Usage records on inventory are tracked to drive the scheduling of maintenance.

Flight status

Flight legs in CAMO have a status property that describes the current status of a flight leg within its life cycle. For example, the initial status for a flight leg is PLAN, but once it leaves the gate, its status goes to OUT.

The following table describes each flight status.

Status	Description
PLAN	The initial status for a planned flight leg. This is where locations, dates, initial tail assignments, and flight capability requirements are recorded for a new flight leg in CAMO. The PLAN status indicates that the flight leg has not started yet.
OUT	A flight leg enters this status once it pushes out from the gate. At this point, the tail cannot be reassigned.
OFF	This status is defined by *weight off wheels* for the aircraft, or when the aircraft has taken off.

Status	Description
ON	This status is defined by *weight on wheels* for the aircraft, or when the aircraft landed on the runway.
IN	A flight leg enters this status once it re-enters the gate. This is a short-lived status; when a flight changes to the IN status, it is then immediately and automatically changed to the COMPLETE status.
RETURN	This *return to gate* status occurs when the aircraft returns from the runway to the gate. This would normally occur for maintenance reasons when the aircraft fails a safety check while on the runway and must return to the gate.
DIVERT	This status occurs when the flight leg is air-born, and the aircraft cannot land at the original destination. Once a new destination is assigned to the flight leg, it enters a DIVERT status to warn the ground crew at the new location that an unexpected flight leg is set to land.
COMPLETE	This status signifies that the flight leg is complete and the flight leg usage has been applied to the aircraft.
EDIT	This status indicates that some information about the flight leg was updated after the flight leg was completed. This can happen when incorrect information was submitted in the original message that completed the flight. The flight status will now remain as EDIT.
ERROR	Once a flight leg has been completed, if it is deemed to be incorrect for any reason, the flight leg can be marked as *in error*, which updates the flight leg record in CAMO accordingly, and remove the usage accrued on the aircraft's current configuration.

Status	Description
CANCEL	While a flight leg is in the PLAN status, it can be canceled to remove it from the flight leg plan, to indicate that the flight leg will no longer occur.

The flight status is not managed explicitly by the client, rather it is derived from flight data. For example, if a flight has an UP-DATE but no DOWN-DATE, the flight status is reported as OFF (meaning that the flight is currently in the air).

Time zones

All Flight dates and time must be provided in the same time zone as that of the system time set within CAMO. The CAMO application server converts the date and time the user sees from the time zone the system time is set to, to the appropriate local time zone.

In the CAMO UI, departure dates display the departure location's time zone for `<scheduleddeparture-date>`, `<actual-departure-date>`, or `<up-date>` fields. Arrival dates display the arrival location's time zone for the `<scheduled-arrival-date>`, `<actual-arrival-date>`, or `<down-date>` fields.

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.

Operational service scenarios for flights

The Flight API supports the following operational service scenarios.

Creating a planned flight

Goal: To create a flight record in CAMO with a planned status.

Summary: A new flight is planned, and you must create this record in CAMO.

Preconditions:

- The aircraft inventory and airport details must exist in CAMO.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns an *flights-response-2.0* outbound message.

Result: A new flight is created in CAMO. The flight is in PLAN status, and the task deadlines for the aircraft are updated to reflect the new flight.

Creating a planned flight leg

Goal: To create a planned flight leg in CAMO.

Summary: In CAMO, each flight leg is recorded as an individual flight. The flight legs are linked together through a master flight number. In this scenario, you are creating a flight leg, and then linking it to a master flight.

Preconditions:

- The aircraft inventory and airport details must exist in CAMO.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: A new flight leg is created in CAMO. The flight leg will be in a PLAN status, and the task deadlines for the aircraft will be updated to reflect the new flight leg.

Creating a historical flight leg

Goal: To create a completed flight leg record in CAMO.

Summary: A flight has already taken place, and you must enter the record into CAMO. You can specify all of the details and complete the flight without going through the individual flight stages.

Preconditions:

- None

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: A flight leg record is created in CAMO with the data specified in the message. Usage is applied to the aircraft configuration. The flight leg is marked as historical (COMPLETE) in CAMO. Maintenance schedules are updated to reflect the impacts of the accrued usage.

If the flight is an out-of-sequence historical flight, the accrued usage is applied to the aircraft based on the arrival date for the flight and accrued usage values and scheduled maintenance for following flights is updated. For more information about usage accrual, see *Adding and editing usage*.

Changing an aircraft for a flight leg

Goal: To assign a new tail to an existing flight leg.

Summary: A new aircraft is assigned to a planned flight leg that already exists in CAMO. Once the new aircraft is assigned, the flight leg remains in PLAN status, and all scheduling details for the flight leg are updated to reflect the new aircraft assignment.

Preconditions:

- The flight leg must already exist in CAMO.
- The existing flight leg is in PLAN or CANCEL status. Using an invalid status would result in a hard stop error; for recovery steps, see *Recovery steps for changing the aircraft of a departed flight*.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The new aircraft tail is assigned to the flight leg. The flight leg remains in a PLAN status, and the details of the flight leg are transferred to the new aircraft.

Updating the estimated departure time

Goal: To update the estimated departure date and time of a flight leg.

Summary: The flight scheduling is updated, and the expected departure time is altered. This does not mean the flight leg has been rescheduled, only that the expected flight leg departure time has shifted. For example, this often occurs when a flight leg is delayed.

Preconditions:

- The flight leg exists in CAMO.
- The flight leg has not yet departed.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The estimated departure date is updated. The scheduled dates remain unchanged, and the flight leg remains in PLAN status.

Updating the estimated arrival time

Goal: To update the estimated arrival date and time of a flight leg.

Summary: The flight scheduling is updated, and the estimated arrival time is altered. This does not mean the flight leg has been rescheduled, only that the expected flight leg arrival time has shifted. For example, this often occurs when a flight leg is delayed.

Preconditions:

- The flight leg exists in CAMO.
- The flight leg has not yet landed.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The estimate arrival date is updated. The scheduled dates remain unchanged, and the flight leg status remains unchanged.

Assigning or updating the departure gate

Goal: To assign or update the departure gate for a flight leg. After a departure gate is assigned for a flight, it can be updated to a another gate, but it cannot removed from the flight record.

Summary: The airport departure terminal gate is assigned or re-assigned for a flight leg.

Preconditions:

- None

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The departure gate for the flight leg is updated, and the flight leg status remains unchanged.

Assigning or updating the arrival gate

Goal: To assign or update the arrival gate for a flight leg. After an arrival gate is assigned for a flight, it can be updated to a another gate, but it cannot removed from the flight record.

Summary: The airport arrival terminal gate is assigned or re-assigned for a flight leg.

Preconditions:

- None

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The arrival gate for the flight leg is updated, and the flight leg status remains unchanged.

Assigning or updating the arrival airport

Goal: To assign or update the arrival airport for a flight leg. After the arrival airport is assigned to a flight, it can be updated to another airport, but it cannot be removed from the flight record.

Summary: A flight leg is diverted or scheduled to land at a different airport than was planned originally. The arrival location must be updated to reflect this change.

Preconditions:

- The flight leg must not have landed.
- The flight leg can be either in plan status, or mid-flight.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The arrival airport is updated. The remaining flight leg attributes remain unchanged. Locationaware planning views in CAMO may be updated.

Creating a flight disruption

Goal: To create a new flight disruption record in CAMO.

Summary: A flight disruption occurs, and you must load records into CAMO. A flight disruption can be a flight delay, cancelation, incident, accident, and so on, and can be caused by a maintenance activity, or by another event such as weather or catering services.

Preconditions:

- The flight affected by the disruption exists in CAMO.

Messaging: The external system sends an inbound *flight-disruption-1.0* message. CAMO returns a *flight-disruption-responses-1.0* outbound message.

Result: The flight disruption record is created in CAMO.

Updating a flight disruption

Goal: To update an existing flight disruption record in CAMO.

Summary: The details of an existing flight disruption record have changed, and CAMO must be updated.

Preconditions:

- The flight disruption record exists in CAMO.

Messaging: The external system sends an inbound *flight-disruption-1.0* message. CAMO returns a *flight-disruption-responses-1.0* outbound message.

Result: The flight disruption record is updated in CAMO.

Canceling a planned flight

Goal: To cancel an existing planned flight record in CAMO.

Summary: A planned flight is cancelled, and you must record this in CAMO.

Preconditions:

- The flight exists in CAMO.
- The flight is in PLAN status.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The planned flight is cancelled in CAMO. The flight leg status is updated to CANCEL. An historical record of the cancelled flight is maintained in CAMO. The date of the cancelation is the time the message is received.

Replanning a canceled flight leg

Goal: Replan a flight leg that was canceled.

Summary: A flight leg has been canceled and will be updated with a PLAN status.

Preconditions:

- The flight leg is in the CANCEL status.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: A flight leg is updated in CAMO. The flight leg will be in a PLAN status, and the task deadlines for the aircraft will be updated to reflect the new flight leg.

Recording an aircraft leaving the gate

Goal: To record the date and time that an aircraft leaves the gate.

Summary: An aircraft has pushed out from the gate and is on the runway awaiting take off.

Preconditions:

- The flight leg is in a PLAN status before pushing out from gate.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The date and time of the event are recorded in CAMO. The status of the flight leg becomes OUT.

Recording an aircraft returning to the gate

Goal: To record the date and time that an aircraft returns to the gate from a runway.

Summary: An aircraft is on the runway, but does not take off, and must return to the gate.

Preconditions:

- The aircraft is on the runway and so the flight leg is in the OUT status.
- The aircraft did not take off.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is marked as RETURN. The date and time of the event is recorded.

Recording an aircraft take-off

Goal: To record the date and time that an aircraft takes off.

Summary: An aircraft has just left the runway, and is beginning its assent. This is the weight-offwheels event.

Preconditions:

- The flight leg is in an OUT status and has not yet taken off.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is updated to OFF. The date and time of the event is recorded.

Recording an aircraft turning back or diverted to a new destination

Goal: To update the arrival airport, and record the new arrival date of the aircraft.

Summary: The flight leg is updated while the aircraft is in the air. The aircraft will not be landing at the scheduled arrival airport; rather, it will be returning to the departure airport, or it will be diverted to a different airport than was scheduled originally.

Preconditions:

- The flight leg must be airborne.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is updated to DIVERT. The destination airport and scheduled arrival date and time are updated. Location-aware planning views in CAMO may be updated to reflect this change.

Recording an aircraft landing

Goal: To record the date and time that an aircraft lands at the destination airport.

Summary: The aircraft has just landed on the runway at the arrival airport, so the weight of the aircraft is back on the landing gears.

Preconditions:

- The flight leg is airborne.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is updated to ON. The date and time of the event are recorded.

Recording an aircraft arrival at the gate

Goal: To record the date and time that an aircraft arrives at the gate.

Summary: The aircraft has taxied to the arrival gate, and the parking brake is applied.

Preconditions:

- The flight leg status is ON.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is momentarily updated to IN, and then automatically switches to COMPLETE. The date of the gate arrival is recorded. Usage is accrued on the aircraft, and the scheduled maintenance is updated to reflect the flight leg usage.

Canceling an existing flight leg

Goal: To cancel an existing flight leg.

Summary: The flight leg has been created and assigned to an aircraft in CAMO, but for unforeseen circumstances, the flight leg is canceled.

Preconditions:

- The flight leg exists and is assigned to an aircraft in CAMO.
- Either the flight leg is still in the PLAN status or it is in the RETURN status, meaning that aircraft has returned to the departure gate.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is updated to CANCEL. A historical record of the canceled flight leg is maintained in CAMO. The date of the cancelation is the time the message is received. Location-aware planning views in CAMO may be updated to reflect this change.

Marking a flight leg in error

Goal: To mark a flight leg in error.

Summary: A flight leg has been created, assigned to an aircraft tail, and usage has been recorded against it; however, you discover that the flight leg was created due to a clerical error of some sort. In CAMO, you cannot cancel a flight that has progressed beyond the PLAN or RETURN status, but you can mark the flight in error to negate it.

Preconditions:

- The flight leg exists in CAMO.
- The flight status is not PLAN or RETURN.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is first auto-completed, and then marked as ERROR. A historical record of the error flight leg is maintained in CAMO. The record is displayed as crossed out on

the Inventory Details, History tab, Usage sub-tab. The equivalent of the usage specified for the flight leg record is subtracted from the aircraft. If the flight marked as ERROR occurred before a following flight, the TSN values shown for the crossed out row are the values for the flight that follows.

Completing a flight leg

Goal: To record a flight leg as complete.

Summary: An aircraft arrives at the gate of the final destination of the flight leg, and flight leg usage can be accrued on the aircraft.

Preconditions:

- The actual arrival time is provided.
- Optionally, usage data is provided in the message.

Messaging: The external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response-2.0* outbound message.

Result: The flight leg status is marked as a historical (COMPLETE) record. Usage is accrued on the aircraft, and the scheduled maintenance is updated to reflect the *flight* leg usage.

It is possible to receive a *flights* message for a completed flight leg after receiving the *flights* message for a later flight for the same aircraft. The *flights* message received after the fact is called an out of sequence flight. When this happens, CAMO updates the accrued usage on the aircraft as of the date the out of sequence flights were completed, corrects all subsequent accrued usage values, and updates the scheduled maintenance based on the corrected accrued usage. For more information about usage accrual, see *Adding and editing usage*.

Operational service scenarios usage and measurement

The Flight API supports the following operational service scenarios.

Assigning new usage totals to inventory

Goal: To record usage totals against an aircraft or aircraft assembly.

Summary: Usage totals were not specified when the flight was completed, and you must update CAMO. An integrating system can update inventory usage by specifying the usage totals.

Preconditions:

- The inventory assembly for the usage record exists in CAMO.
- Usage data is provided in the message.

Messaging: The external system sends an inbound *usage-1.1* message. CAMO returns a *usages-response-1.0* outbound message.

Result: The usage is recorded against the assembly, and will be applied to all of the sub-inventory items. Maintenance schedules are updated to reflect the new usage data.

Incrementing the usage on inventory

Goal: To record an incremental update to the usage totals against an aircraft or aircraft assembly.

Summary: Usage totals were not incremented when the flight was completed, and you must update CAMO. An integrating system can update inventory usage by specifying the usage deltas.

Preconditions:

- The inventory assembly for the usage record exists in CAMO.
- Usage delta is provided in the message.

Messaging: The external system sends an inbound *usage-1.1* message. CAMO returns a *usages-response-1.0* outbound message.

Result: The usage is updated against the assembly and will be applied to all of the sub-inventory items. Maintenance schedules are updated to reflect the new usage data.

Recording an out of sequence usage

Goal: To add a historical usage record delta for an aircraft or aircraft assembly, and to adjust the delta values of any usage records that have occurred since.

Summary: Historical usage information is found for an aircraft or aircraft assembly that needs to be entered into CAMO. An external system can provide usage records out of sequence. This is a rare occurrence, as the usage information is normally collected and entered in sequence.

Preconditions:

- A usage record for the aircraft or aircraft assembly must already exist with a date later than the date and time in the message, for each usage parameter in the message. An out-of-sequence (historic) usage record cannot be the most recent usage record for the inventory and usage parameter.

Messaging: The external system sends an inbound *usage-1.1* message. CAMO returns a *usages-response-1.0* outbound message.

Result: The usage is recorded against the assembly, and will be applied to all sub-inventory items. The delta values for any usage records for the assembly and sub-assemblies are adjusted to reflect the fact that a new delta usage record has been entered. Maintenance schedules are updated to reflect the new usage data. If no later usage record exists for the inventory and usage parameter, CAMO rejects the message. This includes the case where the submitted record would become the latest record, or where its collection date is the same as the current latest record.

To record the most recent usage, submit it as a non-historic record so that the Current Usage is updated.

Negating an existing usage record

Goal: To negate a usage record that was entered previously.

Summary: A usage record that had been entered erroneously must be removed from CAMO.

Preconditions:

- Usage record to be removed must exist in CAMO.

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

Result: The usage is negated. Any usage delta is applied to the next usage record.

Recording a flight measurement

Goal: To enter a measurement record associated with a flight.

Summary: Inventory measurement information was not specified when the flight was completed, and you must update CAMO.

Measurements can be applied to both historical and non-historical flights.

Preconditions:

- The inventory assembly for the usage record exists in CAMO.
- Measurement data is provided in the message.

Messaging: The external system sends an inbound *measurement-1.0* message. CAMO returns a *measurements-response-1.0* outbound message. Alternately, for non-historical flights only, the external system sends an inbound *flights-2.0* message. CAMO returns a *flights-response- 2.0* outbound message.

Result: The measurement data shall be recorded against the assembly.

Querying service scenarios

The Flight API supports the following querying service scenarios.

Obtaining a flight identifier

Goal: To obtain a unique flight identifier to be shared between the external system and CAMO.

Summary: An external system needs to identify a flight for future messaging. The external system queries CAMO for the unique flight identifier. This scenario assumes that the external system allows CAMO to generate an external identifier to share between systems.

Preconditions:

- The aircraft tail must exist in CAMO.

Messaging: The external system sends an inbound *flight-identifier-request-1.1* message. CAMO returns a *flight-identifier-response-1.1* outbound message.

Result: CAMO returns a unique external flight identifier to the external system for the specified flight.

Obtaining an aircraft status

Goal: To obtain the current status of an aircraft.

Summary: An external system needs to identify the status of a flight, and queries CAMO for the information. This scenario can arise when you wish to assign an aircraft to the flight leg, but must first check the current status of the aircraft.

Preconditions:

- The aircraft tail must exist in CAMO.

Messaging: The external system sends an inbound *aircraft-status-request-1.0* message. CAMO returns an *aircraft-status-response-1.0* outbound message.

Result: CAMO returns the location of the aircraft along with the condition of the aircraft.

> **Note**
>
> The aircraft status message capability node is now deprecated. See the *Obtaining aircraft capabilities* scenario for aircraft capability messages.

Obtaining the earliest maintenance deadline for the aircraft

Goal: To obtain the next upcoming task and deadline.

Summary: An external system needs to identify the next upcoming task and deadline for an aircraft, and queries CAMO for the information. This scenario can arise when you require the upcoming maintenance information for the aircraft before assigning the aircraft to a flight leg.

Preconditions:

- The aircraft tail must exist in CAMO.
- If usage-based deadlines are required, the usage-based deadline parameters must be specified in the message.

Messaging: The external system sends an inbound *earliest-deadlines-request-1.2* message. CAMO returns an *earliest-deadlines-response-1.2* outbound message.

Result: CAMO returns the task information for the earliest calendar-based deadline for all tasks on the aircraft. If usage-based deadlines are specified in the message, CAMO also returns the task information for the earliest deadline for each of the specified usage-based deadline parameters.

Obtaining the next due maintenance event

Goal: To obtain the next due maintenance task for an aircraft.

Summary: An external system needs to identify the next due maintenance task for an aircraft, and queries CAMO for the information. This scenario can arise when you require the upcoming maintenance information for the aircraft before assigning the aircraft to a flight leg.

Preconditions:

- The aircraft tail must exist in CAMO.

Messaging: The external system sends an inbound *next-due-maintenance-task-request-1.0* message. CAMO returns a *next-due-maintenance-task-response-1.0* outbound message.

Result: CAMO returns the deadline and work package identifier for the aircraft.

Obtaining aircraft capabilities

Goal: To obtain the aircraft capabilities for a single aircraft or an entire fleet.

Summary: An external system needs the current capabilities of an aircraft or a fleet, and queries CAMO for the information. This scenario can arise when you require the information of what the aircraft is capable of to make routing decisions.

Preconditions:

- The aircraft tail must exist in CAMO.

Messaging: The external system sends an inbound *aircraft-capabilities-request-1.0* message. CAMO returns an *aircraft-capabilities-response-1.0* outbound message.

Result: CAMO returns the capability requirements for the fleet or the specified aircraft.

Notification service scenarios

The Flight API supports the following notification service scenarios.

Publishing aircraft fault information

Goal: To receive notification about defect life cycle data.

Summary: If you are using CAMO for defect tracking, and using maintenance requirements to address the defects, this scenario retrieves information about the life cycles of defects from CAMO. This information contains life cycle events such as creation, certification and deferral.

Preconditions:

- The fault must have a severity code equal to MEL.

Messaging: CAMO sends a *fault-updated-event-1.0* outbound message.

Result: CAMO sends a notification service message for each defect created in CAMO, which will then be followed by a deferral or close message for each fault as the events occur in CAMO.

Publishing inventory with updated deadlines

Goal: To receive notification that an inventory's deadlines have been updated.

Summary: Inventory whose task deadlines have been updated are passed to the external system. The deadline modifications can occur for many reasons such as completing work packages, updating actual tasks after modifying task definitions, and rescheduling tasks.

Preconditions:

- The deadline is modified.

Messaging: CAMO sends an *inventory-deadlines-updated-event-1.0* outbound message.

Result: CAMO sends a notification service message for each inventory whose task deadlines have been updated.

Publishing aircraft work package information

Goal: To receive notification about the status of a work package for an aircraft.

Summary: Planned maintenance, as well as the status of in work maintenance, is passed to the external systems.

Preconditions: The notification message is sent when one of the following events occurs in CAMO:

- A work package is scheduled or rescheduled.
- A work scope is committed or uncommitted for a work package.
- A work package is started or un-started.
- A work package is delayed.
- A work package is canceled.
- A work package is completed.

Messaging: CAMO sends a *work-package-updated-event-1.2* outbound message.

Result: CAMO sends a notification service message for each planned work package, which will then be followed by status updates (Scheduled/Start/End/Delay) for each work package.

Publishing updated aircraft capabilities

Goal: To receive notification that an aircraft's current or configured capability level has been updated.

Summary: When a user updates a current or configured aircraft capability level, CAMO sends out a message to the external system specifying the new capability level of the aircraft.

Preconditions:

- An aircraft's current or configured capability level is modified.

Messaging: CAMO sends an *aircraft-capabilities-updated-event-1.0* outbound message.

Result: CAMO sends a notification service message for each aircraft whose current or configured capability levels have been modified.

Recovery steps for changing the aircraft of a departed flight

When a flight following system or middleware sends a message through the Flight API into Maintenix, attempting to update the aircraft identifier of a flight that is in the OUT, RETURN, OFF, DIVERT, or ON status, the system generates a hard stop error.

To recover from the hard stop error:

- If the flight is in the OUT status, the flight with the original aircraft must be returned and canceled. Then, the flight must be re-planned with the new aircraft using the same external key.
- If the flight is in the RETURN status, the flight with the original aircraft must be canceled. Then, the flight must be re-planned with the new aircraft using the same external key.
- If the flight is in the OFF, DIVERT, or ON status, the flight must be moved to ERROR. Then a replacement flight must be created with a new external key.

3

Setting up the Flight API

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

License requirement

To use the Flight API, you must have the `FLIGHT_FOLLOWING_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 Flight API functionality as described in the table below.

Flight 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.
EARLIEST_DEADLINE _TASK_ORIGINATOR _CODES	This parameter restricts the tasks that are returned to the external system in the earliest-deadlines-response message. By default, the parameter is set to return all tasks. To set the Flight API to return only those tasks with certain task originators, set the value of the parameter to the comma-separated list of task originator codes that you want included in your response.

Additionally, there are several CAMO configuration parameters that affect the creation of flights. This includes flights created using the Flight API, and flights created within the CAMO UI. For a detailed description of these parameters, see the *Administration Guide*.

Alerts

You can configure the following role-based alerts to specify which roles receive the alerts:

- **Flight Following completed a flight:** alerts users that a flight was completed by the Flight API, and the usage was updated.

For more information on setting the roles for the alerts, see to *Administration Guide*.

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.

flights-2.0

The `flights` inbound message is sent by external systems to deliver flight information to CAMO. The `flights-2.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Creating_a_Planned_Flight.xml`: This message corresponds to the *Creating a planned flight* scenario. In this message, a new planned flight is created.
- `Creating_a_Planned_Flight_Leg.xml`: This message corresponds to the *Creating a planned flight leg* scenario. In this message, a new flight leg is created and assigned to a master flight.
- `Creating_an_Historic_Flight_Leg.xml`: This message corresponds to the *Creating an historic flight leg* scenario. In this message, a new historical flight is created.
- `Changing_the_Aircraft_for_a_Flight_Leg.xml`: This message corresponds to the *Changing an aircraft for a flight leg* scenario. In this message, the aircraft from the `Create_Planned_Flight_Leg.xml` message is changed.
- `Updating_Estimated_Departure_Time.xml`: This message corresponds to the *Updating the estimated departure time* scenario. In this message, an estimated departure time is added to the `Create_Planned_Flight_Leg.xml` message.
- `Updating_Estimated_Arrival_Time.xml`: This message corresponds to the *Updating the estimated arrival time* scenario. In this message, an estimated arrival time is added to the `Update_Estimated_Departure_Time.xml` message.
- `Assigning_the_Departure_Gate.xml`: This message corresponds to the *Assigning or updating the departure gate* scenario. In this message, the departure gate is assigned to the `Create_Planned_Flight_Leg.xml` message.
- `Assigning_the_Arrival_Gate.xml`: This message corresponds to the *Assigning or updating the arrival gate* scenario. In this message, the arrival gate is assigned to the `Assign_the_Departure_Gate.xml` message.
- `Updating_the_Arrival_Airport.xml`: This message corresponds to the *Assigning or updating the arrival airport* scenario. In this message, the arrival airport from the `Create_Planned_Flight_Leg.xml` message is changed.
- `Canceling_Planned_Flight.xml`: This message corresponds to the *Canceling a planned flight* scenario. In this message, the flight from the `Create_Planned_Flight.xml` message is canceled.
- `Recording_Aircraft_Leaving_Gate.xml`: This message corresponds to the *Recording an aircraft arrival at the gate* scenario. In this message, the `Create_Planned_Flight_Leg.xml` message is updated to record the aircraft leaving the gate.
- `Recording_Aircraft_Returning_to_Gate.xml`: This message corresponds to the *Recording an aircraft returning to the gate* scenario. In this message, the `Record_Aircraft_Leaving_Gate.xml` message is updated to record the aircraft returning to the gate.

- `Recording_Aircraft_Take-off.xml`: This message corresponds to the *Recording an aircraft take-off* scenario. In this message, the `Record_Aircraft_Leaving_Gate.xml` message is updated to record the aircraft take-off.
- `Recording_Aircraft_Diverted_to_New_Destination.xml`: This message corresponds to the *Recording an aircraft turning back or diverted to a new destination* scenario. In this message, the `Record_Aircraft_Take-off.xml` message is updated to record that the aircraft is diverted to a new destination.
- `Recording_Aircraft_Landing.xml`: This message corresponds to the *Recording an aircraft landing* scenario. In this message, the `Record_Aircraft_Take-off.xml` message is updated to record the aircraft landing.
- `Recording_Aircraft_Arrival_at_Gate.xml`: This message corresponds to the *Recording an aircraft arrival at the gate* scenario.In this message, the `Record_Aircraft_Landing.xml` message is updated to record the arrival date information.
- `Canceling_Existing_Flight_Leg.xml`: This message corresponds to the *Canceling an existing flight leg* scenario. In this message, the `Record_Aircraft_Take-off.xml` message is updated to record the flight cancelation.
- `Marking_a_Flight_Leg_In_Error.xml`: This message corresponds to the *Marking a flight leg in error* scenario. In this message, the `Record_Aircraft_Returning_to_Gate.xml` message is updated to record the flight cancelation.
- `Completing_a_Flight_Leg.xml`: This message corresponds to the *Completing a flight leg* scenario. In this message, the `Record_Aircraft_Arrival_at_Gate.xml` message is updated to complete the flight. Usage and measurements are recorded.
- `Replanning_a_Canceled_Flight_Leg.xml`: This message corresponds to the *Replanning a canceled flight leg* scenario. In this message, a canceled flight leg is replanned to updated its scheduling information.

Message contents

The following sections provide detailed information for using the flights messages.

Dates

In flights messages, the `<scheduled-departure-date>` and `<scheduled-arrival-date>` are mandatory in every message.

The following dates are used to transition the flight status: `<actual-departure-date>`, `<actualarrival- date>`, `<down-date>` and `<up-date>`. These dates may be optional or mandatory depending on the current flight status.

The `<estimated-departure-date>` and `<estimated-arrival-date>` are temporary values. The `<estimated-departure-date>` is used for setting the departure date in CAMO until the Flight API receives the `<actual-departure-date>`. The `<estimated-arrival-`

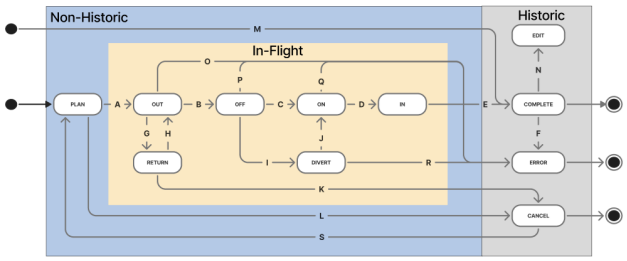
date> is used for setting the arrival date in CAMO until the Flight API receives the <actual-arrival-date>. The <actualdeparture- date> and <actual-arrival-date> should be sent only when the status of flight has changed.

Arrival and departure gates

The <arrival-gate> and <departure-gate> elements are optional in every message; however, if you set either of these values you must continue to provide them for each subsequent flights message. If you do not provide the gate information in a subsequent message, CAMO will clear the gate information from the flight record.

Flight status transitions

The following diagram displays the life cycle of a flight. For more information about flight status, see *Flight status*.



Flight status transitions

The messaging events that cause the flight status to change are as follows:

A: To transition a flight status from PLAN to OUT, set the <actual-departure-date>.

B: To transition a flight status from OUT to OFF, set the <up-date>.

C: To transition a flight status from OFF to ON, set the <down-date>.

D: To transition a flight status from ON to IN, set the <actual-arrival-date>. The flight status is then transitioned to COMPLETE automatically.

E: The transition from IN to COMPLETE is automatic.

F: To transition a flight status from COMPLETE to ERROR, set the mark-in-error attribute to 1.

G: To transition a flight status from OUT to RETURN, set the <arrival-airport> to the same value as the <departure-airport>. Do not set the <actual-arrival-date>.

H: To transition a flight status from RETURN to OUT, set the <actual-departure-date> to a new value and set the <arrival-airport> to a value different than the <departure-airport>.

I: To transition a flight status from OFF to DIVERT, set the `<arrival-airport>` to a new value.

J: To transition a flight status from DIVERT to ON, set the `<down-date>`.

K: To transition a flight status from RETURN to CANCEL, set the `mark-in-error` attribute to `1`.

L: To transition a flight status from PLAN to CANCEL, set the `mark-in-error` attribute to `1`.

M: To transition a flight status directly to COMPLETE, create a flight leg that includes the `<actual-arrival-date>`.

N: To transition a flight status from COMPLETE to EDIT, send a message that alters the flight (any attribute) after it has been completed.

O: To transition a flight status from OUT to ERROR, set the `mark-in-error` attribute to `1`.

P: To transition a flight status from OFF to ERROR, set the `mark-in-error` attribute to `1`.

Q: To transition a flight status from ON to ERROR, set the `mark-in-error` attribute to `1`.

R: To transition a flight status from DIVERT to ERROR, set the `mark-in-error` attribute to `1`.

S: To transition a flight status from CANCEL to PLAN, set the `mark-in-error` attribute to `0`.

Updating existing flights

You can send a message to update the details of an existing flight, depending on the status of the flight.

CAMO uses one of the following possible entries to identify the flight in your message:

- External key.
- Internal identifier.
- Unique combination of departure airport, departure date and flight name.

If you send a message identifying an existing flight using one of the previous entries, CAMO will use the information in your message to update the details of the existing flight. If your flight identifiers are new (they do not yet exist in CAMO), a new flight is created.

The following table provides the list of elements that may be updated, depending on the current status of the flight:

Element / Status	PLAN	OUT	RETURN	OFF	DIVERT	ON	COMPLETE	EDIT	IN	CANCEL	ERROR
aircraft-identifier	Y	-	-	-	-	-	-	-	N*	N	N
flight-attributes:											
name	Y	Y	Y	Y	Y	Y	Y	Y			

Element / Status	PLAN	OUT	RETURN	OFF	DIVERT	ON	COMPLETE	EDIT	IN	CANCEL	ERROR
master-flight-no	Y	Y	Y	Y	Y	Y	Y	Y			
operation-type	Y	Y	Y	Y	Y	Y	Y	Y			
reason	Y	Y	Y	Y	Y	Y	Y	Y			
description	Y	Y	Y	Y	Y	Y	Y	Y			
capability-requirements:											
capability-requirement:											
capability-code	Y	-	-	-	-	-	-	-			
level-code	Y	-	-	-	-	-	-	-			
measurements	Y	Y	Y	Y	Y	Y	-	-			
usages:											
usage	-	-	-	-	-	-	Y	Y			
airports:											
departure-airport	Y	Y	Y	Y	Y	Y	Y	Y			
arrival-airport	Y	Y	Y	Y	Y	Y	Y	Y			
departure-gate	Y	Y	Y	Y	Y	Y	Y	Y			
arrival-gate	Y	Y	Y	Y	Y	Y	Y	Y			
dates:											

Element / Status	PLAN	OUT	RETURN	OFF	DIVERT	ON	COMPLETE	EDIT	IN	CANCEL	ERROR
scheduled-departure-date	Y	Y	Y	Y	Y	Y	Y	Y			
scheduled-arrival-date	Y	Y	Y	Y	Y	Y	Y	Y			
estimated-departure-date	Y	-	Y	-	-	-	-	-			
estimate-darrival-date	Y	Y	Y	Y	Y	Y	-	-			
actual-departure-date	-	Y	-	Y	Y	Y	Y	Y			
actual-arrival-date	-	-	-	-	-	-	Y	Y			
up-date	-	-	-	Y	Y	Y	Y	Y			
down-date	-	-	-	-	-	Y	Y	Y			

** Flights are in the IN status only very briefly. When a flight status changes to IN, it is automatically and immediately changed to COMPLETE.*

Adding measurements

You can add a measurement for a flight using the `flights` message, or you can use the measurement message to send a measurement on its own. Both methods enable you to add a flight measurement, only the measurement message enables you to send a measurement for a historical flight.

Adding and editing usage

You can receive usage for an aircraft using the `flights` message, or you can use the usage message to send usage that is independent of the flight. For more information, see *Operational service scenarios usage and measurement*. If you use the `flights` message, the usage is recorded as a flight record in CAMO. If you use the usage message, the usage is recorded as a usage record and the usage record is applied to the inventory, but it is not associated with a flight.

With the `flights` message, usage is accrued when either of the following occurs:

- the message completes the flight, which means that the message provides the actual-arrival date for the first time for the flight
- the message updates usage related information of a historical flight, which means that the message is for an existing completed flight and that it includes corrected values

Note

Usage can be added (by completing a flight) or edited (by editing a completed flight) even if later flights are already completed in the system for the same aircraft. The flight for which you are adding or editing usage after the fact is called an out of sequence flight.

When a message is received for an out of sequence flight, CAMO updates the accrued usage values for the affected inventory items as of the date when the flight was actually completed, and corrects all accrued usage values calculated for the inventory from any flight completed later than the out of sequence flight.

This occurs on all affected inventory items including the aircraft itself, its sub-assemblies, and its components, even if the sub-assemblies or components were uninstalled from the aircraft before the out-of-sequence message was received. This ensures that usage-based deadlines for maintenance tasks are recalculated and correct, based on the most accurate information about completed flights.

The table below lists three ways to edit a flight's usage using the `flights` message.

Elements in the flights message for accruing usage

Message element(s)	Details
<dates> <up-date> <down-date> OR <actual-departure-date> <actual-arrival-date>	If no <usages> section is provided, CAMO will automatically calculate the HOURS, CYCLES, and LANDINGS based on other information in the flight. • CYCLES and LANDINGS will be automatically set to 1. • HOURS will be calculated as either the difference between the <up-date> and <down-date> if provided, or the difference between <actual-departure-date> and <actual-arrival-date> if up and down are not provided.
<usages> <usage> <usage-id> <usage-parm> <delta>	If the <usages> section is provided but no <inventory-id> element is provided within the <usages> section, then the same <delta> value will be applied to every assembly on the aircraft at the time of the flight: aircraft, engines, APUs, etc. This option allows you to apply deltas of usage parameters other than CYCLES, LANDING or HOURS. It also allows you to override the default calculations for CYCLES, LANDINGS and HOURS mentioned above.
<usages> <usage> <inventory-id> <part-number-id> <oem-part-number> <manufacturer-id> <serial-number> <usage-id> <usage-parm> <delta>	If the <usages> section is provided and an <inventory-id> element is provided within the <usages> section, then the <delta> value will only be applied to the assembly identified by this <inventory-id> element. Other assemblies on the same aircraft will not receive this <delta> This option allows you to record different delta values against different assemblies. For example, it would allow you to record different EOT (Engine Operating Time) against the left and right engines. It also allows you to record usage against a sub-assembly which is not tracked against the aircraft itself. For example: AOT (APU Operating Time) is only recorded against the APU assembly, not the aircraft assembly.

Adding and editing capability requirements

The `flights` message allows you to optionally add capability requirements for a flight. This can be done while creating flights or updating PLAN status flights only.

Elements in the flights message for managing capability requirements

When	Message element(s)	Details

Creating a flight
Editing flights in PLAN state
	• <capability-requirements>
	  • <capability-requirement>
	  • <capability-code>
	  • <level-code>
		CAMO saves the <capability-code> and <level-code> values returned through the `flights` message in the database.
		There must exist valid values for <capability-code> and <level-code> for the capability requirements in the database, otherwise the `flights` message will not be received successfully and an error message will appear.

1

Mandatory elements

This section maps the mandatory elements for the flights message in each supported scenario.

- **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/Attribute	Scenarios							
	Creating a planned flight	Creating a planned flight leg	Creating a historic flight leg	Changing the aircraft for a flight leg	Updating estimated departure time	Updating estimated arrival time	Assigning the departure gate	Assigning the arrival gate
mark-in-error="1"								
<flight-identifier>	M U	M U	M U	M D	M D	M D	M D	M D

[1] One of the sub-elements of <aircraft-identifier> is mandatory in the message.

Element/Attribute	Scenarios							
	Creating a planned flight	Creating a planned flight leg	Creating a historic flight leg	Changing the aircraft for a flight leg	Updating estimated departure time	Updating estimated arrival time	Assigning the departure gate	Assigning the arrival gate
<flight-attributes> <name>	M	M	M	M D	M D	M D	M D	M D
<flight-attributes> <master-flightno>		M						
<flight-attributes> <operationtype>								
<flight-attributes> <reason>								
<flight-attributes> <description>								
<flight-attributes> <capabilityrequirements>	D	D	D	D				
<aircraft-identifier> <barcode>	M D [1]	M D [1]		M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <inventory-identifier>	M D [1]	M D [1]		M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <aircraft-registration-code>	M D [1]	M D [1]		M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <internal-identifier>	M D [1]	M D [1]		M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<usages>								
<measurements>								
<airports> <departure-airport>	M D	M D	M D	M D	M D	M D	M D	M D

Element/Attribute	Scenarios							
	Creating a planned flight	Creating a planned flight leg	Creating a historic flight leg	Changing the aircraft for a flight leg	Updating estimated departure time	Updating estimated arrival time	Assigning the departure gate	Assigning the arrival gate
<airports> <arrival-airport>	M D	M D	M D	M D	M D	M D	M D	M D
<airports> <departure-gate>								
<airports> <arrival-gate>								
<dates> <scheduled-departuredate>	M	M	M	M	M	M	M	M
<dates> <scheduled-arrival-date>	M	M	M	M	M	M	M	M
<dates> <actual-departure-date>			M					
<dates> <actual-arrival-date>			M					
<dates> <up-date>								
<dates> <down-date>								
<dates> <estimated-departuredate>					M			
<dates> <estimated-arrival-date>						M		

Element/Attribute	Scenarios							
	Assigning the departure airport	Assigning the arrival airport	Canceling planned flight	Recording aircraft leaving gate	Recording aircraft returning to gate	Recording aircraft take-off	Recording aircraft turning back or diverted	Recording aircraft landing
mark-in-error="1"								
<flight-identifier>	M D	M D	M D	M D	M D	M D	M D	M D
<flight-attributes> <name>	M D	M D	M D	M D	M D	M D	M D	M D
<flight-attributes> <master-flightno>								
<flight-attributes> <operation-type>								
<flight-attributes> <reason>								
<flight-attributes> <description>								
<flight-attributes> <capability-requirements> <capability-requirement> <capability-code>								
<flight-attributes> <capability-requirements> <capability-requirement> <level-code>								
<aircraft-identifier> <barcode>	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <inventory-identifier>	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]

Element/Attribute	Scenarios							
	Assigning the departure airport	Assigning the arrival airport	Canceling planned flight	Recording aircraft leaving gate	Recording aircraft returning to gate	Recording aircraft take-off	Recording aircraft turning back or diverted	Recording aircraft landing
<aircraft-identifier> <aircraft-registration-code>	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <internal-identifier>	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]	M D [1]
<usages>								
<measurements>								
<airports> <departure-airport>	M D	M D	M D	M D	M D	M D	M D	M D
<airports> <arrival-airport>	M D	M D	M D	M D	M D	M D	M D	M D
<airports> <departure-gate>								
<airports> <arrival-gate>								
<dates> <scheduled-departuredate>	M	M	M	M	M	M	M	M
<dates> <scheduled-arrival-date>	M	M	M	M	M	M	M	M
<dates> <actual-departure-date>				M	M	M	M	M
<dates> <actual-arrival-date>								
<dates> <up-date>						M	M	M
<dates> <down-date>								M

Element/Attribute	Scenarios							
	Assigning the departure airport	Assigning the arrival airport	Canceling planned flight	Recording aircraft leaving gate	Recording aircraft returning to gate	Recording aircraft take-off	Recording aircraft turning back or diverted	Recording aircraft landing
<dates> <estimated-departuredate>								
<dates> <estimated-arrival-date>								
<capability-requirement>								

Element/Attribute	Scenarios			
	Recording aircraft arrival at gate	Canceling existing flight leg	Marking a flight leg in error	Completing a flight leg
mark-in-error="1"		M	M	
<flight-identifier>	M D	M D	M D	M D
<flight-attributes> <name>	M D	M D	M D	M D
<flight-attributes> <master-flight-no>				
<flight-attributes> <operation-type>				
<flight-attributes> <reason>				
<flight-attributes> <description>				
<flight-attributes> <capabilityrequirements> <capabilityrequirement> <capability-code>				
<flight-attributes> <capabilityrequirements> <capabilityrequirement> <level-code>				

Element/Attribute	Scenarios			
	Recording aircraft arrival at gate	Canceling existing flight leg	Marking a flight leg in error	Completing a flight leg
<aircraft-identifier> <barcode>	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <inventory-identifier>	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <aircraft-registration-code>	M D [1]	M D [1]	M D [1]	M D [1]
<aircraft-identifier> <internal-identifier>	M D [1]	M D [1]	M D [1]	M D [1]
<usages>				
<measurements>				
<airports> <departure-airport>	M D	M D	M D	M D
<airports> <arrival-airport>	M D	M D	M D	M D
<airports> <departure-gate>				
<airports> <arrival-gate>	M			
<dates> <scheduled-departure-date>	M	M	M	M
<dates> <scheduled-arrival-date>	M	M	M	M
<dates> <actual-departure-date>	M		M	M
<dates> <actual-arrival-date>	M		M	M
<dates> <up-date>	M		M	M
<dates> <down-date>	M		M	M
<dates> <estimated-departure-date>				
<dates> <estimated-arrival-date>				
<capability-requirement>				

Errors and exceptions

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

- The XML message format is invalid.
- Elements marked as mandatory are not provided.
- Elements that must correspond to existing data or reference terms in CAMO are invalid.
- The inventory is locked.
- The inventory is archived.

Mapping elements to CAMO UI

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

Element	Corresponding entry in CAMO GUI
<flight-identifier>	N/A
<flight-attributes> <name>	Flight Details page. Flight group box, Flight Name field.
<flight-attributes> <masterflight-no>	Flight Details page. Flight group box, Master Flight field.
<flight-attributes> <operationtype>	Flight Details page. Flight Information tab, Operation Type field.
<flight-attributes> <reason>	Flight Details page. Flight Information tab, Flight Reason field.
<flight-attributes> <description>	Flight Details page. Flight Information tab, Description group box.
<flight-attributes> <capabilityrequirements>	Flight Details page. Flight Information tab, Capability Requirements field.
<aircraft-identifier> <barcode>	Inventory Details page. Details tab, Barcode field.
<aircraft-identifier> <inventory-identifier>	Inventory Details page.
<aircraft-identifier> <aircraft-registration-code>	Inventory Details page. Details tab, Registration Code field.
<aircraft-identifier> <internal-identifier>	N/A

Element	Corresponding entry in CAMO GUI
<usages>	Flight Details page. Usage Accrued group box.
<measurements>	Flight Details page. Measurements tab.
<airports> <departure-airport>	Flight Details page. Flight Information tab, Departure Airport field.
<airports> <arrival-airport>	Flight Details page. Flight Information tab, Arrival Airport field.
<airports> <departure-gate>	Flight Details page. Flight Information tab, Departure Gate field.
<airports> <arrival-gate>	Flight Details page. Flight Information tab, Arrival Gate field.
<dates> <scheduled-departure-date>	Flight Details page. Flight Information tab, Scheduled Departure Date field.
<dates> <scheduled-arrival-date>	Flight Details page. Flight Information tab, Scheduled Arrival Date field.
<dates> <actual-departure-date>	Flight Details page. Flight Information tab, Actual Departure Date field.
<dates> <actual-arrival-date>	Flight Details page. Flight Information tab, Actual Arrival Date field.
<dates> <up-date>	Flight Details page. Flight Information tab, Up Date field.
<dates> <down-date>	Flight Details page. Flight Information tab, Down Date field.
<dates> <estimated-departure-date>	Flight Details page. Flight Information tab, Actual Departure Date field (replaced when actual-departure-date is set).
<dates> <estimated-arrival-date>	Flight Details page. Flight Information tab, Actual Arrival Date field (replaced when actual-arrival-date is set).

flights-response-2.0

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

The `flights-response-2.0.xsd` schema contains annotations describing each element.

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

flight-disruption-1.0

The `flight-disruption` inbound message is sent by external systems to create or edit flight disruption records into CAMO.

The `flight-disruption-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Creating_a_Flight_Disruption.xml`: This message corresponds to the *Creating a flight disruption* scenario. In this message, a new flight disruption record is created in CAMO.
- `Updating_a_Flight_Disruption.xml`: This message corresponds to the *Updating a flight disruption* scenario. In this message, the flight disruption record created in the previous sample is updated. The disruption type is set to canceled.

Message contents

When creating a new flight disruption record, you must specify all mandatory fields in the message, as outlined in the mandatory elements table. When updating an existing message, specify the external key and flight identifier details, but only specify the details of the flight disruption that you want to update.

To identify the task or fault that caused the flight disruption use the `<maintenance-event>` element, and specify any one of the following child elements:

- `<internal-identifier>`: The CAMO internal identifier of the task or fault.
- `<barcode>`: The barcode of the task or fault.
- `<logbook-reference>`: The logbook reference number for the task or fault.
- `<external-identifier>`: If the task or fault was created with the External Maintenance API (EMA), the external identifier or key that was provided by the EMA.

When creating a new flight disruption, the `<technical-delay>` is defaulted to FALSE if it is not specified in the message.

Mandatory elements

This section maps the mandatory elements for the flight-disruption 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 a flight disruption	Updating a flight disruption
<external-key>	M D	M D
<internal-identifier>	D	D
<flight-identifier>	M D	M D
<disruption-types>	M D	D
<delay-code>	M D	D
<maintenance-delay>	M	
<technical-delay>	M	
<flight-stage>	D	D
<notes>		
<maintenance-event> <internal-identifier>	D	D
<maintenance-event> <barcode>	D	D
<maintenance-event> <logbook-reference>	D	D
<maintenance-event> <external-identifier>	D	D

Errors and exceptions

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

- The XML message format is invalid.
- Elements marked as mandatory are not provided.
- Elements that must correspond to existing data or reference terms in CAMO are invalid.
- The flight does not exist in CAMO.

Mapping elements to CAMO UI

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

Element	Corresponding entry in CAMO GUI
<external-key>	N/A
<internal-identifier>	Inventory Details page.
<flight-identifier>	Flight Details page.
<disruption-types>	Flight Disruption Details page, Disruption Types field.
<delay-code>	Flight Disruption Details page, Delay Code field.
<maintenance-delay>	Flight Disruption Details page, Maintenance Delay field.
<technical-delay>	Flight Disruption Details page, Technical Delay field.
<flight-stage>	Flight Disruption Details page, Flight Stage field.
<notes>	Flight Disruption Details page, Notes field.
<maintenance-event> <internal-identifier>	N/A
<maintenance-event> <barcode>	Task Details page, Details group box, Barcode field.
<maintenance-event><logbook-reference>	Task Details page, Details group box, Logbook Reference field.
<maintenance-event> <external-identifier>	N/A

flight-disruption-responses-1.0

The `flight-disruption-responses` outbound message is returned to the external system upon a successful flight-disruption inbound message.

The `flight-disruption-responses-1.0.xsd` schema contains annotations describing each element.

Each `flight-disruption-responses` message contains the `internal-identifier` generated by CAMO. The `internal-identifier` can be used to identify the flight disruption in subsequent messages.

measurement-1.0

The `measurement` inbound message is sent by external systems to add measurement values to a flight.

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

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Recording_Flight_Measurement.xml`: This message corresponds to the *Recording a flight measurement* scenario. In this message, measurement values are applied to a flight.

Message contents

You can add a measurement for a flight using the `flights` message, or you can use the `measurement` message to send a measurement on its own. Both methods enables you to add a flight measurement, only the `measurement` message enable you to send a measurement for an historical flight.

The Flight API message schema does not support the measurement conversion factors. The inbound messages must contain the default unit of measure for the measurement values.

Mandatory elements

This section maps the mandatory elements for the `measurement` 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	Scenario
	Record a flight measurement
<flight-identifier>	M D
<inventory-identifier>	
<measurement-id> <parameter>	M D
<measurement-id> <value>	M

Errors and exceptions

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

- The XML message format is invalid.
- Elements marked as mandatory are not provided.
- Elements that must correspond to existing data or reference terms in CAMO are invalid.
- The flight does not exist in CAMO.
- The measurement parameter does not exist in CAMO.
- The inventory is locked.
- The inventory is archived.

Mapping elements to CAMO UI

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

Element	Corresponding entry in CAMO GUI
<flight-identifier>	N/A
<inventory-identifier>	Inventory Details page.
<measurement-id> <parameter>	Flight Details page, Measurements tab, Parameter column.
<measurement-id> <value>	Flight Details page, Measurements tab, Value column.

measurements-response-1.0

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

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

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

usage-1.2

The `usage` inbound message is sent by external systems to add or update usage on an inventory.

The `usage-1.2.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Assigning_New_Usage_Totals_to_Inventory.xml`: This message corresponds to the *Assigning new usage totals to inventory* scenario. In this message, new usage records are assigned to an inventory.
- `Incrementing_Usage_on_Inventory.xml`: This message corresponds to the *Incrementing the usage on inventory* scenario. In this message, the usage created in `Assign_New_Usage_Totals_to_Inventory.xml` is updated.
- `Recording_Out_of_Sequence_Usage.xml`: This message corresponds to the *Recording an out of sequence usage* scenario. In this message, an historical usage delta is recorded. In this case, the date of the usage collection occurs before the `Increment_Usage_on_Inventory.xml`, but after the date of the initial usage record in the `Assign_New_Usage_Totals_to_Inventory.xml` message.

Message contents

You can send usage for an aircraft using the flights message, or you can use the usage message to send usage that is independent of the flight.

If you use the `flights` message, the usage is recorded in a flight record in CAMO. If you use the `usage` message, the usage is recorded in a usage record in CAMO, and it is applied to the inventory, but not associated with a flight. This is useful to record usage gained in non-flight events such as monthly odometer readings for APUs. If your organization records flights, it is best to record usage by using the `flights` message. See *Adding and editing usage* for information on sending usage using the `flights` message.

To assign new usage or correct existing usage with the `usage` message, use the "time since new" (TSN), "time since overhauled" (TSO), and "time since install" (TSI) totals to replace any existing usage values in CAMO. To update an existing usage value in CAMO, use the delta value to specify the increment to change the existing usage value.

You may specify negative delta value to decrease an existing usage value. You can also send an out-of-sequence usage record by setting the `process-as-historic` element to `true`.

Note A usage record submitted with `process-as-historic` set to `true` must have at least one later usage record already recorded in CAMO for the same inventory and usage parameter. Out-of-sequence (historic) processing only adjusts the deltas of the surrounding usage records. It does not update the inventory's current usage. Therefore, if the submitted historic record would become the latest usage record for the inventory and usage parameter, CAMO rejects the message and does not create the record. This includes the cases where:

- No usage history exists for the inventory and usage parameter,
- The collection date of the submitted record is the same as the current latest usage record,
- In a multi-parameter message, any one usage parameter has no later usage record.

To record the most recent usage, submit the message with `process-as-historic` set to `false` (or omit it) so that current usage is updated correctly.

The following is an example to illustrate how an out-of-sequence usage record works.

Your existing data is as follows:

- Usage Record A
  - February 1, 2019
  - TSN = 100 hours
- Usage Record C
  - February 10, 2019
  - TSN: 200 hours
  - Delta: 100 hours

You can add a new usage record as follows:

- Usage Record B
  - February 6, 2019
  - TSN: 140 hours

This will create a usage record with a date of February 6th, 2019, and a TSN value of 140 hours. CAMO validates the TSN by looking up the usage records occurring prior to and after the specified date (February 6, 2009); usage records A and C in this example. The TSN value of 140 from

usage record B is valid, because it is in between the TSN values of records A and B (100 and 200 respectively).

The delta used when creating usage record B is set to 40, because this is the delta between usage record B and A. The delta value for the subsequent usage record (usage record C) is updated to 60 because this is the delta between usage record B and C. The overall total usage for the component remains the same.

Your resulting data is as follows:

- Usage Record A
  - February 1, 2019
  - TSN = 100 hours
- Usage Record
  - February 6, 2019
  - TSN: 140 hours
  - Delta: 40 hours
- Usage Record C
  - February 10, 2019
  - TSN: 200 hours
  - Delta: 60 hours

CAMO will not update any other usage snapshots or (sub)component usage values. It will also not make any corrections for components that may have been installed or removed after the usage correction occurred.

Mandatory elements

This section maps the mandatory elements for the usage 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		
	Assigning new usage totals to inventory	Incrementing usage on inventory	Recording out of sequence usage
<usage-attributes> <name>	M	M	M

Element	Scenarios		
	Assigning new usage totals to inventory	Incrementing usage on inventory	Recording out of sequence usage
<usage-attributes> <collection-date>	M	M	M
<usage-attributes> <process-ashistoric>			M
<usage-attributes> <documentreference>			
<usage-attributes> <externalidentifier>	M	M	M
<assembly><barcode>	M D	M D	M D
<collected-usage> <usage-parm>	M D	M D	M D
<collected-usage> <value> <delta>		M	
<collected-usage> <value> <total> <tsn>	M		M
<collected-usage> <value> <total> <tso>			
<collected-usage> <value> <total> <tsi>			

Errors and exceptions

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

- The XML message format is invalid.
- The usage parameter does not exist in CAMO.
- The assembly does not exist in CAMO.
- The usage parameter is not assigned to the selected assembly.
- The inventory is locked.
- The inventory is archived.

- The historic usage record (process-as-historic = true) would be the latest usage record for the inventory and usage parameter.

Mapping elements to CAMO UI

Element	Corresponding entry in CAMO GUI
<usage-attributes> <name>	Usage Record Details page, Name field.
<usage-attributes> <collection-date>	Usage Record Details page, Collection Date field.
<usage-attributes> <process-as-historic>	N/A
<usage-attributes> <document-reference>	Usage Record Details page, Document Reference field.
<usage-attributes> <external-identifier>	N/A
<assembly><barcode>	Inventory Details page.
<collected-usage> <usage-parm>	Inventory Details page, Details tab, Current Usage group box, Usage Parm column.
<collected-usage> <value> <delta>	Usage Record Details page, Usage Accrued tab, Delta column.
<collected-usage> <value> <total> <tsn>	Inventory Details page, Details tab, Current Usage group box, TSN column.
<collected-usage> <value> <total> <tso>	Inventory Details page, Details tab, Current Usage group box, TSO column.
<collected-usage> <value> <total> <tsi>	Inventory Details page, Details tab, Current Usage group box, TSI column.

usages-response-1.0

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

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

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

negate-usage-1.0

The `negate-usage` inbound message is sent by external systems negate a usage record that was previously sent by the external system.

The `negate-usage-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Negating_an_Existing_Usage_Record.xml`: This message corresponds to the *Negating an existing usage record* scenario. In this message, a usage record that was previously sent to CAMO is removed from the system.

Message contents

In order to negate a usage record that was previously sent from an external system, you must specify the external identifier of the previous message to correctly identify the usage record.

Mandatory elements

This section maps the mandatory elements for the `negate-usage` 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	Scenario
	Negating an existing usage record
<external-identifier>	M
<internal-identifier>	
<note>	

Errors and exceptions

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

- The XML message format is invalid.
- The usage record does not exist in CAMO.
- The inventory is locked.
- The inventory is archived.

negate-usage-response-1.0

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

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

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

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.

aircraft-status-request-1.0

The `aircraft-status-request` inbound message is sent by the external system. CAMO returns the requested information in an `aircraft-status-response` outbound message.

The `aircraft-status-request-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Aircraft_Status_Request.xml`: This message corresponds to the *Obtaining an aircraft status* scenario. In this message, the aircraft barcode is sent.

aircraft-status-response-1.0

CAMO sends an `aircraft-status-response` outbound message following an aircraft-status-request inbound message from the external system.

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

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Aircraft_Status_Response.xml`: This message corresponds to the *Obtaining an aircraft status* scenario.In this message, the information requested in `Aircraft_Status_Request.xml` is returned to the external system.

Mapping elements to CAMO UI

The following table displays the corresponding field in the CAMO user interface for each of the `aircraft-status-response` message elements.

Element	Corresponding entry in CAMO GUI
<airport-code>	Inventory Details page, Details tab, Location field.
<condition>	Inventory Details page, Details tab, Condition field.
<capability>	Inventory Details page, Details tab, Capability field.

earliest-deadlines-request-1.2

The `earliest-deadlines-request` inbound message is sent by the external system. CAMO returns the requested information in an `earliest-deadlines-response` outbound message.

The `earliest-deadlines-request-1.2.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Earliest_Deadlines_Request_1.xml`: This message corresponds to the *Obtaining the earliest maintenance deadline for the aircraft* scenario. In this message, the aircraft barcode is sent.
- `Earliest_Deadlines_Request_2.xml`: This message corresponds to the *Obtaining the earliest maintenance deadline for the aircraft* scenario. In this message, the aircraft barcode and specific usage parameters are sent.

earliest-deadlines-response-1.2

CAMO sends an `earliest-deadlines-response` outbound message following an `earliest-deadlines-request` inbound message from the external system.

The `earliest-deadlines-response-1.2.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Earliest_Deadlines_Response_1.xml`: This message corresponds to the *Obtaining the earliest maintenance deadline for the aircraft* scenario. In this message, the information requested in `Earliest_Deadlines_Request_1.xml` is returned to the external system.
- `Earliest_Deadlines_Response_2.xml`: This message corresponds to the *Obtaining the earliest maintenance deadline for the aircraft* scenario.In this message, the information requested in `Earliest_Deadlines_Request_2.xml` is returned to the external system.

flight-identifier-request-1.1

The `flight-identifier-request` inbound message is sent by the external system. CAMO returns the requested information in a `flight-identifier-response` outbound message.

The `flight-identifier-request-1.1.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Flight_Identifier_Request.xml`: This message corresponds to the *Obtaining a flight identifier* scenario. In this message, the aircraft barcode and a point in time during the flight is sent to CAMO.

flight-identifier-response-1.1

CAMO sends an `flight-identifier-response` outbound message following a `flight-identifier-request` inbound message from the external system.

The `flight-identifier-response-1.1.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Flight_Identifier_Response.xml`: This message corresponds to the *Obtaining a flight identifier* scenario. In this message, the information requested in `Flight_Identifier_Request.xml` is returned to the external system.

Mapping elements to CAMO UI

The following table displays the corresponding field in the CAMO user interface for each of the `flight-identifier-response` message elements.

Element	Corresponding entry in CAMO GUI
<external-key>	N/A
<flight-name>	Flight Details page, Flight Name field.
<scheduled-departure-date>	Flight Details page, Flight Information tab, Scheduled Departure Date field.
<departure-airport>	Flight Details page, Flight Information tab, Departure Airport field.
<internal-identifier>	N/A

next-due-maintenance-task-request-1.0

The `next-due-maintenance-task-request` inbound message is sent by the external system. CAMO returns the requested information in a `next-due-maintenance-task-response` outbound message.

The `next-due-maintenance-task-request-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Next_Due_Maintenance_Task_Request.xml`: This message corresponds to the *Obtaining the next due maintenance event* scenario. In this message, the aircraft barcode is sent.

next-due-maintenance-task-response-1.0

CAMO sends an `next-due-maintenance-task-response` outbound message following a `next-duemaintenance- task-request` inbound message from the external system.

The `next-due-maintenance-task-response-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Next_Due_Maintenance_Task_Response.xml`: This message corresponds to the *Obtaining the next due maintenance event* scenario. In this message, the information requested in `Next_Due_Maintenance_Task_Request.xml` is returned to the external system.

aircraft-capabilities-request-1.0

The `aircraft-capabilities-request` inbound message is sent by the external system. CAMO returns the requested information in a `aircraft-capabilities-response` outbound message.

The `aircraft-capabilities-request-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Aircraft_Capabilities_Request.xml`: This message corresponds to the *Obtaining aircraft capabilities* scenario. In this message, if the information is needed for an entire fleet, the inbound message does not include any aircraft identifiers. If the information is needed for a single aircraft, the inbound message includes one of the following aircraft identifiers:
  - barcode
  - inventory-identifier
  - aircraft-registration-code
  - internal-identifier

aircraft-capabilities-response-1.0

CAMO sends an `aircraft-capabilities-response` outbound message following an `aircraft-capabilities-request` inbound message from the external system.

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

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Aircraft_Capabilities_Response.xml`: This message corresponds to the *Obtaining aircraft capabilities* scenario. In this message, the following information for the fleet or the specified aircraft is sent:
  - capability-code
  - capability-description
  - configured-capability-level
  - current-capability-level
  - last-modifed

6

Notification service messages

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

- **Sample Messages:** A description of each of the sample messages that complete the supported scenarios.
- **Message Triggers:** A list of events that trigger the outbound notification message.
- **Mapping Elements to the CAMO GUI:** A table with the corresponding fields in CAMO for each element.

fault-updated-event-1.0

The `fault-updated-event` outbound message is sent to the external system.

To enable this notification service message, you must set the `ACTIVE_BOOL` to 1 for the following triggers CAMO's `UTL_TRIGGER` table.

TRIGGER_ID	TRIGGER_CD	TRIGGER_NAME
99987	MX_CF_CERTIFY	FAULT CERTIFICATION
99986	MX_CF_NFF	FAULT NFF
99959	MX_EDIT_FAULT	EDIT FAULT
99958	MX_CF_SET_OP_RESTRICTION	SET OPER RESTRICTION
99988	MX_TS_DEFER	TASK DEFER
99921	MX_FAULT_DEADLINE_EXTENDED	EXTEND FAULT DEADLINES

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

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Publishing_Aircraft_Fault_Information.xml`: This message corresponds to the *Publishing aircraft fault information* scenario. In this message, fault information is published.

Message triggers

The `fault-updated-event` message is published to external systems when one of the following events occurs in CAMO:

- A user adds operation restrictions to a fault.
- A user edits a fault.
- A user edits a fault to downgrade the severity of the fault.
- A fault is deferred.
- A fault is completed.
- A user extends a deadline on a fault.

Note Set the `FLIGHT_ADAPTER_PUBLISH_MEL_ONLY` utl_config_parm to `TRUE` if you want the flight adapter to publish only MEL severity fault messages.

Mapping elements to CAMO UI

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

Element	Corresponding entry in CAMO GUI
<name>	Task Details page.
<description>	Task Details page, Task Execution tab, Description group box.
<barcode>	Task Details page, Barcode field.
<aircraft-identifier>	Inventory Details page.
<aircraft-status> <capability>	Inventory Details page, Details tab, Capability field.
<aircraft-status> <operatingstatus>	Inventory Details page, Details tab, Inventory Condition field.
<failed-system>	Task Details page, Aircraft field.

Element	Corresponding entry in CAMO GUI
<status> <log-reference>	Task Details page, Logbook Reference field.
<source>	Task Details page, Fault Information tab, Fault Source field.
<found-on-date>	Task Details page, Fault Information tab, Found on Date field.
<found-during-flight>	Task Details page, Fault Information tab, Found During Flight field.
<completed-date> <severity>	Task Details page, Fault Information tab, Fault Severity field.
<deferral-information>	Task Details page, Fault Information tab, Fault Details group box.

inventory-deadlines-updated-event-1.0

The `inventory-deadlines-updated-event` outbound message is sent to the external system.

To enable this notification service message, you must set the `ACTIVE_BOOL` to 1 for the following triggers CAMO's `UTL_TRIGGER` table.

TRIGGER_ID	TRIGGER_CD	TRIGGER_NAME
99957	MX_FL_DEADLINE_UPDATED	Inventory Deadlines Updated

The `inventory-deadlines-updated-event-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Publishing_Inventory_Deadlines_Updated.xml`: This message corresponds to the *Publishing inventory with updated deadlines* scenario. In this message, an inventory with updated deadlines is sent to the external system.

Message triggers

The `inventory-deadlines-updated-event` message is published to external systems when the deadline for an inventory is updated for any reason.

Mapping elements to CAMO UI

The following table displays the corresponding field in the CAMO user interface for each of the `inventory-deadlines-updated-event` message elements.

Element	Corresponding entry in CAMO GUI
<oem-serial-number>	Inventory Details page, OEM Serial No field.
<oem-part-number>	Inventory Details page, OEM Part Number field.

work-package-updated-event-1.2

The work-package-updated-event outbound message is sent to the external system.

To enable this notification service message, you must set the `ACTIVE_BOOL` to 1 for the following triggers CAMO's `UTL_TRIGGER` table.

TRIGGER_ID	TRIGGER_CD	TRIGGER_NAME
99995	MX_TS_STARTWORK	start work order request
99994	MX_TS_UNDOSTARTWORK	unstart work order request
99998	MX_TS_CREATE	plan work order request
99997	MX_TS_COMMIT	commit work order request
99996	MX_TS_UNCOMMIT	uncommit work order request
99991	MX_TS_CANCEL	cancel the task
99992	MX_TS_COMPLETE	return to service request
99990	MX_TS_SCHEDULE_ EXTERNAL	schedule the work order external
99993	MX_TS_DELAY	estimate return to service request
99989	MX_TS_SCHEDULE_ INTERNAL	schedule the work order internal

The `work-package-updated-event-1.1.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Publishing_Aircraft_Work_Package_Information.xml`: This message corresponds to the *Publishing aircraft work package information* scenario. In this message, work package information is published.

Message triggers

The `work-package-updated-event` message is published to external systems when one of the following events occur in CAMO:

- A work package is scheduled or rescheduled in CAMO.
- A work scope is committed or uncommitted for a work package in CAMO.
- A work package is started or un-started in CAMO.
- A work package is delayed in CAMO.
- A work package is cancelled in CAMO.
- A work package is completed in CAMO.

Mapping elements to CAMO UI

The following table displays the corresponding field in the CAMO user interface for each of the `work-package-updated-event` message elements.

Element	Corresponding entry in CAMO GUI
<work-package-no>	Work Package Details page, Work Package No field.
<name>	Work Package Details page.
<aircraft> <aircraft-identifier>	Inventory Details page.
<aircraft> <aircraft-status> <capability>	Inventory Details page, Details tab, Capability field.
<aircraft> <aircraft-status> <operating-status>	Inventory Details page, Details tab, Inventory Condition field.
<barcode>	Work Package Details page, Details tab, Barcode field.

Element	Corresponding entry in CAMO GUI
<status>	Work Package Details page, Details tab, Status field.
<work-types>	Work Package Details page, Details tab, Work Type(s) field.
<type-subtype>	Work Package Details page, Details tab, Type - Subtype field.
<location>	Work Package Details page, Details tab, Work Location field.
<scheduled-start-date>	Work Package Details page, Details tab, Scheduled Start field.
<scheduled-end-date>	Work Package Details page, Details tab, Scheduled End field.
<actual-start-date>	Work Package Details page, Details tab, Actual Start Date field.
<actual-end-date>	Work Package Details page, Details tab, Estimated End field.
<external-vendor>	Work Package Details page, RO Vendor field.
<delay> <delay-reason>	Work Package Details page, History tab, Stage Reason column.
<delay> <delay-description>	Work Package Details page, History tab, Note column.
<last-user-note>	Work Package Details page, Details tab, History group box.

aircraft-capabilities-updated-event-1.0

The `aircraft-capabilities-updated-event` outbound message is sent to the external system.

To enable this notification service message, you must set the `ACTIVE_BOOL` to 1 for the following triggers CAMO's `UTL_TRIGGER` table.

TRIGGER_ID	TRIGGER_CD	TRIGGER_NAME
99914	MX_AC_CAPABILITIES_UPDATED	Aircraft Capabilities Updated

The `aircraft-capabilities-updated-event-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Aircraft_Capabilities_Updated_Event.xml`: This message corresponds to the *Publishing updated aircraft capabilities* scenario. In this message, the updated aircraft capability is sent to the external system.

Message triggers

The `aircraft-capabilities-updated-event` message is published to external systems when the aircraft's current or configured capability level is updated.

The following actions in CAMO may result in a change to an aircraft's capability levels:

- Directly updating capability levels for the aircraft (from the Capabilities tab).
- Removing available capabilities or levels from the aircraft's assembly (if it results in removing or degrading a level set on the aircraft).
- Unlocking an aircraft, if it results in capabilities being degraded for the aircraft (due to previous assembly changes).

Mapping elements to CAMO UI

The following table displays the corresponding field in the CAMO user interface for each of the `aircraft-capabilities-updated-event` message elements.

Element	Corresponding entry in CAMO GUI
<capability-code>	Inventory Details page, Capabilties tab, Capability field.
<configured-level-code>	Inventory Details page, Capabilties tab, Configured level field.
<current-level-code>	Inventory Details page, Capabilties tab, Current level field.

7

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`

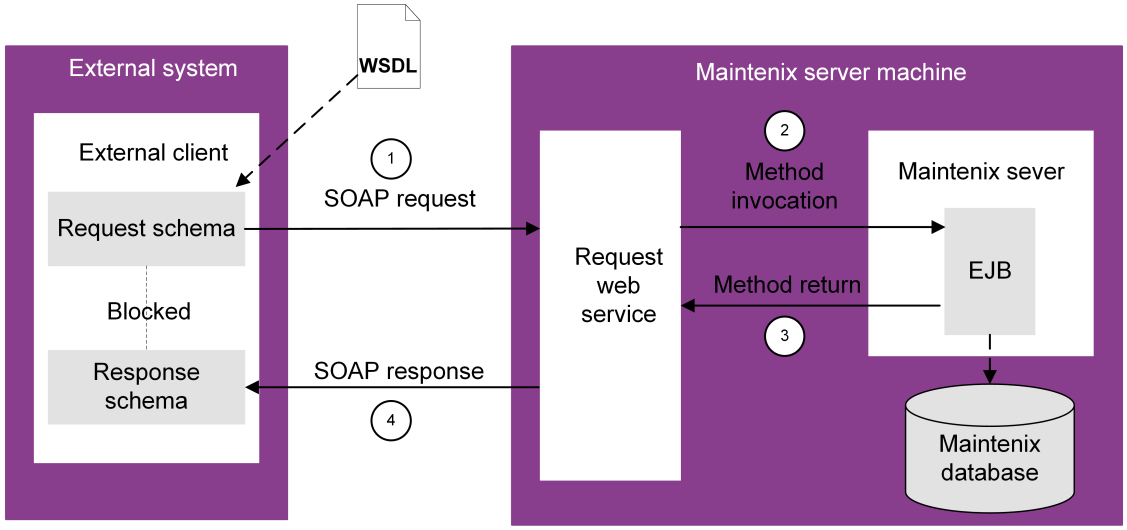
The same Web service is called, regardless of the data that is passed.

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

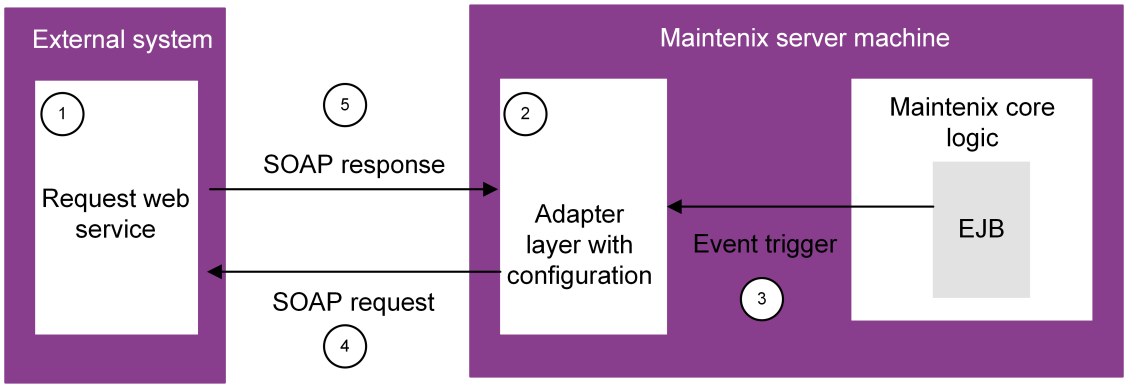
Notification service messages are used to alert an external system that something happened within Maintenix. For example an external system can be notified whenever a fault is raised, or a purchase order is updated.

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.