IFS Aviation Maintenance

Human Resources API Guide

ICAM 26.2



Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

1 Introduction....7
About CAMO Integrations....7
CAMO Human Resources API....8

2 Human Resources API scenarios....9
Creating a new user....9
Updating a user's information....9
Enabling a user....10
Disabling a user....10
Deleting a user....10
Assigning a user to a department....11
Unassigning a user from a department....11
Assigning a role to a user....12
Unassigning a role from a user....12
Assigning an authority to a user....12
Unassigning an authority from a user....13
Assigning all authorities to a user....13
Unassigning all authorities from a user....13
Assigning a skill to a user....13
Unassigning a skill from a user....14
Assigning a license to a user....14
Updating a user's license....14
Unassigning a license from a user....15
Suspending a user's license....15
Reactivating a user's suspended license....15
Configuration parameters....16
Shift adjustment scenarios....17
Creating a shift adjustment for a user....17
Updating a user's shift adjustment....17
Deleting a shift adjustment....18
Creating a temporary crew assignment....18

3 Setting up the Human Resources API....19
License requirement....19
Configuration parameters....19

4 Operational service messages 21
create_user-2.0 21
Sample messages 22
Message contents 22
Mandatory elements 22
Mapping elements to CAMO 24
user_created-2.0 25
update_user-1.0 25
Sample messages 25
Message contents 25
Mandatory elements 25
Mapping elements to CAMO 26
user_updated-1.0 27
enable_user-1.0 27
Sample messages 27
Message contents 27
Mandatory elements 27
Mapping elements to CAMO 28
user_enabled-1.0 28
disable_user-1.0 28
Sample messages 28
Message contents 28
Mandatory elements 29
Mapping elements to CAMO 29
user_disabled-1.0 29
delete_user-1.0 29
Sample messages 29
Message contents 30
Mandatory elements 30
Mapping elements to CAMO 30
user_deleted-1.0 30
assign_user_to_department-1.0 31
Sample messages 31
Message contents 31
Mandatory elements 31
Mapping elements to CAMO 31

user_assigned_to_department-1.0 32
unassign_user_from_department-1.0 32
Message contents 32
Mandatory elements 32
Mapping elements to CAMO 33
user_unassigned_from_department-1.0 33
assign_role_to_user-1.0 33
Sample messages 33
Message contents 34
Mandatory elements 34
Mapping elements to CAMO 34
role_assigned_to_user 34
unassign_role_from_user-1.0 35
Sample messages 35
Message contents 35
Mandatory elements 35
Mapping elements to CAMO 35
role_unassigned_from_user 36
assign_authority_to_user-1.0 36
Sample messages 36
Message contents 36
Mandatory elements 36
Mapping elements to CAMO 37
authority_assigned_to_user-1.0 37
unassign_authority_from_user-1.0 37
Sample messages 37
Message contents 38
Mandatory elements 38
Mapping elements to CAMO 38
authority_unassigned_from_user-1.0 38
set_all_authority_for_user-1.0 39
Sample messages 39
Message contents 39
Mandatory elements 39
Mapping elements to CAMO 40
all_authority_set_for_user-1.0 40
assign_skill_to_user-1.0 40

Sample messages....40
Message contents....40
Mandatory elements....40
Mapping elements to CAMO....41
skill_assigned_to_user-1.0....41
unassign_skill_from_user-1.0....41
Sample messages....41
Message contents....42
Mandatory elements....42
Mapping elements to CAMO....42
skill_unassigned_from_user-1.0....42
user-licenses-1.0....43
Sample messages....43
Message contents....43
Mandatory elements....44
Mapping elements to CAMO....45
user-licenses-response-1.0....45
user-planned-attendance-1.0....45
Sample messages....46
Message contents....46
Mandatory elements....46
Mapping elements to CAMO....48
user-planned-attendance-response-1.0....49

5 Integration framework....50

Operational and querying service messages....50
SOAP request message structure....52
SOAP response message structure....53
Notification service messages....55
Configuring Maintenix for the external system's web service....56
SOAP request message structure - notification service....58
SOAP response message structure - notifications....60
Setting up the IFS Maintenix Message Ordering Processor....60
Message streams....60
Adding new message streams....60
Configuring message order timing....61
Sequencing inbound messages....62

Managing integration messages in Maintenix..... 63
Viewing integration messages..... 63
Sending messages manually within Maintenix..... 65
Send message manually.....66
Edit and resend an integration message.....67
Understanding message errors.....67
Archive integration messages.....68
Logging..... 68

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.

CAMO Human Resources API

You can use the Human Resources (HR) API to load personnel information such as user details, skills, authorities, departments, licenses, work roles, and planned attendance from an external system. CAMO uses this personnel information to create and maintain an identity for each employee, which is used to schedule work assignments in CAMO, Production Planning and Control (PP&C), and the Long Range Planner (LRP).

2

Human Resources API scenarios

Creating a new user

Goal: To create a new user record in CAMO.

Summary: New personnel have joined your organization, and you must create user accounts for them in CAMO.

Preconditions:

- The user name must not exist in CAMO.
- The user code must not exist in CAMO.

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

Result: A new user is created in CAMO.

> **Note** Maintenix 8.3 SP11 restricts co-deployed Maintenix administrator users from creating new users via the HR API.

Updating a user's information

Goal: To update an existing user record in CAMO.

Summary: A user's personal details have changed, and you must update their user account in CAMO.

Preconditions:

- The user name must exist in CAMO.

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

Result: The user record is updated in CAMO.

Enabling a user

Goal: To unlock a user's account in CAMO.

Summary: A user account was previously locked, and you must unlock the user so that they can log in and use CAMO.

Preconditions:

- The user name must exist in CAMO.

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

Result: The specified user's account is unlocked.

Disabling a user

Goal: To lock a user's account in CAMO.

Summary: A user has left the company, and you want to lock their account so that they can no longer log in to CAMO. You may also wish to lock a user's account temporarily while you are making significant changes to the user's details.

Preconditions:

- The user name must exist in CAMO.

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

Result: The specified user's account is locked. That user cannot log in to CAMO or perform any actions within CAMO.

Deleting a user

Goal: To delete a user record from CAMO.

Summary: A user record was created in error, and you must delete that user's account from CAMO. If the user's account has a history in CAMO, then you cannot delete the user's account; you can only disable that user's account.

Preconditions:

- The user name must exist in CAMO.

Messaging: If the user account is not linked to historical activities in CAMO, then the external system sends an inbound *delete_user-1.0* message, and CAMO returns a *user_deleted-1.0* outbound message.

If the user account is linked to historical activities in CAMO, then the external system sends an inbound *delete_user-1.0* message, and CAMO returns a *user_disabled-1.0* outbound message.

Result: If the user account is not linked to historical activities in CAMO, then the specified account is deleted from CAMO. If the user account is linked to historical activities in CAMO, then the specified account is disabled in CAMO.

Assigning a user to a department

Goal: To assign a user to a department in CAMO.

Summary: New personnel have joined your organization, and you must assign the new user to a department in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The department must exist in CAMO.

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

Result: The user is assigned to the specified department.

Unassigning a user from a department

Goal: To unassign a user from a department in CAMO.

Summary: A user is currently assigned to a department, and you must unassign them.

Preconditions:

- The user name must exist in CAMO.
- The department must exist in CAMO.

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

Result: The user is unassigned from the specified department.

Assigning a role to a user

Goal: To assign a user to a role in CAMO.

Summary: New personnel have joined your organization, and you must assign the new user to a role in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The role must exist in CAMO.

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

Result: The user is assigned to the specified role.

Unassigning a role from a user

Goal: To unassign a user from a role in CAMO.

Summary: A user is currently assigned to a role, and you must unassign them.

Preconditions:

- The user name must exist in CAMO.
- The role must exist in CAMO.

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

Result: The user is unassigned from the specified role.

Assigning an authority to a user

Goal: To assign an authority to a user in CAMO.

Summary: New personnel have joined your organization, and you must assign an authority to the user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The authority must exist in CAMO.

Messaging: The external system sends an inbound *assign_authority_to_user-1.0* message. CAMO returns an *authority_assigned_to_user-1.0* outbound message.

Result: The authority is assigned to the specified user.

Unassigning an authority from a user

Goal: To unassign an authority from a user in CAMO.

Summary: An authority is currently assigned to a user, and you must unassign the authority.

Preconditions:

- The user name must exist in CAMO.
- The authority must exist in CAMO.

Messaging: The external system sends an inbound *unassign_authority_from_user-1.0* message. CAMO returns an *authority_unassigned_from_user-1.0* outbound message.

Result: The authority is unassigned from the specified user.

Assigning all authorities to a user

Goal: To assign all authorities to a user in CAMO.

Summary: In this situation you assign all authorities to a user in CAMO.

Preconditions:

- The user name must exist in CAMO.

Messaging: The external system sends an inbound *set_all_authority_for_user-1.0* message. CAMO returns an *all_authority_set_for_user-1.0* outbound message.

Result: All authorities are enabled for the specified user.

Unassigning all authorities from a user

Goal: To unassign all authorities from a user in CAMO.

Summary: A user has all authorities enabled, and you want to disable it.

Preconditions:

- The user name must exist in CAMO.

Messaging: The external system sends an inbound *set_all_authority_for_user-1.0* message. CAMO returns an *all_authority_set_for_user-1.0* outbound message.

Result: All authorities are disabled for the specified user.

Assigning a skill to a user

Goal: To assign a skill to a user in CAMO.

Summary: New personnel have joined your organization, and you must assign a skill to the user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The skill must exist in CAMO.

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

Result: The skill is assigned to the specified user.

Unassigning a skill from a user

Goal: To unassign a skill from a user in CAMO.

Summary: A skill is currently assigned to a user, and you must unassign the skill.

Preconditions:

- The user name must exist in CAMO.
- The skill must exist in CAMO.

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

Result: The skill is unassigned from the specified user.

Assigning a license to a user

Goal: To assign a license to a user in CAMO.

Summary: In this situation, you must assign a license to a user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The license must exist in CAMO.
- The license is not already assigned to the user.

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

Result: The license is assigned to the specified user.

Updating a user's license

Goal: To update license information for a user in CAMO.

Summary: In this situation, you must update the effective date or release number of a license, for a user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The license must exist in CAMO.
- The license is assigned to the user.

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

Result: The license information is updated for user.

Unassigning a license from a user

Goal: To unassign a license from a user in CAMO.

Summary: In this situation, you must assign a license to a user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The license must exist in CAMO.

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

Result: The license is unassigned from the specified user.

Suspending a user's license

Goal: To suspend a license for a user in CAMO.

Summary: In this situation, you must suspend a license for a user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The license must exist in CAMO.

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

Result: The specified license for the user is suspended.

Reactivating a user's suspended license

Goal: To reactivate a license for a user in CAMO whose license is suspended.

Summary: In this situation, you must reactivate a suspended license for a user in CAMO.

Preconditions:

- The user name must exist in CAMO.
- The license must exist in CAMO.

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

Result: The specified license for the user is reactivated.

Configuration parameters

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

HR 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.
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.

HR API role configuration parameters

Parameter	Description
API_HR	This parameter controls the ability to utilize all functions of the HR API. The integration user configured to send messages into the HR API must have this parameter enabled on at least one role.

For information about configuration parameters for Production Planning and Control, which may relate to the HR API functionality, see the CAMO Administration Guide.

Shift adjustment scenarios

CAMO receives an inbound user_planned_attendance message from the external HR system when there are temporary changes to a user's schedule. These changes are called *shift adjustments* in CAMO. A successfully received message creates, deletes, or edits shift adjustments in the user's schedule; it can also create a temporary crew assignment.

Results in CAMO

All changes made using the user-planned-attendance message can be observed on the **User Details** page, **Schedule** tab, in the **Schedule** box or the **Shift Adjustments** band.

Creating a shift adjustment for a user

Goal: Add a shift adjustment to a user's schedule in CAMO.

Summary: A user's schedule has a new variation.

Preconditions:

- The user name, location, and shift must exist in CAMO.
- When provided, the crew must exist in CAMO.

Messaging:

- The external system sends an inbound *user-planned-attendance-1.0* message.
- CAMO returns a *user-planned-attendance-response-1.0* outbound message.

Result: The shift adjustment is added to CAMO, on the **User Details** page, **Schedule** tab, **Shift Adjustments** box.

Updating a user's shift adjustment

Goal: Update an existing shift adjustment for a user in CAMO.

Summary: Either the skill or crew of a user's existing schedule has changed. To change any other element, you must delete the shift adjustment and add a new one.

Preconditions:

- The user name, location, and shift must exist in CAMO
- When provided, the crew must exist in CAMO.

Messaging:

- Change the value in either the `<labour-skill>` or the `<crew>` element.
- The external system sends an inbound *user-planned-attendance-1.0* message.
- CAMO returns a *user-planned-attendance-response-1.0* outbound message.

Result: The shift adjustment is updated in CAMO, on the **User Details** page, **Schedule** tab, **Shift Adjustments** box.

Deleting a shift adjustment

Goal: Delete a shift adjustment for a user in CAMO.

Summary: A user's schedule variation is no longer necessary.

Preconditions:

- The user name, location, and shift must exist in CAMO.

Messaging:

- Send a message with all the details of the shift adjustment to be deleted, and set the `<duration>` element to 0.0.
- The external system sends an inbound *user-planned-attendance-1.0* message.
- CAMO returns a *user-planned-attendance-response-1.0* outbound message.

Result: The shift adjustment no longer appears in CAMO.

Creating a temporary crew assignment

Goal: Add a temporary crew assignment to a user in CAMO.

Summary: A user joins a crew temporarily.

Preconditions:

- The user name and crew must exist in CAMO.

Messaging:

- Include a value in the `<crew>` element to create the temporary crew assignment.
- The external system sends an inbound *user-planned-attendance-1.0* message.
- CAMO returns a *user-planned-attendance-response-1.0* outbound message.

Result: The crew assignment appears on the **User Details** page, **Crew** column in the **Schedule** box and on the **Shift Adjustments** band.

3

Setting up the Human Resources API

The following chapter discusses the setup and configuration issues of the Human Resources (HR) API.

License requirement

To use the HR API, you must have the HR_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 HR API functionality as described in the table below.

HR 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.
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.

HR API role configuration parameters

Parameter	Description
API_HR	This parameter controls the ability to utilize all functions of the HR API. The integration user configured to send messages into the HR API must have this parameter enabled on at least one role.

For information about configuration parameters for Production Planning and Control, which may relate to the HR API functionality, see the CAMO Administration Guide.

4

Operational service messages

The following chapter provides detailed information for each of the Human Resources 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.

create_user-2.0

The create_user inbound message is sent by external systems to create an invoice in CAMO. The `create_user-2.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Creating_New_User.xml`: This message corresponds to the *Creating a new user* scenario. In this message, a new user is created.

Message contents

The `create_user` message can create new users, setting the following attributes:

- User Name
- Password
- First Name
- Middle Name
- Last Name
- HR Code
- E-mail
- Address
- Pay Method
- Hourly Salary
- Authorities
- Roles
- Skills
- Departments
- Licenses
- Disable (lock)

You cannot use the `create_user` message to update the attributes of an existing user.

You can use this message to add or assign authorities, roles, skills, departments, and licenses, or you can use the specific Human Resources API message for each item.

If you do not specify a password for the new user, the default user password is used. The default password for all new users is controlled by the DEFAULT_PASSWORD configuration parameter.

All new users are added to the group of unassigned users in your default organization in CAMO. If you want to assign a new user to any organization, you must complete this action within CAMO manually.

Mandatory elements

This section maps the mandatory elements for the `create_user` 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
	Creating a new user
<user-details> <username>	M U
<user-details> <password>	M
<user-details> <first_name>	M
<user-details> <middle_name>	
<user-details> <last_name>	M
<user-details> <hr_code>	M U
<user-details> <email_address>	
<user-details> <pay_method>	
<user-details> <hourly_cost>	
<authorities> <authority>	
<roles> <role>	
<skills> <skill>	
<departments> <department>	
<licenses> <license><license-definition>	
<licenses> <license> <effectivity-date>	
<licenses> <license> <release-number>	
<licenses> <license> <suspend>	
<disable>	

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<user-details> <username>	User Details page, User name field
<user-details> <password>	N/A
<user-details> <first_name>	User Details page, First Name field
<user-details> <middle_name>	User Details page, Middle Name field
<user-details> <last_name>	User Details page, Last Name field
<user-details> <hr_code>	User Details page, HR Code field
<user-details> <email_address>	User Details page, Email Address field
<user-details> <pay_method>	User Details page, Pay Method field
<user-details> <hourly_cost>	User Details page, Hourly Salary field
<authorities> <authority>	User Details page, Authorities tab
<roles> <role>	User Details page, Role tab
<skills> <skill>	User Details page, Skills tab
<departments> <department>	User Details page, Departments tab
<licenses> <license> <license-definition>	User Details page, Licenses tab
<licenses> <license> <effectivity-date>	User Details page, Licenses tab
<licenses> <license> <release-number>	User Details page, Licenses tab
<licenses> <license> <suspend>	User Details page, Licenses tab
<disable>	User Details page, Locked check box

user_created-2.0

The `user_created` outbound message is returned to the external system upon a successful create_user inbound message. The `user_created-2.0.xsd` schema contains annotations describing each element.

update_user-1.0

The `update_user` inbound message is sent by external systems to create an invoice in CAMO. The `update_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Updating_User_Information.xml`: This message corresponds to the *Updating a user's information* scenario and updates the user's information.

Message contents

The `update_user` message can update any of the following user attributes:

- Password
- First Name
- Middle Name
- Last Name
- HR Code
- E-mail Address
- Pay Method
- Hourly Salary

Within your message, if any of the previous elements contain no data, CAMO will not update the existing value. For example, if you send an `update_user` message with an empty `<middle_name>` element, CAMO will not delete the existing middle name of that user; the user's middle name will remain unchanged.

Mandatory elements

This section maps the mandatory elements for the `update_user` message in each supported scenario. The following keys are used:

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
	Updating a user's information
<username>	M D
<password>	
<first_name>	
<middle_name>	
<last_name>	
<hr_code>	U
<email_address>	
<pay_method>	D
<hourly_cost>	

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<password>	N/A
<first_name>	User Details page, First Name field
<middle_name>	User Details page, Middle Name field
<last_name>	User Details page, Last Name field
<hr_code>	User Details page, HR Code field
<email_address>	User Details page, Email Address field

Element	Corresponding entry in CAMO GUI
<pay_method>	User Details page, Pay Method field
<hourly_cost>	User Details page, Hourly Salary field

user_updated-1.0

The `user_updated` outbound message is returned to the external system upon a successful `update_user` inbound message. The `user_updated-1.0.xsd` schema contains annotations describing each element.

enable_user-1.0

The `enable_user` inbound message is sent by external systems to enable a user in CAMO. The `enable_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Enabling_User.xml`: This message corresponds to the *Enabling a user* scenario. In this message, a user's account is unlocked.

Message contents

The `enable_user` message unlocks the CAMO account of the user specified in the message. The `enable_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `enable_user` 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
	Enabling a user
<username>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field

user_enabled-1.0

The `user_enabled` outbound message is returned to the external system upon a successful `enable_user` inbound message. The `user_enabled-1.0.xsd` schema contains annotations describing each element.

disable_user-1.0

The `disable_user` inbound message is sent by external systems to disable a user in CAMO. The `disable_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provide examples of the supported scenarios:

- `Disabling_User.xml`: This message corresponds to the *Disabling a user* scenario. In this message, a user's account is locked.

Message contents

The `disable_user` message locks the CAMO account of the user specified in the message. The `disable_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `disable_user` 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
	Disabling a user
<username>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field

user_disabled-1.0

The `user_disabled` outbound message is returned to the external system upon a successful `disable_user` inbound message. The `user_disabled` outbound message is also returned when a `delete_user` inbound message is sent for a user that cannot be deleted.

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

delete_user-1.0

The *delete_user* inbound message is sent by external systems to create an invoice in CAMO. The `delete_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provide examples of the supported scenarios:

- `Deleting_User.xml`: This message corresponds to the *Deleting a user* scenario. In this message, a user's account is deleted.

Message contents

The `delete_user` message deletes the CAMO account of the user specified in the message. If the user's account has a history in CAMO, then you cannot delete the user's account. If you send a `delete_user` message for such a user, CAMO disables the user instead.

Mandatory elements

This section maps the mandatory elements for the `delete_user` 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
	Deleting a user
<username>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field

user_deleted-1.0

The `user_deleted` outbound message is returned to the external system upon a successful delete_user inbound message. The `user_deleted-1.0.xsd` schema contains annotations describing each element.

assign_user_to_department-1.0

The `assign_user_to_department` inbound message is sent by external systems to assign a user to a department in CAMO. The `assign_user_to_department-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provide examples of the supported scenarios:

- `Assigning_User_to_Department.xml`: This message corresponds to the *Assigning a user to a department* scenario. In this message, a user is assigned to a department.

Message contents

The `assign_user_to_department` message assigns a user to a valid department. The `assign_user_to_department` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `assign_user_to_department` 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
	Assigning a user to a department
<username>	M D
<department>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field.
<department>	User Details page, Departments tab.

user_assigned_to_department-1.0

The `user_assigned_to_department` outbound message is returned to the external system upon a successful `assign_user_to_department` inbound message. The `user_assigned_to_department-1.0.xsd` schema contains annotations describing each element.

unassign_user_from_department-1.0

The `unassign_user_from_department` inbound message is sent by external systems to remove a user from a department in CAMO. The `unassign_user_from_department-1.0.xsd` schema contains annotations describing each element.

Message contents

The `unassign_user_from_department` message unassigns a user from a department. The `unassign_user_from_department` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `unassign_user_from_department` 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
	Unassigning a user from a department
<username>	M D
<department>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<department>	User Details page, Departments tab

user_unassigned_from_department-1.0

The `user_unassigned_from_department` outbound message is returned to the external system upon a successful `unassign_user_from_department` inbound message. The `user_unassigned_from_department-1.0.xsd` schema contains annotations describing each element.

assign_role_to_user-1.0

The `assign_role_to_user` inbound message is sent by external systems to assign a role to the user in CAMO. The `assign_role_to_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Assigning_Role_to_User.xml`: This message corresponds to the *Assigning a role to a user* scenario. In this message, a user is assigned to a role.

Message contents

The `assign_role_to_user` message assigns a role to a user. The `assign_role_to_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `assign_role_to_user` 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
	Assigning a role to a user
<username>	M D
<role>	M D

Mapping elements to CAMO

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

Elements	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<role>	User Details page, Roles tab

role_assigned_to_user

The `role_assigned_to_user` outbound message is returned to the external system upon a successful `assign_role_to_user` inbound message. The `role_assigned_to_user-1.0.xsd` schema contains annotations describing each element.

unassign_role_from_user-1.0

The `unassign_role_from_user` inbound message is sent by external systems to remove the role from the user in CAMO. The `unassign_role_from_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Unassigning_Role_from_User.xml`: This message corresponds to the *Unassigning a role from a user* scenario. In this message, a user is unassigned from a role.

Message contents

The `unassign_role_from_user` message unassigns a role from a user. The `unassign_role_from_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `unassign_role_from_user` 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
	Unassigning a role from a user
<username>	M D
<role>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<role>	User Details page, Roles tab

role_unassigned_from_user

The `role_unassigned_from_user` outbound message is returned to the external system upon a successful `unassign_role_from_user inbound` message. The `role_unassigned_from_user-1.0.xsd` schema contains annotations describing each element.

assign_authority_to_user-1.0

The `assign_authority_to_user` inbound message is sent by external systems to assign an authority to a user in CAMO. The `assign_authority_to_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provides examples of the supported scenarios:

- `Assigning_Authority_to_User.xml`: This message corresponds to the *Assigning an authority to a user* scenario. In this message, an authority is assigned to a user.

Message contents

The `assign_authority_to_user` message assigns an authority to a user. The `assign_authority_to_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `assign_authority_to_user` 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 Assigning an authority to a user
<username>	M D
<authority>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<authority>	User Details page, Authorities tab

authority_assigned_to_user-1.0

The `authority_assigned_to_user` outbound message is returned to the external system upon a successful `assign_authority_to_user` inbound message. The `authority_assigned_to_user-1.0.xsd` schema contains annotations describing each element.

unassign_authority_from_user-1.0

The `unassign_authority_from_user` inbound message is sent by external systems to remove an authority from a user in CAMO. The `unassign_authority_from_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provides examples of the supported scenarios:

- `Unassigning_Authority_from_User.xml`: This message corresponds to the *Unassigning an authority from a user* scenario. In this message, an authority is unassigned from a user.

Message contents

The `unassign_authority_from_user` message unassigns an authority from a user. The `unassign_authority_from_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `unassign_authority_from_user` 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
	Unassigning an authority from a user
<username>	M D
<authority>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<authority>	User Details page, Authorities tab

authority_unassigned_from_user-1.0

The `authority_unassigned_from_user` outbound message is returned to the external system upon a successful `unassign_authority_from_user` inbound message. The `authority_unassigned_from_user-1.0.xsd` schema contains annotations describing each element.

set_all_authority_for_user-1.0

The `set_all_authority_for_user` inbound message is sent by external systems to assign to a user all authorities defined in CAMO. The `set_all_authority_for_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Assigning_all_Authorities_to_User.xml`: This message corresponds to the *Assigning all authorities to a user* scenario. In this message, the all-authority Boolean is enabled for a user.
- `Unassigning_all_Authorities_from_User.xml`: This message corresponds to the *Unassigning all authorities from a user* scenario. In this message, the all-authority Boolean is disabled for a user.

Message contents

The `set_all_authority_for_user` message toggles the all-authority Boolean for a user. The `set_all_authority_for_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `set_all_authority_for_user` 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	
	Assigning all authorities to a user	Unassigning all authorities from a user
<username>	M D	M D
<all_authority>	M	M

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field.
<all_authority>	User Details page, All Authority check box

all_authority_set_for_user-1.0

The `all_authority_set_for_user` outbound message is returned to the external system upon a successful `set_all_authority_for_user` inbound message. The `all_authority_set_for_user-1.0.xsd` schema contains annotations describing each element.

assign_skill_to_user-1.0

The `assign_skill_to_user` inbound message is sent by external systems to assign a skill to the user. The `assign_skill_to_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provides examples of the supported scenarios:

- `Assigning_Skill_to_User.xml`: This message corresponds to the *Assigning a skill to a user* scenario. In this message, a skill is assigned to a user.

Message contents

The `assign_skill_to_user` message assigns a skill to a user. The `assign_skill_to_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `assign_skill_to_user` 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
	Assigning a skill to a user
<username>	M D
<skill>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<skill>	User Details page, Skills tab

skill_assigned_to_user-1.0

The `skill_assigned_to_user` outbound message is returned to the external system upon a successful `assign_skill_to_user` inbound message. The `skill_assigned_to_user-1.0.xsd` schema contains annotations describing each element.

unassign_skill_from_user-1.0

The `unassign_skill_from_user` inbound message is sent by external systems to remove a skill from a user in CAMO. The `unassign_skill_from_user-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample message provides examples of the supported scenarios:

- `Unassigning_Skill_from_User.xml`: This message corresponds to the *Unassigning a skill from a user* scenario. In this message, a skill is unassigned from a user.

Message contents

The `unassign_skill_from_user` message unassigns a skill from a user. The `unassign_skill_from_user` message can perform no other functions.

Mandatory elements

This section maps the mandatory elements for the `unassign_skill_from_user` 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
	Unassigning a skill from a user
<username>	M D
<skill>	M D

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<username>	User Details page, User name field
<skill>	User Details page, Skills tab

skill_unassigned_from_user-1.0

The `skill_unassigned_from_user` outbound message is returned to the external system upon a successful `unassign_skill_from_user` inbound message. The

`skill_unassigned_from_user-1.0.xsd` schema contains annotations describing each element.

user-licenses-1.0

The `user-licenses` inbound message is sent by external systems to assign a license to a user in CAMO. The `user-licenses-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Assigning_License_to_User.xml`: This message corresponds to the *Assigning a license to a user* scenario. In this message, a license is assigned to a user.
- `Updating_User_License.xml`: This message corresponds to the *Updating a user's license* scenario. In this message, a user's license is updated.
- `Suspending_User_License.xml`: This message corresponds to the *Suspending a user's license* scenario. In this message, a user's license is suspended.
- `Reactivating_User_License.xml`: This message corresponds to the *Reactivating a user's suspended license* scenario. In this message, a user's license is reactivated.
- `Unassigning_License_from_User.xml`: This message corresponds to the *Unassigning a license from a user* scenario. In this message, a license is unassigned from a user.

Message contents

The `user-licenses` message can assign, update, suspend, unsuspend, or unassign licenses for CAMO users.

In each `user-licenses` inbound message, you must identify the user by specifying only one of the following elements:

- <user-name>
- <hr-code>
- <internal-identifier>

You can perform multiple license operation for a single user in one message.

When the <update> element is present in the message, the specified license is assigned or updated for the user. If the license is not already assigned to the user, then the license is assigned to the user, and the effective date and release number provided in the message are recorded. If the license is already assigned to the user, then the effective date and release number provided in the message are updated.

When the <suspend> element is present in the message, the specified license is suspended for that user.

When the <unsuspend> element is present in the message, the specified license is unsuspended for that user.

Mandatory elements

This section maps the mandatory elements for the `user-licenses` 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				
	Assigning a license to a user	Updating a user's license	Suspending a user's license	Reactivating a user's suspended license	Unassigning a license from a user
<user> <user-name>	M* D	M* D	M* D	M* D	M* D
<user> <hr-code>	M* D	M* D	M* D	M* D	M* D
<user> <internalidentifier>	M* D	M* D	M* D	M* D	M* D
<update> <user-licenses>	M D	M D			
<update> <effectivitydate>					
<update> <releasenumber>					
<unassign>					M D
<suspend> <licensedefinition>			M D		
<suspend> <reason>					
<unsuspend> <licensedefinition>				M D	
<unsuspend> <reason>					

*The <user-name>, <hr-code>, or <internal-identifier> is mandatory in this scenario.

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<user> <user-name>	User Details page, User name field
<user> <hr-code>	User Details page, HR Code field
<user> <internal-identifier>	Does not appear in the UI
<update> <license-definition>	User Details, Licenses tab, Active Licenses box
<update> <effectivity-date>	User Details, Licenses tab, Active Licenses box
<update> <release-number>	User Details, Licenses tab, Active Licenses box
<unassign>	User Details, Licenses tab, Inactive Licenses box
<suspend> <license-definition>	
<suspend> <reason>	User Details, Licenses tab, Inactive Licenses box
<unsuspend> <license-definition>	User Details, Licenses tab, Active Licenses box
<unsuspend> <reason>	User Details, Licenses tab, Active Licenses box

user-licenses-response-1.0

The `user-licenses-response` outbound message is returned to the external system upon a successful `user-licenses inbound` message. The `user-licenses-response-1.0.xsd` schema contains annotations describing each element.

user-planned-attendance-1.0

The `user-planned-attendance` inbound message is sent by external systems to update attendance information about a user in CAMO. The `user-planned-attendance-1.0.xsd` schema contains annotations describing each element.

Sample messages

The following sample messages provide examples of the supported scenarios:

- `Adding_New_Planned_Attendance.xml`: This message corresponds to the *Adding a new planned attendance* for a user scenario. In this message, a new planned attendance is added for a user.
- `Updating_User_Planned_Attendance.xml`: This message corresponds to the *Updating a user's planned attendance* scenario. In this message, a user's planned attendance is updated.

Message contents

The `user-planned-attendance message` can add or update shift adjustments for CAMO users. This information enables CAMO Production Planning and Control (PP&C) and line and hangar capacity to plan maintenance activities.

In each `user-planned-attendance` inbound message, you must identify the user by specifying any of the following elements:

- <user-name>
- <hr-code>
- <internal-identifier>

To determine whether to create a new shift adjustment, or update an existing one, CAMO examines the combination of <user> (which includes either <username>, <hr-code>, or <internal-identifier>), <date>, <location> and <shift-code>. If the combination of entries identifies an existing shift adjustment in CAMO, then the `user-planned-attendance` message will update the shift adjustment, otherwise, a new shift adjustment is created.

When you are updating a shift adjustment, you can update the labor skill or crew. You cannot update any other information.

When you are assigning a new shift code to a user, the duration must be a positive value. When you are updating a shift adjustment, if a user's shift code already exists, and the duration specified in the message is positive, then the user shift code is updated; otherwise, if the duration is 0.0 or a negative value, then the shift code is removed.

In the `user-planned-attendance` message, dates are specified as DD-MMM-YYYY (e.g, 30-NOV-2013), hours are specified to one decimal of precision, and times are specified in the 24-hour clock to one decimal of precision (e.g., 11:30 PM is 23.5).

If you do not provide the labor skill in the message, the default labor skill for that user is assigned.

Mandatory elements

This section maps the mandatory elements for the `user-planned-attendance` 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			
	Adding a shift adjustment for a user	Updating a user's shift adjustment	Deleting a user's shift adjustment	Creating a temporary crew assignment
<user> <user-name>	M* D	M* D	M* D	M* D
<user> <hr-code>	M* D	M* D	M* D	M* D
<user> <internal-identifier>	M* D	M* D	M* D	M* D
<attendance-information> <date>	M	M	M	M
<attendance-information> <location> <location-code>	M D	M D	M D The location code must be linked to the crew department.	M D
<attendance-information> <location> <location-name>	Not mandatory. Invalid entries will be ignored.			
<attendance-information> <time-details> <shift-code>	M** D	M** D	M** D	M** D
<attendance-information> <time-details> <working-period> <start-hour>	M** D	M** D	M** D	M** D
<attendance-information> <time-details> <working-period> <duration>	M** D	M** D	M** D	M** D

Element	Scenario			
	Adding a shift adjustment for a user	Updating a user's shift adjustment	Deleting a user's shift adjustment	Creating a temporary crew assignment
<attendance-information> <time-details> <working-period> <working-hours>	M** D	M** D	M** D	M** D
<attendance-information> <line-capacity>	Cannot be null. When included, must be `true` or `false`.			
<attendance-information> <labour-skill>	M D	M D Can be updated in this message.	D	D
<attendance-information> <crew>	D	D Can be updated in this message.	D	D Not a mandatory element, but is required to assign a temporary crew.

*The <user-name> OR <hr-code> OR <internal-identifier> is mandatory.

**Under <attendance-information>, either <shift-code>, <working-period>, or both are accepted. If <working-period> is provided, all sub-elements are mandatory.

Mapping elements to CAMO

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

Element	Corresponding entry in CAMO GUI
<user><user-name>	User Details page, User name field
<user><hr-code>	User Details page, HR Code field

Element	Corresponding entry in CAMO GUI
<user><internal-identifier>	Does not appear in UI
<attendance-information><date>	Does not appear in UI
<attendance-information><location><location-code>	User Details page, Schedule box, Location column
<attendance-information><location><location-name>	Does not appear in UI
<attendance-information><time-details><shift-code>	User Details page, Schedule box, Shift column
<attendance-information><time-details><workingperiod><start-hour>	Does not appear in UI
<attendance-information><time-details><workingperiod><duration>	Does not appear in UI
<attendance-information><time-details><workingperiod><working-hours>	Does not appear in UI
<attendance-information><line-capacity>	Does not appear in UI
<attendance-information><labour-skill>	User Details page, Schedule box, Primary Skill column
<attendance-information><crew>	User Details page, Schedule box, Crew

user-planned-attendance-response-1.0

The `user-planned-attendance-response` outbound message is returned to the external system upon a successful `user-planned-attendance` inbound message. The `user-planned-attendance-response-1.0.xsd` schema contains annotations describing each element.

5

Integration framework

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

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

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

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

External systems can interact with Maintenix using the following methods:

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

Operational and querying service messages

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

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

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

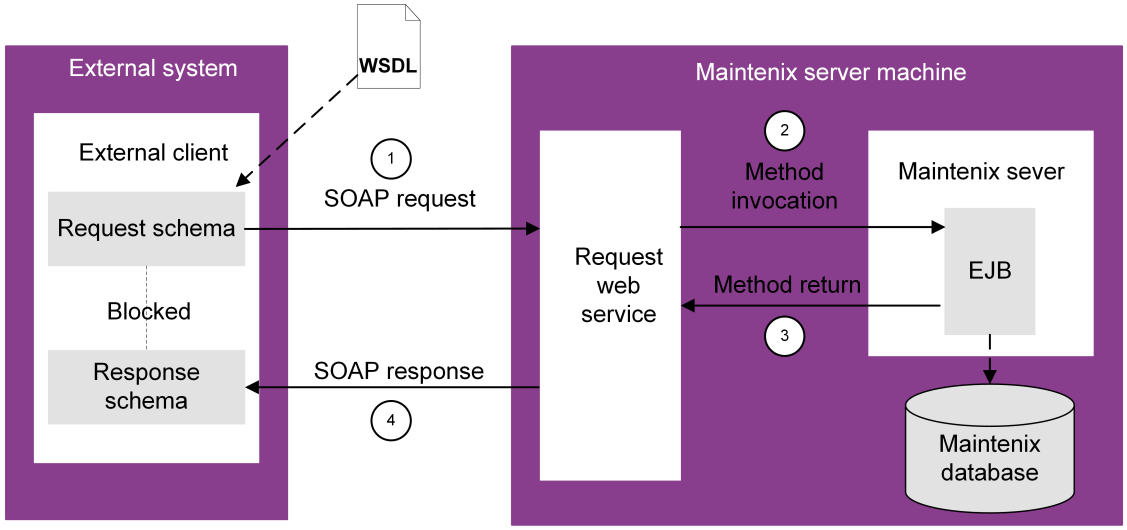
The same Web service is called, regardless of the data that is passed.

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

See related topics on the format of the requests.

Integration scenario

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



Operational and querying message web service

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

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

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

SOAP request message structure

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

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

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

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

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

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

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

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

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

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

SOAP response message structure

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

The <response> element has the following attributes:

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

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

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

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

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

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

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

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

</soapenv:Envelope>
```

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

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

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

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

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

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

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

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

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

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

Notification service messages

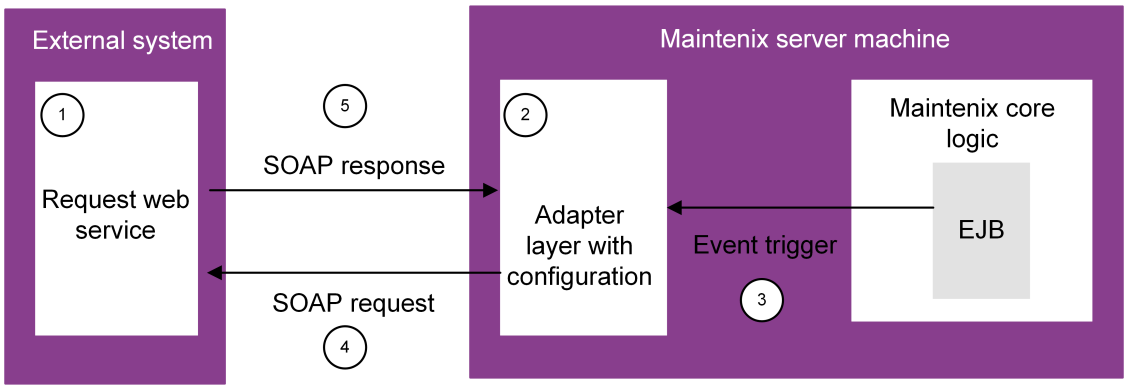
Notification service messages are used to alert an external system that something happened within Maintenix. For example an external system can be notified whenever a fault is raised, or a purchase order is updated.

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

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

Integration scenario

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



Notification message web service

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

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

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

Configuring Maintenix for the external system's web service

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

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

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

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

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

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

SOAP request message structure - notification service

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

The <response> element has the following attributes:

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

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

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

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

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

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

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

         <work-package
```

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

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

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

</soapenv:Envelope>
```

SOAP response message structure - notifications

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

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

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

Setting up the IFS Maintenix Message Ordering Processor

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

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

Message streams

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

Adding new message streams

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

Parameters to be set in the INT_MESSAGE_ORDER

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

Configuring message order timing

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

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

Configuration Parameters

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

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

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

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

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

Sequencing inbound messages

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

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

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

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

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

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

Managing integration messages in Maintenix

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

Viewing integration messages

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

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

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

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

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

Inbound tab

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

Outbound tab

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

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

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

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

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

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

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

Sending messages manually within Maintenix

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

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

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

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

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

Send message manually

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

Prerequisite

You must have the ACTION_SEND_MESSAGE permission enabled.

Steps

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

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

3. Click **OK**.

Edit and resend an integration message

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

Prerequisite

You must have the ACTION_RESEND_MESSAGE permission.

Steps

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

Understanding message errors

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

Maintenix tracks the following types of errors:

- Integration framework errors
- Business logic errors

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

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

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

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

Archive integration messages

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

Prerequisite

You must have the ACTION_ARCHIVE_MESSAGE permission enabled.

Steps

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

Logging

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

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

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

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

ABOUT IFS

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

#forthechallengers

ifs.com

WHERE WE ARE

AMERICAS

+1 888 437 4968

ASIA PACIFIC

+65 63 33 33 00

EUROPE EAST

+48 22 577 45 00

EUROPE CENTRAL

+49 9131 77 340

UK & IRELAND

+44 1494 428 900

FRANCE, BENELUX AND IBERICA

+33 3 89 50 72 72

MIDDLE EAST AND AFRICA

+971 4390 0888

NORDICS

+46 13 460 4000

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