IFS Aviation Maintenance

External Maintenance API Guide

ICAM 26.2



Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

1 Introduction....4
About CAMO Integrations....4
The External Maintenance API (EMA)....5
Definitions....5
Assumptions....6
Limitations....6

2 EMA setup....7

3 Processing messages....8
Creating historical records....9
Creating and updating work packages and repair orders....9
Completing work packages and repair orders....10
Creating and updating tasks....11
Creating tasks against systems....12
Task revisions using EMA, CAMO, and maintenance programs....12
Task revisions using EMA and CAMO....12
Task revisions using maintenance programs....13
Creating and updating faults....13
Corrective actions....14
Troubleshooting tasks....15
Adding resulting events....15
Linking tasks....16
Identifying main inventory....16
Creating and updating part requirements....16
Auto-creation of installed inventory....17
Updating installed inventory usage....18
Processing removed parts....18
Processing replacement of installed parts....18
Creating and updating labor requirements....18
Creating and updating measurements....19
Completing tasks and faults....20
Creating sign off requirements....20
Creating usage snapshots....21
Creating usage snapshots for work packages and repair orders....21

Creating usage snapshots for tasks and faults..... 22
Handling erroneous messages..... 22

4 Integration framework..... 23
Operational and querying service messages..... 23
SOAP request message structure..... 25
SOAP response message structure..... 26
Notification service messages..... 28
Configuring Maintenix for the external system's web service..... 29
SOAP request message structure - notification service..... 31
SOAP response message structure - notifications..... 33
Setting up the IFS Maintenix Message Ordering Processor..... 33
Message streams..... 33
Adding new message streams..... 33
Configuring message order timing..... 34
Sequencing inbound messages..... 35
Managing integration messages in Maintenix..... 36
Viewing integration messages..... 36
Sending messages manually within Maintenix..... 38
Send message manually..... 39
Edit and resend an integration message..... 40
Understanding message errors..... 40
Archive integration messages..... 41
Logging..... 41

1 Introduction

About CAMO Integrations

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

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

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

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

Note Not all APIs support all messaging scenarios.

Depending on the integration, there are different framework options.

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

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

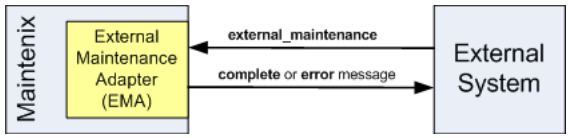
Some newer integrations are accessed by sending JSON requests to a publicly exposed HTTP endpoint.

The External Maintenance API (EMA)

The External Maintenance API (EMA) enables data transfer between CAMO and an external system to assist in the planning and execution of Line Maintenance, Heavy Maintenance and Shop Maintenance.

As shown in the following figure, the EMA processes `external_maintenance` messages received from the external system and returns a successful completion message or an error message.

CAMO EMA Interface



The EMA uses the messaging schema provided in your installation package.

Note

The element names in the schema are case sensitive. IFS coding standards require the element names to be lower case.

Definitions

Fault

An entity reporting that an aircraft or component is not working as expected. Tasks that fix faults are known as *corrective actions*.

Maintenance Event

A task, a fault, a work package, or a repair order.

New Maintenance Event

A maintenance event that does not exist in CAMO yet.

Historical Maintenance Event

A maintenance event that is complete.

Active Work Package / Repair Order:

A WP/RO that has a `COMMIT` or `IN WORK` status.

Assumptions

1. The external system orders the external maintenance messages, before sending them to the EMA.
2. Tracked inventory can be uniquely identified by the serial number, part number and manufacturer code or manufacturer name.
3. Systems can be uniquely identified by the parent assembly and the sub configuration slot.
4. A baseline task can be uniquely identified by the task code and the assembly configuration slot.

Limitations

1. The EMA does not update configuration changes for historical messages.
2. The EMA does not support unassigning tasks from WP/RO.
3. Using EMA, only the `MINOR` faults can be deferred.
4. The EMA does not support specifying deadlines for faults.
5. Processing replacement of installed parts may require manual update of the serial number (see *Processing replacement of installed parts*).

2

EMA setup

The following configuration parameters must be set in the utl_config_parm table for the EMA to function properly:

SIGNOFF_HR_CD: The default HR_CD used for processing messages, when the HR_CD is not specified.

AUTO_CALC_USAGE_SNAPSHOT: If this parameter is set to TRUE and no configuration snapshot is provided in the `external_system` message, then the EMA attempts to get the usage totals from a flight or usage record that is completed just before the completion date of the fault or task specified in the message. If it cannot find such a flight or usage record, then it uses the current usage totals in the INV_CURR_USAGE table.

AUTO_CREATE_PART: If this parameter is set to TRUE and the part number doesn't exist in the database and the message contains no serial number information, then the EMA assumes that the part is a consumable part. Hence, it creates and activates the part automatically.

SHOW_PROCESS_TIME: If this parameter is set to TRUE, the processing time for the message is calculated and added to the info message in the log file, to enable measuring the API performance.

To successfully import data provided by the external system into CAMO, the following configuration parameters must be set to WARN or OK:

- RELEASE_LOW_SEV_OPEN_FAULT
- RELEASE_AOG_FAULT
- RELEASE_MISSING_COMPONENT
- RELEASE_NEAR_DUE_FAULT
- RELEASE_NEAR_DUE_TASK
- RELEASE_OVERDUE_FAULT
- RELEASE_OVERDUE_TASK

3

Processing messages

All maintenance events in CAMO are associated with a Work Package (WP) or a Repair Order (RO). CAMO groups tasks into work packages and repair orders as follows:

- Tasks performed on aircraft are grouped into work packages.
- Tasks performed on components are grouped into repair orders.

The EMA provides the following capabilities:

- Creating repair orders (active or historical)
- Updating repair orders
- Creating work packages (active or historical)
- Updating scheduled work packages
- Creating tasks (either ad-hoc, or based on a task definition) against the following:
  - A work package (active or historical)
  - A repair order (active or historical)
  - An inventory (including systems)
- Assigning existing tasks to the following:
  - A work package (active or historical)
  - A repair order (active or historical)
- Updating installed or removed parts while creating a new task:
  - Creating new serialized parts
  - Updating usage totals
- Updating parts used, labor hours spent, task measurements, and skills used for a task (new or historical)
  - All labor requirements are signed-off by a default system user.
- Completing tasks (new or historical)
- Completing work packages (active or historical)
- Completing repair orders (active or historical)

Creating historical records

When a message is received by the EMA, the `non_historical_bool` (the first-level child element of the `external_maintenance` message) is used to determine whether the data is historical or not. Since non-historical data triggers configuration changes in CAMO, data integrity may be lost if non-historical events are received out of sequence.

```
<external_maintenance ...
<non_historical_bool>false</non_historical_bool>
...
```

Creating and updating work packages and repair orders

To create a work package (WP) or repair order (RO), the external system must use a unique external key for the WP or RO.

When CAMO receives a message with a WP/RO number, it checks the existing WPs/ROs in the IN WORK or COMMIT status. If it cannot find such a WP/RO, CAMO creates a new one and updates it with the information in the message.

If the work package is in the COMMIT status, CAMO also starts the work package.

The following is a sample of an `external_maintenance` message for creating a repair order:

```
<external_maintenance ...
   <non_historical_bool>true</non_historical_bool>
   <task_class_cd>RO</task_class_cd>
   <task_subclass_cd>SV</task_subclass_cd>
   <event_sdesc>EmaWO-3-3-001 - Repair Order with no task</
event_sdesc>
   <work_type_cd>TURN</work_type_cd>
   <task_originator_cd>CUSTOMER</task_originator_cd>
   <task_ref_sdesc>External Reference</task_ref_sdesc>
   <issued_dt>2005-08-14 15:10:48</issued_dt>
   <ro_ref_sdesc>The RO Number</ro_ref_sdesc>
   <wo_ref_sdesc>The WorkPackage Number</wo_ref_sdesc>
   <event_ldesc>The WorkPackage Description</event_ldesc>
   <contact_info_sdesc>My Contact Information</contact_info_sdesc>
   <doc_ref_sdesc>EmaWO-3-3-001</doc_ref_sdesc>
   <sched_start_dt>2005-08-28 09:00:00</sched_start_dt>
   <actual_start_dt>2005-08-28 09:00:00</actual_start_dt>
   <sched_end_dt>2005-08-28 17:00:00</sched_end_dt>
   <actual_end_dt>2005-08-28 17:00:00</actual_end_dt>
   <ext_key_sdesc>TSEmaWO-3-3-001</ext_key_sdesc>
   <hr_cd>amac</hr_cd>
   <inventory>
      <serial_no_oem>EMALOOSEENGINE</serial_no_oem>
      <part_no_oem>ACME738-1-1B</part_no_oem>
```

```
      <manufacturer>
         <manufact_cd>ACME</manufact_cd>
         <manufact_name>ACME Company</manufact_name>
      </manufacturer>
      <ac_reg_cd/>
   </inventory>
   <location>
      <loc_cd>THEAIRPORT/SHOP</loc_cd>
      <loc_name>The Airport/Shop</loc_name>
   </location>
   </signoff_requirements>
   <usages>
      <usage_snapshot>
         <data_type_cd>HOURS</data_type_cd>
         <tsn_qt>50</tsn_qt>
         <tso_qt>0</tso_qt>
         <tsi_qt>0</tsi_qt>
         <assmbl_tsn_qt>2800</assmbl_tsn_qt>
         <assmbl_tso_qt>0</assmbl_tso_qt>
      </usage_snapshot>
   </usages>
</external_maintenance>
```

Completing work packages and repair orders

CAMO marks a WP as complete only when the actual end date is provided by the external system. If the actual end date is not provided, the work package remains open even after all the tasks in it are complete.

A non-historical work package cannot be completed until all active tasks against the main inventory and its sub-components are unassigned from the current work package.

A non-historical work package cannot be completed if it contains open faults. All faults must be completed or deferred. If the troubleshooting task or any corrective sub-tasks are incomplete, a fault cannot be completed. Only faults with a severity code of MINOR can be deferred using the EMA.

The following sample of an external_maintenance message displays the format of the actual end date element:

```
<external_maintenance ...
...
   <sched_start_dt/>
   <actual_start_dt>2007-07-05 14:00:00</actual_start_dt>
   <sched_end_dt/>
   <actual_end_dt>2007-07-05 15:00:00</actual_end_dt>
...
```

Creating and updating tasks

Tasks are represented as children of WPs or ROs. The EMA supports creating ad-hoc tasks, or tasks created from task definitions in the following categories:

- Corrective Action
- Inspection
- Installation
- Overhaul
- Part Transformation (baseline only)
- Removal
- Replacement
- Servicing
- Unknown

All tasks must have a unique external key, which is used by the external system for identification purposes.

All tasks and faults are processed sequentially in the order in which they are specified in the message. You can use the task_ord element to specify a different order for tasks.

When the EMA receives a message containing a non-historical baseline task, the EMA checks whether the task exists in CAMO already.

When checking for an existing non-historical baseline task, CAMO:

1. Searches for a task with a matching external key.
   - The task must be in the WP or loose.
   - The task cannot be a forecast task or a historical task.

   > **Note** The task can be completed if the external ID matches the task, regardless of the task revision specified in the EMA message and the task revision in CAMO.

2. Searches for a task created using the specified task definition, or a task created against the specified inventory.
   a. Searches for the task definition.
   b. Searches for the inventory record.

When checking for an existing ad hoc task, CAMO:

- Searches for a task with a matching external key. If found, validate the task name and inventory using the following information in the message:
  - The task name in the EVENT_SDESC element.
  - The inventory in the INVENTORY element.

If the task does not exist in CAMO, a new task is created. If the task does exist in CAMO, the task details are updated with the information contained in the message. If the existing task is found in more than one work package, then the work package that is due first is used. If an open and unassigned existing task is found, it is re-assigned to the current work package.

Creating tasks against systems

You can create ad hoc and baseline tasks against a system. To uniquely identify a system in CAMO, the external_maintenance message must:

- Uniquely identify the parent inventory by providing part number, serial number, and manufacturer.
- Specify the configuration slot for the system, which must be in the inventory hierarchy that start from the parent inventory for the system. DO NOT specify a configuration slot when identifying a tracked component.

Task revisions using EMA, CAMO, and maintenance programs

CAMO handles task revisions differently depending on whether or not your organization uses maintenance programs.

Task revisions using EMA and CAMO

You can specify a previous task revision when creating baseline tasks by entering a revision date in the baseline task element. This is may be useful when creating historical data.

The revision date element is optional. If you do not specify a revision date, the current active task revision is selected.

The following sample of an `external_maintenance` message displays the format of the revision date element:

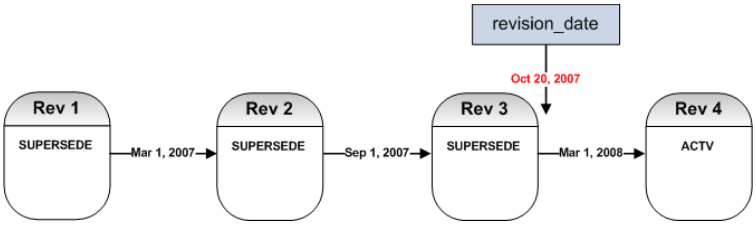
```
...
<task>
...
   <task_baseline>
      <task_cd>SUBTASK3</task_cd>>
      <assmbl_cd>738-1</assmbl_bom_cd>
      <assmbl_bom_cd>738-1</assmbl_bom_cd>
      <revision_date>2007-10-20 10:32:00</revision_date>
   </task_baseline>
...
```

When the revision date is specified, the EMA examines the revision history of the task definition. The EMA selects the latest revision occurring prior to the revision date.

In the following situations, a EMA will not return the latest revision occurring prior to the revision date:

- The revision date occurs prior to first revision of the task definition. In this case, the EMA will select the first revision of the task definition.
- The retrieved task definition is OBSOLETE. In this case, the EMA will select the previous revision of the task definition.

In the following diagram, the revision date specified in the message is Oct 20, 2007. The EMA looks up the latest revision occurring before Oct 20, 2007. In this example, Revision 3 is selected as it was active between Sept. 1, 2007 and Mar. 1, 2008.



Task revisions using maintenance programs

When CAMO receives an EMA message, it will check to see whether or not the task is part of a maintenance program. If it is part of a maintenance program, it will select that task revision. If not, it will determine the revision as per the *Task revisions using EMA and CAMO* section.

If the maintenance program task has a temporary revision issued, the temporary revision will be used.

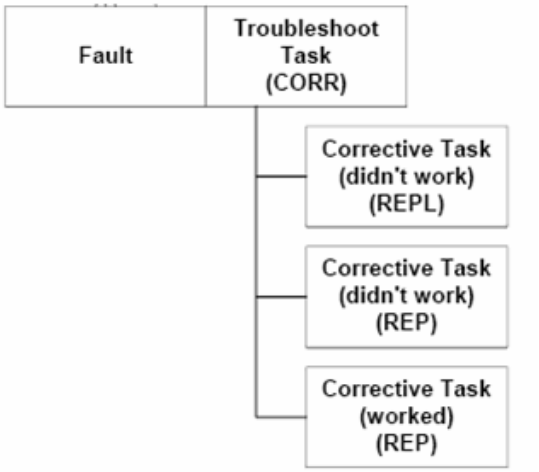
Creating and updating faults

A fault reports that an aircraft or component is not working as expected. The maintenance task that rectifies a fault is known as a Corrective Action. EMA allows importing faults without an associated corrective action into CAMO by creating a default troubleshooting task, with the task class code of CORR, to satisfy CAMO business logic. This situation typically occurs when an `external_maintenance` message contains a fault, with the event status code of No Fault Found.

All tasks and faults are processed sequentially in the order in which they are specified in the message.

Corrective actions

You can use the corrective_actions element to identify a *fault-to-task* dependency. Each fault in CAMO is linked to a troubleshooting task of type CORR, which can have several corrective sub-tasks representing possible fixes for the fault.



This structure must be encoded as follows:

```
<fault>
...
   <troubleshooting_task>
   </troubleshooting_task>
   <corrective_actions>
      <task></task>
      <task></task>
   </corrective_actions>
</fault>
```

The corrective_actions element is also optional. If more that one task child is provided as subelements of corrective_actions, the corr_fix_bool must be set to TRUE for the task that fixed the problem.

When CAMO receives an external maintenance message containing a fault, it performs the following validation:

- If the fault exists and the work package of the fault and the work package provided in the EMA message match, then CAMO updates the fault. At present, EMA only supports updating the fault description and the troubleshooting tasks.
- If the fault exists (based on its external reference key) but the fault's work package and the work package provided as part of the XML message do not match then CAMO responds with an error message as follows:

[MXERR-nnnnn] The external reference key {external reference} specified for the fault {fault name} is not unique. Please use a different external reference key.

- If the fault doesn't exist, CAMO creates the fault

Troubleshooting tasks

The troubleshooting_task element is optional as a default is automatically created by CAMO when the fault is created. The troubleshooting_task element should only be provided to override the default attributes assigned by CAMO.

Adding resulting events

You can add resulting events to historical and non-historical faults. Resulting events are optional. The only preresiquite is that any resulting event code that you add must already exist in CAMO.

This structure of the message must be encoded as follows:

```
...
<fault>
   <event_status_cd>CFCERT</event_status_cd>
   <event_ldesc>Test Fault</event_ldesc>
   <fail_sev_cd/>
   <sdr_bool/>
   <event_sdesc>Fault for Test Add Resulting Events</event_sdesc>
   <found_on_date>2008-03-08 12:00:00</found_on_date>
   <actual_end_dt>2008-03-07 15:00:00</actual_end_dt>
   <ext_key_sdesc>test Add Resulting Events</ext_key_sdesc>
   <inventory>
      <serial_no_oem>5023222</serial_no_oem>
      <part_no_oem>HSEE-167</part_no_oem>
         <manufacturer>
            <manufact_cd>ACME</manufact_cd>
            <manufact_name>ACME Aerospace</manufact_name>
         </manufacturer>
         <ac_reg_cd/>
   </inventory>
   <resulting_events>
      <resulting_event_cd>ABT</resulting_event_cd>
      <resulting_event_cd>CNX</resulting_event_cd>
   </resulting_events>
</fault>
...
```

Linking tasks

All task relations are specified as parent element—child element relationships in the `external_maintenance` message. The parent is known as the initiating event and the child as the resultant event.

A *task-to-task* dependency is represented by a recursive relationship, as a task element that has a child task element of type DEPT.

To link a fault directly to a task it was found, you can create a found fault for a task or fault. For information on how to raise faults, see the *CAMO User Guide*.

Identifying main inventory

EMA can identify the main inventory (component or aircraft) in a message, in several ways:

- An aircraft can be identified by its tail number.
- An aircraft or component can be identified by part number and serial number, if the combination is unique.
- In case of duplicates, the optional manufacturer segment is also used.
- If no manufacturer code is present, and exactly one of the resolved duplicates is of type ACFT, it is chosen over any other inventory type.
- If duplicates cannot be resolved, an exception is thrown and processing fails.
- If the inventory cannot be found, an exception is thrown and processing fails.

Creating and updating part requirements

CAMO uses only the part requirements specified in the `external_maintenance` message. For the baseline task, the EMA deletes any default part requirements defined from the baseline.

The following sample of an `external_maintenance` message displays the format of the part requirements section:

```
...
<part_requirements>
   <part_requirement>
      <rmvl_part>
         <assmbl_cd>738-1</assmbl_cd>
         <assmbl_bom_cd>72-30-09</assmbl_bom_cd>
         <eqp_pos_cd>1.1</eqp_pos_cd>
         <serial_no_oem>EMATEMP3</serial_no_oem>
         <part_no_oem>6079T99G09</part_no_oem>
         <manufacturer>
            <manufact_cd>ACME</manufact_cd>
```

```
            <manufact_name>ACME Company</manufact_name>
         </manufacturer>
         <quantity>1</quantity>
         <part_note/>
         <remove_reason_cd>ACCESS</remove_reason_cd>
      </rmvl_part>
      <inst_part>
         <assmbl_cd>738-1</assmbl_cd>
         <assmbl_bom_cd>72-30-09</assmbl_bom_cd>
         <eqp_pos_cd>1.1</eqp_pos_cd>
         <serial_no_oem>EMA3008</serial_no_oem>
         <part_no_oem>6079T99G09</part_no_oem>
         <part_group_cd/>
         <manufacturer>
            <manufact_cd>ACME</manufact_cd>
            <manufact_name>ACME Company</manufact_name>
         </manufacturer>
         <quantity>1</quantity>
         <part_note/>
      </inst_part>
   </part_requirement>
</part_requirements>
...
```

Auto-creation of installed inventory

When an ad hoc task requires a serialized part, the serial number of the part is provided in the EMA message. If the serialized part does not exist in CAMO already, the part is created automatically, and any usage is updated.

When a baseline task requires a serialized part that does not exist in CAMO already, the inventory is not created automatically.

You can alter the default behavior for ad hoc tasks. Set the auto_create_inv attribute on the inst_part to false to specify that no inventory is created for ad hoc tasks requiring serialized parts that do not exist in CAMO already. In effect, this makes both ad hoc and baseline tasks behave in the same manner.

If the auto_create_inv attribute is omitted, or set to true, the default behavior is expected.

If the inventory exists in CAMO already but is installed somewhere else, the API does not create a new inventory. At task completion, CAMO sends an alert to Technical Records.

The following sample of an external_maintenance message displays the format of the part requirements section with the auto_create_inv attribute:

```
...
<part_requirement>
...
      <inst_part auto_create_inv="false">
         <assmbl_cd>738-1</assmbl_cd>
```

```
         <assmbl_bom_cd/>
         <eqp_pos_cd>1.1</eqp_pos_cd>
      <serial_no_oem>EMA2121</serial_no_oem>
      <part_no_oem>6079T77P06</part_no_oem>
      <manufacturer>
         <manufact_cd>ABC</manufact_cd>
         <manufact_name>ABC Company</manufact_name>
      </manufacturer>
      <quantity>1</quantity>
      <part_note/>
   </inst_part>
</part_requirement>
...
```

Updating installed inventory usage

When an inventory item is installed in maintenance task, the external system must update its usage values using the component_usages element (a child of the <inst_part> segment).

Processing removed parts

If the external_maintenance message specifies that an inventory with serial number NNN has been removed in a task, CAMO updates the inventory with the data in the message. If the removed inventory can not be found in the system, CAMO sends an alert to Technical Records at task completion.

Processing replacement of installed parts

You can request CAMO to process a part replacement in one of two ways:

- You can use an external_maintenance message that contains a part removal task and an part installation task for the same configuration slot OR
- You can update the serial number for the configuration slot manually (using a valid serial number) and then use an external_maintenance message with only a part installation task for the configuration slot.

Creating and updating labor requirements

CAMO generates the default labor requirements as follows:

- For the baseline tasks, CAMO uses the labor requirement defined by the baseline.
- For the non-historical ad hoc tasks, the default labor requirement is defined by the CAMO configuration data (ADHOC_TASK_LABOUR and ADHOC_CORRTASK_LABOUR).

The EMA updates the labour requirements with the information provided in the `external_maintenance` message.

The following sample of an external_maintenance message displays the format of the labor requirements section:

```
...
<labour_skills>
   <labour>
      <hr_cd>mlacroix</hr_cd>
      <labour_skill_cd>LBR</labour_skill_cd>
      <labour_time_cd>REG</labour_time_cd>
      <sched_hr>1</sched_hr>
      <actual_hr>1</actual_hr>
      <actual_start_dt>2005-07-03 12:45:00</actual_start_dt>
      <actual_end_dt>2005-07-03 12:45:00</actual_end_dt>
      <labour_cost>0</labour_cost>
   </labour>
   <labour>
      <hr_cd>amac</hr_cd>
      <labour_skill_cd>RII-INSP</labour_skill_cd>
      <labour_time_cd>REG</labour_time_cd>
      <sched_hr>1</sched_hr>
      <actual_hr>1</actual_hr>
      <actual_start_dt>2005-07-03 12:45:00</actual_start_dt>
      <actual_end_dt>2005-07-03 12:45:00</actual_end_dt>
      <labour_cost>0</labour_cost>
   </labour>
</labour_skills>
...
```

Creating and updating measurements

The EMA supports task measurements.

The following sample of an `external_maintenance` message displays the format of the measurements section:

```
...
<measurements>
   <measurement>
      <data_type_cd>TSIG</data_type_cd>
      <data_value>CLEAR</data_value>
   </measurement>
   <measurement>
      <data_type_cd>CAL</data_type_cd>
      <data_value>2008-11-16</data_value>
   </measurement>
</measurements>
...
```

Each data type must contain one data value that corresponds to the data type.

Each data type must correspond to a measurement parameter that exists in CAMO. In CAMO, each measurement parameter is associated to one of the domain types.

In the previous EMA message sample, the TSIG data type would correspond to a CAMO measurement parameter with a CH domain type, and the CAL data type would correspond to a CAMO measurement parameter with a CME domain type.

The EMA validates each measurement specified in a message against the measurement parameter in CAMO. The following criteria are enforced by the EMA for each domain type:

- CH (Character): The value must be specified in MIM_DATA_VALUE table. The value is not case sensitive.
- CHK (Check Boxes): The value must be either 1/0, TRUE/FALSE, true/false, Y/N, YES/NO, yes/ no. A blank value is defaulted to false.
- CME (Calendar Measurement): The value must be in date format YYYY-MM-DD.
- ME (Measurement): The value must be a valid number.
- TEXT (Free Form Text): The value must not be longer than 4000 characters.

When a measurement is supplied for a task in an EMA message, the EMA checks whether the measurement already exists for the task. If the measurement exists, the EMA updates the measurement details with the information provided in the message. If the measurement does not exist, the EMA adds the measurement to the task.

Limitations:

- EMA message schema does not support the measurement conversion factors. The inbound messages must contain the default unit of measure for the measurement values.
- EMA messages cannot reference an assembly measurement.

Completing tasks and faults

To complete a task, provide the actual end date for the task. The following sample of an external_maintenance message displays the format of the actual end date element for a task:

```
...
<task>
...
   <actual_start_dt/>
   <actual_end_dt>2007-07-05 15:00:00</actual_end_dt>
...
```

Creating sign off requirements

For non-historical WP and RO, CAMO generates the default signature rows based on the CAMO configuration data (BLANK_CHECK_SIGNATURE for WPs and BLANK_RO_SIGNATURE

for ROs). The EMA updates the sign-off requirements with the information provided in the external_maintenance message.

For historical WP and RO, the EMA imports the sign-off requirements provided in the external_maintenance message into CAMO.

The following sample of an external_maintenance message displays the format of the sign off requirement section:

```
...
<signoff_requirements>
   <signoff_requirement>
      <labour_skill_cd>PILOT</labour_skill_cd>
      <hr_cd>jsmith</hr_cd>
      <signoff_date>2005-08-28 17:00:00</signoff_date>
   </signoff_requirement>
</signoff_requirements>
...
```

Creating usage snapshots

If the external system provides usage data, CAMO creates usage snapshots for the main inventory, task inventory or fault inventory.

The following sample of an external_maintenance message displays the format of the usage snapshot section:

```
...
<usages>
   <usage_snapshot>
      <data_type_cd>HOURS</data_type_cd>
      <tsn_qt>50</tsn_qt>
      <tso_qt>0</tso_qt>
      <tsi_qt>0</tsi_qt>
      <assmbl_tsn_qt>2800</assmbl_tsn_qt>
      <assmbl_tso_qt>0</assmbl_tso_qt>
   </usage_snapshot>
</usages>
...
```

Creating usage snapshots for work packages and repair orders

If the external system does not provide any usage data for a WP or an RO, the EMA applies the usage snapshot of the Historical Flight/UR.

If the EMA cannot apply any usage snapshot to the WP or RO, CAMO uses the inventory's current usage values for the maintenance event.

Creating usage snapshots for tasks and faults

If the external system does not provide any usage data for a task or a fault, the EMA applies the Historical Flight/Usage Record.

If the Historical Flight/Usage Record cannot be found or is not applicable, the EMA applies the usage snapshot defined for the WP or RO in the external_maintenance message to the task or fault, if possible.

If the API cannot apply any of usage snapshot data to the task or fault, CAMO uses the inventory's current usage values for the maintenance event.

Handling erroneous messages

If an erroneous external_maintenance message causes run-time exceptions, where missing data or execution of business logic may compromise the integrity of the CAMO system, the EMA responds with an error message.

EMA error messages can be localized.

4

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`

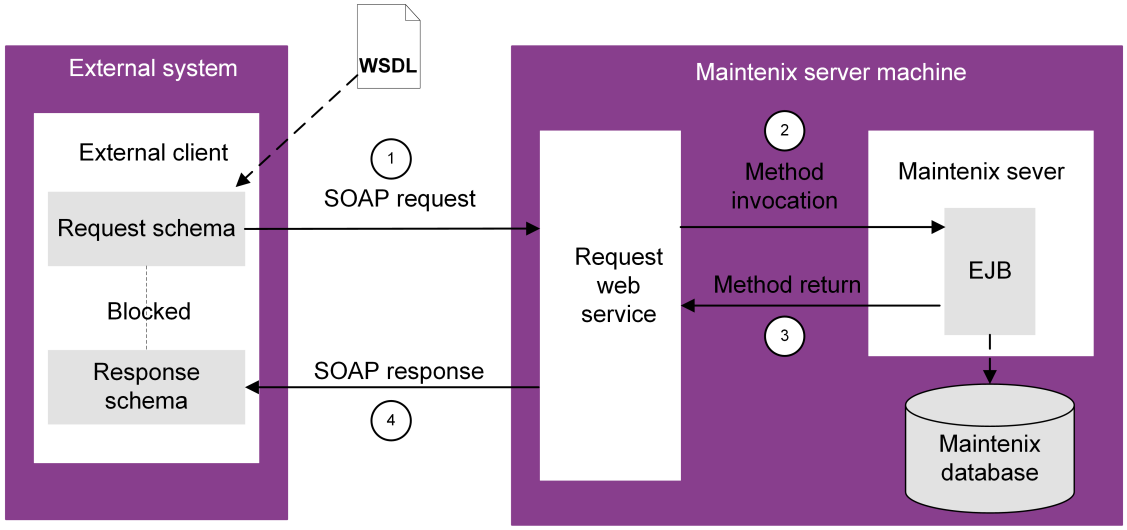
The same Web service is called, regardless of the data that is passed.

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

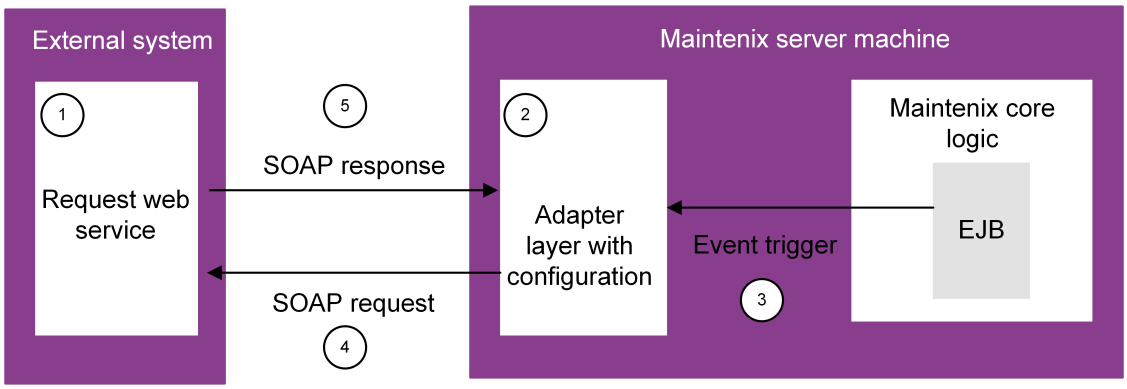
Notification service messages are used to alert an external system that something happened within Maintenix. For example an external system can be notified whenever a fault is raised, or a purchase order is updated.

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.