Oracle Field Service

Integrating with Outbound API

22A

Oracle Field Service
Integrating with Outbound API

22A

F51043-01

Authors: The Field Service Information Development Team

Contents

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the *Oracle Help Center*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we're working to remove insensitive terms from our products and documentation. We're also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Contacting Oracle

Access to Oracle Support

Customers can access electronic support through Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides. Please take one of the following surveys:

- For web-based user guide, *Web-based User Guide Survey*
- For tutorial feedback, *Tutorial Survey*

1 Introduction

Document Purpose

The document is to provide understanding of the basic Outbound API goals, its methods, and relevant SOAP transactions.

Scope of the Document

This document primarily describes the API that is used by the Outbound Interface of Oracle Field Service (OFS) for exchanging information (sending requests and accepting responses) with external systems. It also gives an overview of how the Oracle Field Service Outbound Interface works.

The recommended use for the Outbound Interface is for time-based notifications (for example, notifications to customers) typically using the Reminder and Change notification triggers. For all other system events (for example, Route changes, Activity status changes, Inventory changes, Service Request

changes, and so on), it is recommended to use the Core API/Events REST API for integration.

Target Audience

The document is intended for developers and programmers working with the OFS Outbound Interface in order to integrate OFS with external systems.

Accessing the APIs

To access the Oracle Field Service APIs, you must use the **https://<instance_name>.etadirect.com** URL scheme. All old URL schemes such as, companyname.etadirect.com, na.etadirect.com, eu.etadirect.com, and so on are deprecated for Oracle Field Service versions 15.8 and later.

For example, if you are using **https://companyname.etadirect.com/soap/inbound/?wsdl** to access the Inbound WSDL API, the URL per the new scheme is **https://<instance_name>.etadirect.com/soap/inbound/?wsdl**.

Glossary

Term	Explanation
Activity	Entity of the Oracle Field Service system that represents any time-consuming activity of the resource
Message	Communications within software (which may or may not be readable by humans), as well as person-to-person communications delivered via computer software
Middleware	Software that is used to integrate Oracle Field Service with external systems. Middleware uses Oracle Field Service API to interact with Oracle Field Service
Resource	Element in the resource tree representing a defined company asset
Resource Tree	Hierarchy of company resources, showing “parent-child” relationships
Route	List of activities assigned to a resource for a specific date, or a list of non-scheduled activities assigned to a resource
SOAP	Lightweight protocol for information exchange in a decentralized, distributed environment
SOAP 1.1	see *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*
SOAP Client	Application or part of application that sends SOAP requests to SOAP Service
SOAP Service	Application or part of application that receives SOAP requests sent by SOAP Client
User	1) Person using Oracle Field Service 2) Entity used for authentication and authorization, allowing people or external software to access Oracle Field Service
Visit	Group of activities related to the same customer that generate one customer notification for a case, instead of one notification for an activity

2 Outbound API Overview

Outbound Interface Overview

Outbound API is used for interaction between the OFSC message engine and external Middleware.

Middleware is the software that needs to be developed in order to integrate OFSC with external system(s).

Message Engine

The Message Engine is a part of the OFSC platform designed for creation of messages and their preliminary processing prior to delivery.

OFSC Message Engine is highly configurable and can initiate sending messages triggered by different events that take place in the OFSC system (such as activity creation, cancellation, completion, or reassignment; inventory installation, de-installation, exchange or hit, messages initiated by the user of the OFSC system) or the state of the system at a specific moment of time (activity is not started on time etc.).

Creation of Messages by Message Engine

Message Engine has a set of predefined triggers associated with various events in OFSC. Once a certain event happens an appropriate trigger is activated and the Message Engine initiates the corresponding Message Scenario.

Message Scenario consists of one or more Starting Steps and zero or more Inner Steps.

- Starting Steps are executed once when the scenario is executed.
- Inner steps (result handlers) are optionally executed to handle results of Starting Steps or other Inner Steps.

A message is generated whenever a step is executed.

Each message step has the 'Notification Method' property, which defines where messages are sent. Following is the list of some of the Notification Methods:

- E-mail – email notification message – handled internally in OFSC
- External System – message sent to Middleware

> **Note:** only the External System method is related to the Outbound SDK. Other methods are listed here as part of the Message Engine description.

Message Status

Any message generated by OFSC at any moment of time has a Message Status. Message statuses define the flow of message processing. Message statuses are divided into:

Final:

This status means that the message processing is finished and requires no further processing. OFSC will neither send any further requests nor expect any incoming requests regarding this message.

Non-final:

These statuses notify the system that the message processing is not finished.

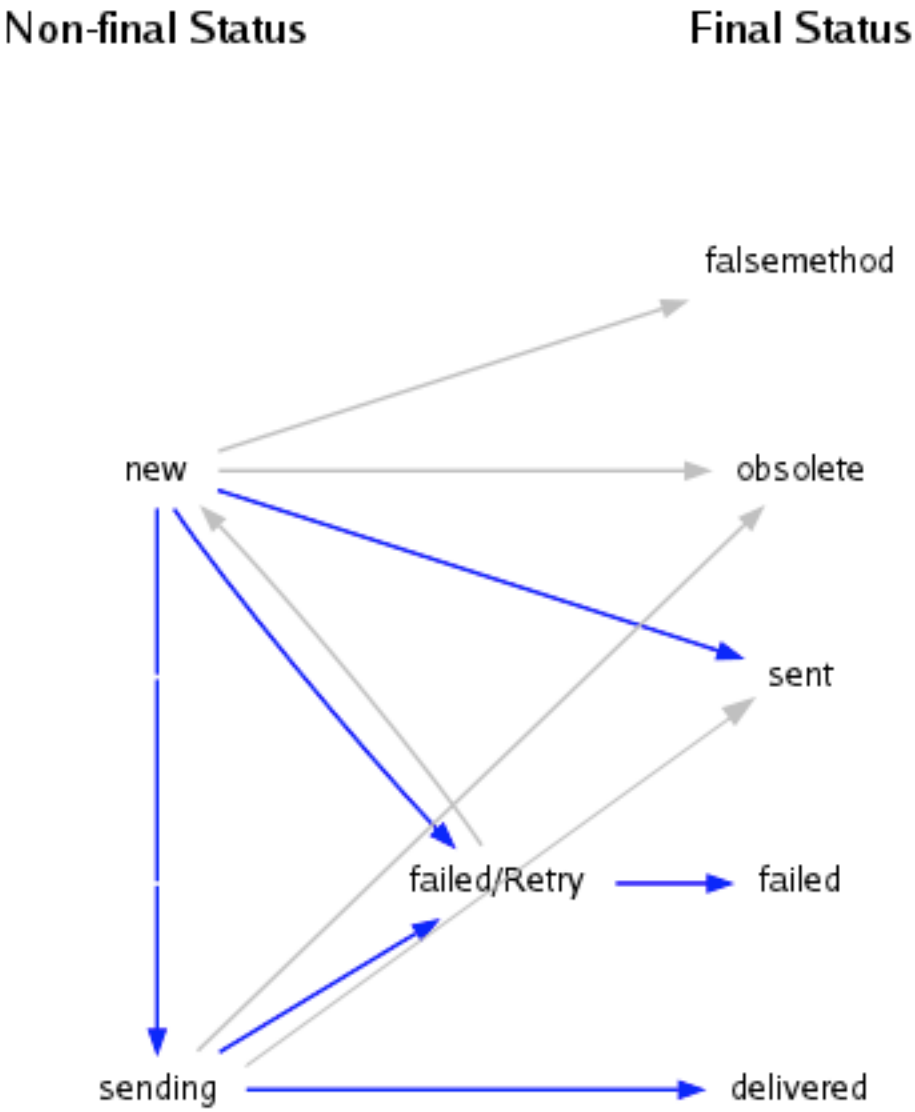
Message Statuses may be changed by Middleware and by OFSC internal processes. The processing ends when the message reaches one of the final message statuses.

List of message statuses:

- **falsemethod** (final) – set by OFSC if the message itself or an associated object (activity, for example) has no fields required for processing by the appropriate method. For example, there is no e-mail address defined for the E-mail method. This status is not applied to the Outbound API messages.
- **obsolete** (final) – set by OFSC if the message is no longer relevant. For example, the day before a message was generated to inform the customer about an activity, but the activity had been cancelled before such message was delivered.
- **delivered** (final) – set by Middleware to signify that the message processing is finished. While 'sent' and 'delivered' statuses are very similar, in most cases 'sent' is used when there is no evidence that the final recipient (person or system) received the message while 'delivered' is used when the receipt is confirmed.
- **failed** (final unless 'attempts' > 1) – set by Middleware to signify that the message delivery has failed, may be set by OFSC when an error occurs and the message did not reach its recipient.
- **sent** (final) – set by Middleware to signify that the message was received by Middleware and there will be no further status updates.
- **new** (non-final) – initial status after a message is created, processing has not started.
- **sending** (non-final) – set by Middleware to signify that the message was received by Middleware but its processing is not finished and there will be further status updates.

Note: messages with the 'failed' status can be returned to the 'new' status and resent for processing (configurable in the 'Step' properties). The 'failed' status is only considered final when there are no retries (default – no retries).

Following diagram reflects state transitions and also includes the final statuses which are set by OFSC itself – gray arrows.



Not all statuses may be used in a specific OFSC implementation.

3 Workflows

Workflows

Outbound API supports two kinds of workflow: Simple and Advanced.

Simple Workflow is used when Middleware operates in a synchronous mode with OFSC, while Advanced workflow is used when Middleware operates in an asynchronous mode with OFSC.

Choosing between Simple and Advanced Workflow

This section will help you understand which workflow will fit your project best.

Below are some descriptive features of the Outbound API which we recommend you consider when making your decision.

Throughput: Outbound performance depends on the amount of messages that can be transmitted per second. If this number is too low, then Outbound will not work properly which may result in the systems (OFSC and External) getting out of sync.

When Middleware receives a request with several messages, it will not receive other messages until it returns a SOAP response for the first batch.

So if Middleware takes 1 second to process each message, then OFSC will not send more than 3600 messages per hour.

For example after a routing run, the routing process generates a message for each activity it moved. If there were 10K activities in a bucket, it would send 10K messages, which would be processed in 3 hours at the rate of 1 message per second.

Batch sending: To achieve faster processing speed, OFSC sends messages in batches of 10 or more messages.

This means that if 50 messages are received by Middleware, then quickly put into queue and the response is returned in 1 second, then a single message gets sent effectively in 0.02 seconds.

Middleware internal queue: To achieve faster processing speed, Middleware can implement an internal queue.

When messages arrive at Middleware from OFSC, Middleware can quickly put messages into internal queue, then return the SOAP response to OFSC immediately. Middleware can then process messages in the queue asynchronously.

This way, 10K messages will be sent from OFSC in less than 5 minutes (assuming a batch of 50 messages is put to the queue in 1 second).

Timeouts. If the SOAP response from Middleware is not returned within 30 seconds, then OFSC considers it a connection failure, aborts the transaction, and resends the messages later.

For example, if a request contains a batch of 50 messages and each of them takes 1 second to be processed, then the whole request will never be processed – OFSC will drop the connection and resend the same messages repeatedly, until they expire in OFSC.

So Middleware must guarantee that it returns every SOAP response within 30 seconds.

Message cancellation

For messages that take very long time to be sent away, it makes sense to use the Advanced Workflow, even if the result is not relevant to OFSC. The reason behind it is that OFSC may know that a message is no longer relevant (e.g. the Activity was rescheduled and the message should be postponed/changed).

When using Advanced workflow, OFSC sends a 'drop_message' request in this case, notifying Middleware that the message no longer needs to be delivered.

This feature can reduce your expenses by canceling obsolete but costly messages and is only available in Advanced workflow.

Scenario	Advanced workflow	Simple workflow	Comments
• Message processing takes long time • OFSC is notified of the result at the end of processing	#	#	OFSC can receive notifications of message processing result in Advanced workflow only.
• Message processing takes long time • OFSC is not notified of the result at the end of processing	#	#	When using Simple workflow, Middleware must implement internal queue, put messages there, and return response ASAP.
• Message processing is very fast • response to OFSC is only returned after all messages are processed	#	#	Simple workflow can be used only if processing a batch of messages takes the same amount of time as putting them to the internal queue.
• Message processing takes long time • processing may be cancelled by OFSC due to new data	#	#	In Advanced workflow, OFSC can send a 'drop_message' command to notify Middleware that the message is obsolete.

Simple Workflow (No Delivery Confirmation)

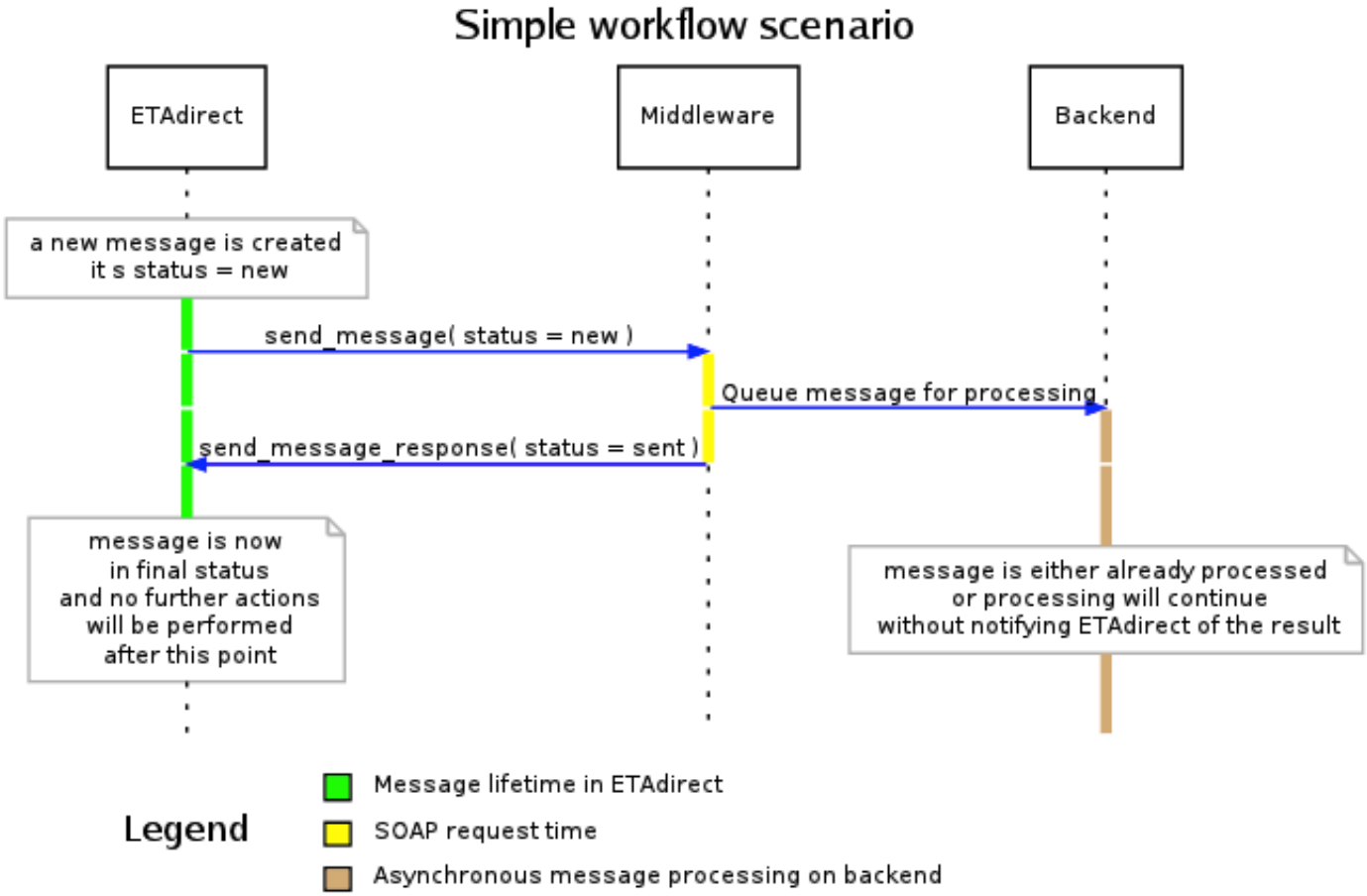
In this workflow Middleware performs the task of delivering messages to the Backend, but does not notify OFSC of the result.

Simple workflow:

- OFSC sends a 'send_message' SOAP request to Middleware
- The Middleware response contains a final message status: 'sent' or 'failed'
- OFSC will not send any further requests and will not expect any requests regarding this message
- Middleware guarantees the message delivery to the final recipient

When the Simple workflow is implemented, special focus should be on the performance of the 'send_message' processing. OFSC will not send the next outbound message until it gets the response for the previous message. If

Middleware interacts with systems where it has no control over the performance and time delays may appear, this may create a queue of messages on the OFSC side.



Middleware Requirements for Simple Workflow

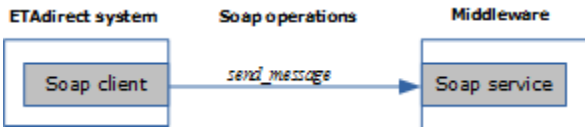
Middleware must implement a SOAP Service using Middleware_Simple. WSDL file provided with this SDK.

The Middleware SOAP Service will implement one method:

- send_message – this method is called by OFSC to send messages to Middleware.

Middleware must respond to 'send_message' requests from OFSC with one of the following statuses:

- sent – message queued for processing
- failed – message failed to be queued for processing



Advanced Workflow (With Delivery Confirmation)

In this workflow Middleware will attempt to deliver messages to the Backend and will notify OFSC of the message processing result afterwards.

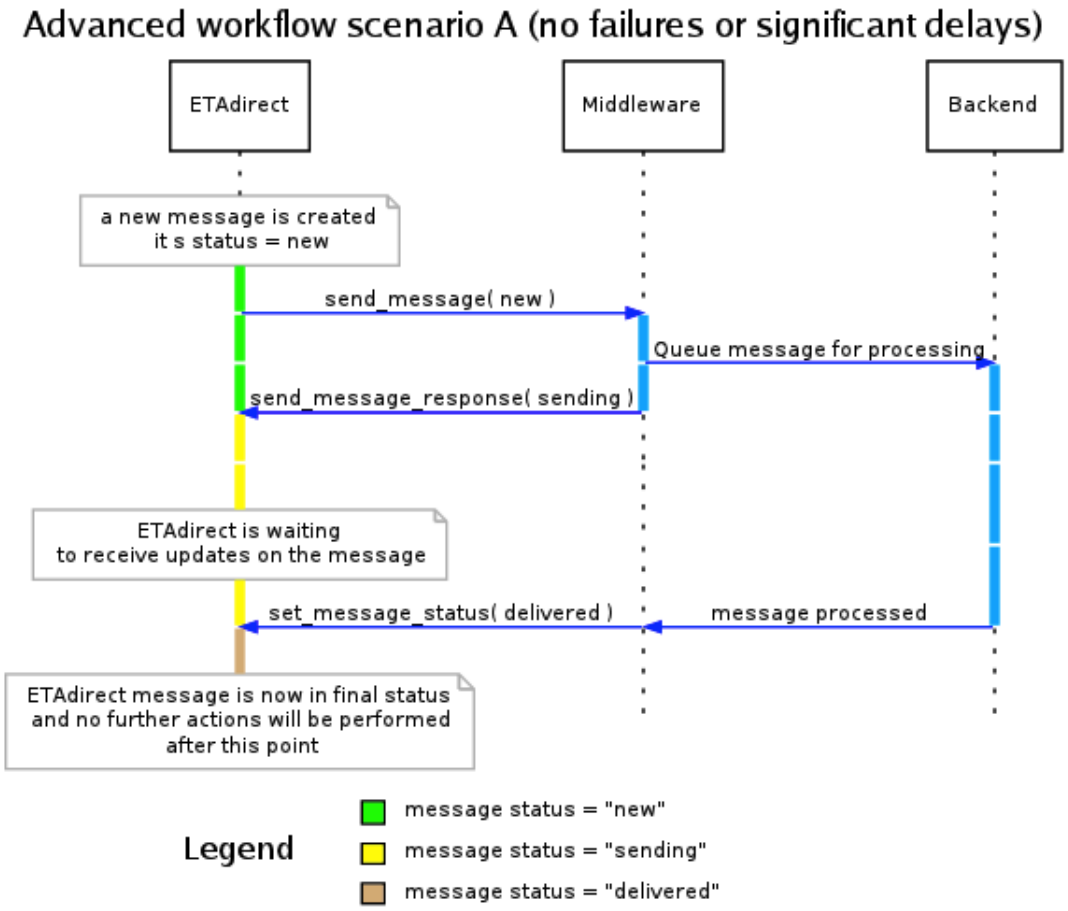
The Advanced workflow keeps OFSC in control of the message processing even after the message was received by Middleware (and up to the moment it is actually processed). This workflow is optimal for the integrations where message delivery takes significant time, for example, integration with IVRs. OFSC can generate hundreds of messages but their processing is limited by the number of voice channels available and the call duration.

Advanced workflow:

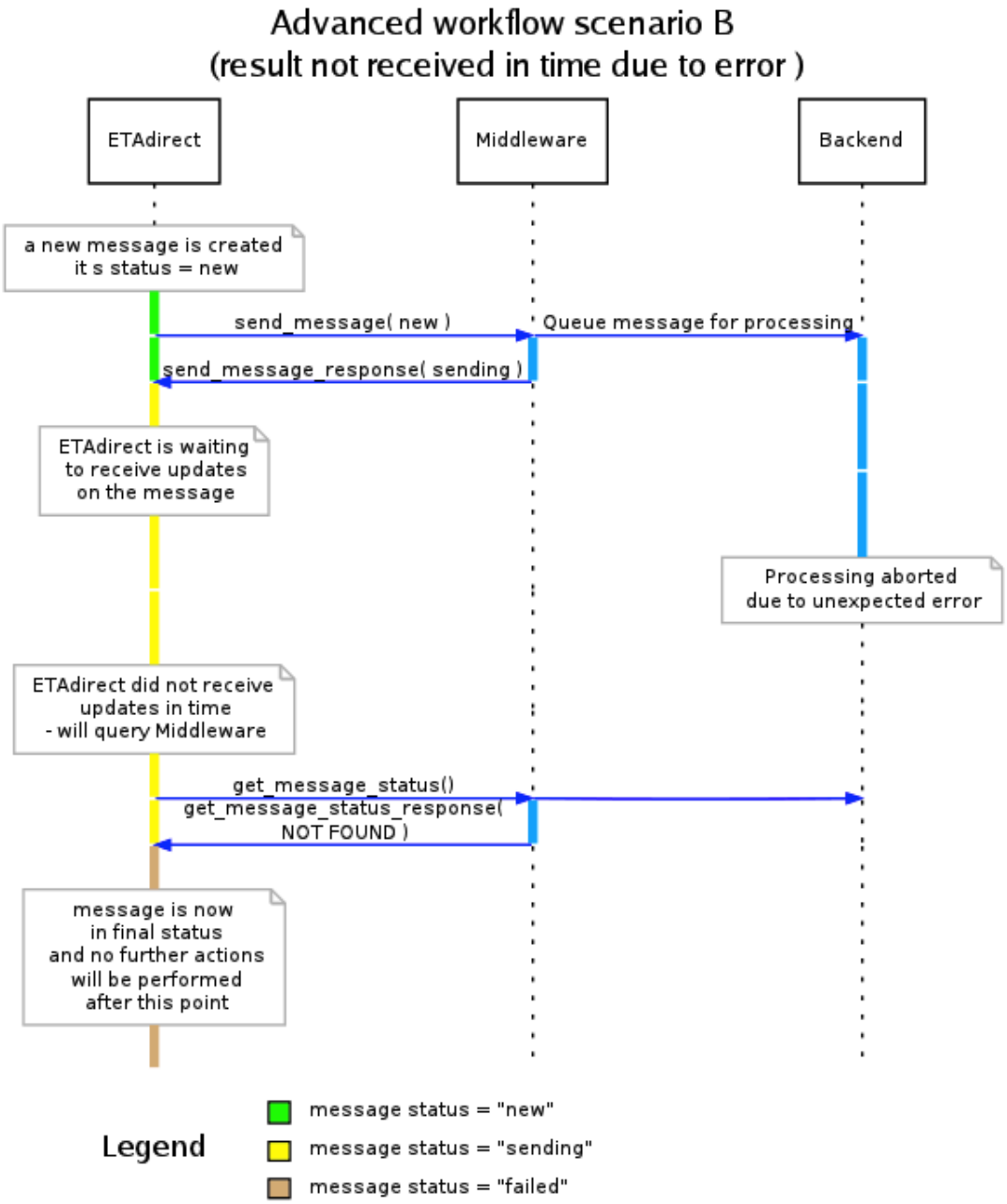
- OFSC sends a 'send_message' request to Middleware
- The Middleware response contains non-final message status 'sending' or final status 'failed'
- If 'failed' was returned, then the processing is finished, otherwise:
- OFSC will wait for some time expecting to receive the 'set_message_status' request from Middleware
- Middleware should send a 'set_message_status' request notifying OFSC of the result of processing. It should set the final status: 'delivered', 'failed', or 'sent'.

Sequence Diagrams of Advanced Workflow

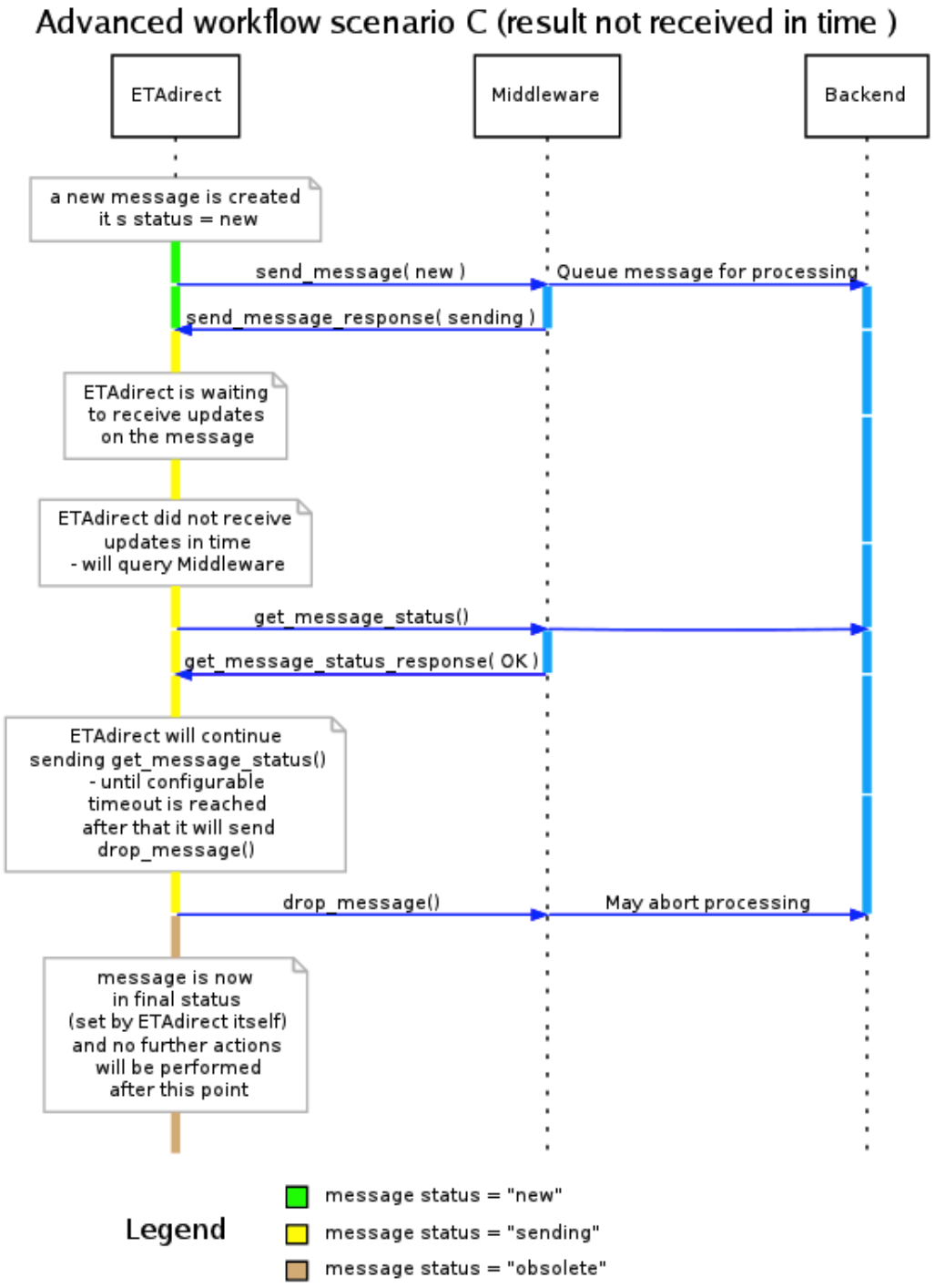
Optimistic Scenario: No Failures or Significant Delays



Error Scenario: Message Lost



Error Scenario: Result not Received in Time



Middleware Requirements for Advanced Workflow

Middleware SOAP Service for Advanced Workflow

Middleware must implement SOAP Service using **Middleware_Advanced.WSDL** file provided with this SDK.

The Middleware SOAP Service will implement three operations:

- send_message – this method is called by OFSC to send messages to Middleware.
- get_message_status – this method is called by OFSC to check if the message is still being processed.
- drop_message – this method is called by OFSC to indicate that message is obsolete and its processing can be stopped.

The Middleware must respond to 'send_message' requests from OFSC with one of the following 2 statuses:

- **sending** – message queued for processing
- **failed** – message failed to be queued for processing

The Middleware must respond to 'get_message_status' requests from OFSC with one of the following codes:

- **OK** if the message is still being processed. OFSC will continue sending 'get_message_status' requests periodically
- **NOT FOUND** if the message is not found. OFSC will mark this message as 'failed'
- **ERROR** if an unexpected error has occurred. OFSC will mark this message as 'failed'

Middleware SOAP Service for Advanced Workflow

Middleware must implement a SOAP Client that connects to OFSC at address:

```
https://{INSTANCE}.etadirect.com/soap/outbound/?wsdl
```

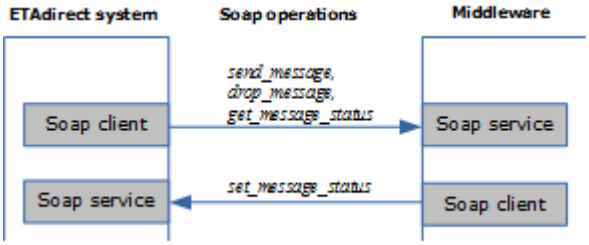
In this URL {INSTANCE} is a subdomain that may change. For example the integration may be done on one instance while the production will run on another instance. The WSDL contents will be the same at both instances, but the endpoint is different.

Middleware SOAP Client must send the following request to OFSC:

- set_message_status – notify OFSC of the message processing result.

The 'set_message_status' should set message status to one of the following final statuses:

- **delivered** – message processed successfully.



4 Implementation Guidelines

Implementation Guidelines

These guidelines are to help developers with their integration using the Outbound API.

Middleware Availability

Customers are responsible for ensuring their middleware is always available to receive messages from Oracle Field Service.

If the middleware is unavailable, it is possible that the messages could time-out or reach the maximum number of retries resulting in messages not being sent.

In such situations there is no mechanism available in Oracle Field Service to resend those messages. Customers that cannot ensure their middleware availability should consider using the Core API - Events (REST service).

Middleware Must Return All Responses Immediately

Responses to 'send_message', 'get_message_status', and 'drop_message' must be returned immediately.

When Middleware receives a message via 'send_message' operation and it needs to do some time-consuming processing, Middleware must return the response with 'sent', 'sending', or 'failed' status and then continue with processing in another thread or process.

Middleware implementation must not engage in any long processing before SOAP response has been returned to OFSC.

This is because Middleware will not receive any more messages on this message scenario step until it returns the SOAP response. Blocking during the SOAP call means that the messages will be processed very slowly and likely slower than they are generated.

Middleware Must Support Bulk SOAP Operations

In order to reduce the number of SOAP requests between the OFSC platform and Middleware, all methods in the Outbound API support bulk data. That is, each SOAP request can contain the data related to several messages. Also, a response record provides a separate execution result on a per message basis.

It is important when implementing SOAP Service to interpret the <messages> element as array. This may not be noticeable in the initial test with a single message, but the SOAP Service that does not have this feature will fail eventually.

The same applies for Middleware SOAP Client for advanced workflow. It should send 'set_message_status' requests periodically with all messages for a given period (e.g. every few seconds). It should not send each message status individually as it arrives to Middleware.

Example of 'send_message' bulk request (details omitted for clarity)

```
<env:Envelope>
 <env:Body>
```

```
<send_message xmlns="urn:toatech:agent">
<user>...</user>
<messages>
<message> <!-- message payload -->
</message> <message> <!-- message payload -->
</message> <message> <!-- message payload --> </message>
... <!-- more messages -->
</messages>
</send_message>
</env:Body>
</env:Envelope>
```

5 Outbound Interface Entities and Structures

User Authentication Structure

All API methods use the 'user' structure as authentication to determine the permissions of the Oracle Field Service client company user.

All customers can use the Client_ID and Client_Secret instead of login and password to populate the User Authentication Structure with credentials:

1. Register an application.
    a. In the Field Service Manage interface, click **Configuration** and select **Applications**.
    b. In the left pane, click the **plus** icon to open the New application window, specify the Application Name and Application ID, and click **Submit**.
    c. Under Authentication settings, select the **Authenticate using Client ID/Client Secret** check box.
    d. Click **Save**.
2. Select the application and under Authentication settings section, click **Show Client ID / Client secret** to view the Client ID and Client Secret.
3. Make a note of the Client ID and Client Secret.

The following table describes the Oracle Field Service SOAP authentication structure mandatory fields.

Name	Type	Description
now	string	current time in ISO 8601 format
company	string	case-insensitive identifier of the Client for which data is to be retrieved. provided by Oracle during integration.
login	string	The client ID of the application.
auth_string	string	authentication hash; The value of this field must be computed as follows: auth_string = SHA256(now + SHA256(CLIENT_SECRET+SHA256(CLIENT_ID)));

For example:

<user>

<now>CURRENT_TIME</now>

<login>CLIENT_ID</login>

<company>INSTANCE_NAME</company>

<auth_string>SHA256(CURRENT_TIME + SHA256(CLIENT_SECRET + SHA256(CLIENT_ID)))</auth_string>

</user>

Mandatory and Optional Properties

Each request sent by the Outbound API includes properties which are necessary for the request to be processed correctly and those which are only sent when certain value(s) are needed. In this respect, properties fall under either of the following two types:

Optional: the property is not necessary for the request to be processed correctly; if such property is not sent, the request will not return an error; the 'Required' column contains 'No' for such property.

Mandatory: the property must be sent in the request; if a mandatory property is invalid or missing, the request is rejected with a corresponding error; the 'Required' column contains 'Yes' for such property.

Authentication

The 'user' structure is used for the request authentication. The relevant error is returned if the authentication fails.

Number	Login	Description
1	now	is different from the current time on the server and this difference exceeds the predefined time-window (30 minutes by default)
2	company	cannot be found in the Oracle Field Service
3	login	cannot be found for this company
4	application is not authorized to use this API	
5	auth_string	when auth_string is not equal to: SHA256(now + SHA256(Client_Secret +SHA256(Client_ID)));;

Otherwise authentication is successful and the request is processed further.

Note: The specifics of the 'user' structure processing differ for different methods used in the Outbound API. Please refer to the description of each method for details.

6 Outbound API Methods Description

Outbound API Methods Description

The Outbound API uses the following methods:

Methods	Description
send_message	the method is called by OFSC to send messages to Middleware
drop_message	the method is called by OFSC to indicate that message is obsolete and its processing can be stopped
get_message_status	the method is called by OFSC to check if the message is still being processed
set_message_status	the method is used to notify OFSC of the message processing result

'send_message' Method

When an internal event or state in the OFSC system triggers a new message transaction (for example, an activity is cancelled or not started in time), the OFSC system establishes an HTTP connection with the Middleware and uses the 'send_message' SOAP method.

Note:

- the 'send_message' transaction execution time is critical, so it is important that 'send_message' does not contain very complex logics so that your system does not create significant delays between the transactions
- as the actual data transfer can be rather time-consuming, the Middleware should have an internal queue implemented.

'send_message' Request

The 'send_message' request specifies the parent application for the message and the message to be sent. The 'send_message' request contains the following parameters:

Name	Required	Description
user	No	'user' node **Note**: the 'user' structure may be ignored in the 'send_message' request
messages	Yes	array of 'message' elements each containing data for a single message

'message' Element of 'send_message' Request

The 'messages' array is a set of 'message' elements. Each 'message' element contains message fields. The list of fields is configured in the course of implementation:

Field	Required	Description
app_host	Yes	three fields that define SOAP API location of the calling application. The address may be used to submit message results ('set_messages_status') **Note:** These fields may be ignored by Middleware as the SOAP API location of OFSC endpoint is known beforehand.
app_port		
app_url		
message_id	Yes	unique ID of the message in the OFSC system used in all other methods to refer to this message integer number that cannot be empty 32-bit integer
address	No	message destination for email – corresponds to the recipient's e-mail address specified for the corresponding message step for external system – the field is empty
send_to	No	time limit for sending the message date and time field represented in GMT (YYYY-MM-DD HH24:MI:SS); value sent in the field is defined in the message step configuration of OFSC and defines the latest time by which message has to be sent agents that deal with a back office system usually should ignore this field, unless it is a part of the solution defined with OFSC
subject	No	filled and preprocessed templates for 'subject' and 'body' text blocks defined in message step configuration that contain all fields required for the information transfer (activity fields, inventory fields, etc.)
body	No	format and content of the text blocks are defined in the course of implementation, but can be changed via OFSC Manage.

'send_message' Request Example

If you select the **Allow basic access authentication** check box from the **Message Scenario, Delivery Channels** screen, then the user credentials are sent using the standard HTTP header "Authorization" in the request. Also, the <user> SOAP structure is sent in the body of the request. The client application can either use the standard HTTP header "Authorization" or the <user> SOAP structure to send user credentials in the request.

```
Authorization: Basic YnJjLnJvb3Q6MQ==
Host: 10.175.207.217
Content-Length: 832
```

```
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
 <soapenv:Body>
 <urn:send_message>
 <urn:user>
 <urn:now>2011-11-23T15:50:23+00:00</urn:now>
 <urn:login>user_name</urn:login>
 <urn:company>company_name</urn:company>
 <urn:auth_string>67c5900a04abc54132a52da8a2320be2</urn:auth_string>
 </urn:user>
 <urn:messages>
 <urn:message>
 <urn:app_host>service.example.com</urn:app_host>
 <urn:app_port>443</urn:app_port>
 <urn:app_url>/soap/</urn:app_url>
 <urn:message_id>2006</urn:message_id>
 <urn:address>someone@examplemail.com</urn:address>
 <urn:send_to>2011-11-24 01:59:00</urn:send_to>
 <urn:subject></urn:subject>
 <urn:body>{
 "appt_number" : "XXX1234",
 "name":"Rakesh Ivanov",
 "phone": "1234567"
 }</urn:body>
 </urn:message>
 </urn:messages>
 </urn:send_message>
 </soapenv:Body>
</soapenv:Envelope>
```

If you do not select the **Allow basic access authentication** check box, the standard HTTP header is not used in the request and the client application can use the <user> SOAP structure for authentication.

```
Host: 10.175.207.217
Content-Length: 832
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
 <soapenv:Body>
 <urn:send_message>
 <urn:user>
 <urn:now>2011-11-23T15:50:23+00:00</urn:now>
 <urn:login>user_name</urn:login>
 <urn:company>company_name</urn:company>
 <urn:auth_string>67c5900a04abc54132a52da8a2320be2</urn:auth_string>
 </urn:user>
 <urn:messages>
```

```
<urn:message>
<urn:app_host>service.example.com</urn:app_host>
<urn:app_port>443</urn:app_port>
<urn:app_url>/soap/</urn:app_url>
<urn:message_id>2006</urn:message_id>
<urn:address>someone@examplemail.com</urn:address>
<urn:send_to>2011-11-24 01:59:00</urn:send_to>
<urn:subject></urn:subject>
<urn:body>{
"appt_number" : "XXX1234",
"name":"Rakesh Ivanov",
"phone": "1234567"
}</urn:body>
</urn:message>
</urn:messages>
</urn:send_message>
</soapenv:Body>
</soapenv:Envelope>
```

'send_message' Response

When the middleware accepts the 'send_message' request it has to return the 'message_response'.

Note: Responses to 'send_message' must be returned as soon as possible.

This structure contains the following parameter fields:

Name	Required	Description
message_id	Yes	'message_id' value from the request
status	Yes	new value of the 'status' field. Possible values are: for messages that do not pass validation the status is 'failed' for messages that are correct but do not require actual transfer to the back office system the status is 'sent' for messages that require data transfer to the back office system the status is 'sending' (and the messages are placed on the internal queue of the middleware) for messages successfully delivered to the back office system the status is 'delivered' **Note:** this is a standard set of statuses to be returned and their sending conditions, but for each project it should be agreed with implementation
description	No	new value of the 'description' field; customer-specific additional value that along with Message Status can influence the flow of a message scenario. For example Message Status 'Failed' can differ subject to its descriptions. **Note:** set of possible descriptions should be agreed with implementation

Name	Required	Description
data	No	new value of the 'data' field – used only if so required by OFSC solution (e.g. can be used to assign values to activity and inventory properties, cancel activities and set them as 'non-scheduled' as described in more details below)
external_id	No	new value of 'external_id' field – identifier of the message in the internal queue (usually not used)
duration	No	new value of 'duration' field for the message record in OFSC (usually not used)
sent	No	actual time when the message was sent: GMT YYYY-MM-DDTHH24:MI:SS+00:00 format date and time
fault_attempt	No	number of the remaining attempts to resend the message in case the notification has failed this way the external system can change the number of the remaining attempts (e.g. stop or continue resending until success) unless there is a particular need to use the functionality, the field should be omitted, so that the fault attempt logic remains in accordance with the corresponding message scenario step **Note:** resending in this case does not create a new message, the same message (with the same id) is being resent.
stop_further_attempts	No	this field should be set to '1' which means that notification attempts are stopped. No other values should be used
time_delivered_start	No	time delivered interval (promised to the customer) in HH:MM:SS format if 'status' returned is 'delivered' or 'sent', the fields are updated for activity/visit
time_delivered_end	No	

'send_message' Response Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:toatech:agent">
 <soapenv:Header/>
 <soapenv:Body>
 <urn:send_message_response>
 <urn:message_response>
 <urn:message_id>2006</urn:message_id>
 <urn:status>sent</urn:status>
 <urn:description>everything is fine</urn:description>
 </urn:message_response>
 </urn:send_message_response>
 </soapenv:Body>
</soapenv:Envelope>
```

'drop_message' Method

The 'drop_message' method is used to remove messages from the agent internal queue, if message sending should be canceled (e.g if the activity has been canceled or deleted).

> **Note:** Sometimes a situation may occur when there is no real need for the method. Not to change the workflow, a simple method can be implemented that always returns an error.

'drop_message' Request

The 'drop_message' request specifies the message to be dropped and contains the following fields:

Name	Required	Description
user	No	'user' node **Note**: the 'user' structure may be ignored in the 'drop_message' request
messages	Yes	array of 'message' elements each containing data for a single message

'message' Element of 'drop_message' Request

The 'messages' array is a set of 'message' element. Each 'message' element contains just one field:

Field	Required	Description
message_id	Yes	ID of the message to be removed

'drop_message' Request Example

If you select the **Allow basic access authentication** check box from the **Message Scenario, Delivery Channels** screen, then the user credentials are sent using the standard HTTP header "Authorization" in the request. Also, the <user> SOAP structure is sent in the body of the request. The client application can either use the standard HTTP header "Authorization" or the <user> SOAP structure to send user credentials in the request.

```
Authorization: Basic YnJjLnJvb3Q6MQ==
Host: 10.175.207.217
Content-Length: 832
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
```

```
<SOAP-ENV:Body>
 <ns1:send_message xmlns="urn:toatech:agent">
 <soapenv:Body>
 <urn:drop_message>
 <urn:user>
 <urn:now>2011-11-23T15:50:23+00:00</urn:now>
 <urn:login>user_name</urn:login>
 <urn:company>company_name</urn:company>
 <urn:auth_string>67c5900a04abc54132a52da8a2320be2</urn:auth_string>
 </urn:user>
 <urn:messages>
 <urn:message>
 <urn:message_id>2006</urn:message_id>
 </urn:message>
 <urn:message>
 <urn:message_id>2007</urn:message_id>
 </urn:message>
 </urn:messages>
 </urn:drop_message>
 </soapenv:Body>
</soapenv:Envelope>
```

If you do not select the **Allow basic access authentication** check box, the standard HTTP header is not used in the request and the client application can use the <user> SOAP structure for authentication.

```
Host: 10.175.207.217
Content-Length: 832
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
<SOAP-ENV:Body>
 <ns1:send_message xmlns="urn:toatech:agent">
 <soapenv:Body>
 <urn:drop_message>
 <urn:user>
 <urn:now>2011-11-23T15:50:23+00:00</urn:now>
 <urn:login>user_name</urn:login>
 <urn:company>company_name</urn:company>
 <urn:auth_string>67c5900a04abc54132a52da8a2320be2</urn:auth_string>
 </urn:user>
 <urn:messages>
 <urn:message>
 <urn:message_id>2006</urn:message_id>
 </urn:message>
 <urn:message>
 <urn:message_id>2007</urn:message_id>
 </urn:message>
 </urn:messages>
 </urn:drop_message>
 </soapenv:Body>
</soapenv:Envelope>
```

'drop_message' Response

The 'drop_message' response is an array of one or more 'message_response' nodes.

Note: Responses to 'drop_message' must be returned as soon as possible.

Each 'message_response' node contains the following fields:

Name	Required	Description
message_id	Yes	ID of the message
result	Yes	node that contains transaction result description

'result' Node of 'drop_message' Response

Each result node contains the following elements

Field	Required	Description
code	Yes	message removal (from the internal middleware queue) result code. Possible values are: NOT FOUND – message ID is unknown to the agent OK – message successfully removed ERROR – either the message is under processing at the moment or an internal agent error occurred
desc	No	error description

'drop_message' Response Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:toatech:agent">
 <soapenv:Body>
 urn:drop_message_response>
 <urn:message_response>
 <urn:message_id>2006</urn:message_id>
 <urn:result>
 <urn:code>OK</urn:code>
 </urn:result>
 </urn:message_response>
 <urn:message_response>
 <urn:message_id>2007</urn:message_id>
 <urn:result>
 <urn:code>ERROR</urn:code>
 <urn:desc>Cannot drop the message. The message is under processing at the moment.</urn:desc>
 </urn:result>
 </urn:message_response>
 </urn:drop_message_response>
 </soapenv:Body>
</soapenv:Envelope>
```

'get_message_Status' Method

The 'get_message_status' method is used to retrieve the message status from the agent internal queue (when the message handling status has not been returned to OFSC in time).

'get_message_status' Request

The 'get_message_status' request specifies the message for which the status is to be retrieved and contains the following fields:

Name	Required	Description
user	No	'user' node **Note**: the 'user' structure may be ignored in the 'get_message_status' request
messages	Yes	array of 'message' elements each containing data for a single message

'message' Element of 'get_message_status' Request

The 'messages' array is a set of 'message' elements. Each 'message' element contains just one field:

Field	Required	Description
message_id	Yes	ID of the message the status of which is to be returned

'get_message_status' Request Example

If you select the **Allow basic access authentication** check box from the **Message Scenario, Delivery Channels** screen, then the user credentials are sent using the standard HTTP header "Authorization" in the request. Also, the <user> SOAP structure is sent in the body of the request. The client application can either use the standard HTTP header "Authorization" or the <user> SOAP structure to send user credentials in the request.

```
Authorization: Basic YnJjLnJvb3Q6MQ==
Host: 10.175.207.217
Content-Length: 832
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
<SOAP-ENV:Body>
 <ns1:send_message xmlns="urn:toatech:agent">
 <user>
```

```
 <now>2017-05-24T11:32:04+00:00</now>
 <company>kh01_i1</company>
 <login>brc.root</login>
 <auth_string>9ee269f38b7d1ae685c4fdebbfd90693</auth_string>
 </user>
 <messages>
 <message>
 <app_host>example.oracle.com</app_host>
 <app_port>10113</app_port>
 <app_url>/outbound/</app_url>
 <message_id>9948341</message_id>
 <address></address>
 <send_to>2017-05-24 11:36:04</send_to>
 <subject></subject>
 <body></body>
 </message>
 </messages>
 </ns1:send_message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If you do not select the **Allow basic access authentication** check box, the standard HTTP header is not used in the request and the client application can use the <user> SOAP structure for authentication.

```
Host: 10.175.207.217
Content-Length: 832
Accept-Charset: utf-8
SOAPAction: "agent_service/send_message"
Keep-Alive: 0
User-Agent: TOA Server
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:toatech:agent">
<SOAP-ENV:Body>
 <ns1:send_message xmlns="urn:toatech:agent">
 <user>
 <now>2017-05-24T11:32:04+00:00</now>
 <company>kh01_i1</company>
 <login>brc.root</login>
 <auth_string>9ee269f38b7d1ae685c4fdebbfd90693</auth_string>
 </user>
 <messages>
 <message>
 <app_host>example.oracle.com</app_host>
 <app_port>10113</app_port>
 <app_url>/outbound/</app_url>
 <message_id>9948341</message_id>
 <address></address>
 <send_to>2017-05-24 11:36:04</send_to>
 <subject></subject>
 <body></body>
 </message>
 </messages>
 </ns1:send_message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

'get_message_status' Response

The 'get_message_status' message response is an array of one or more 'message_response' nodes.

> **Note:** Responses to 'get_message_status' must be returned as soon as possible.

Each 'message_response' node contains the following fields:

Name	Required	Description
message_id	Yes	ID of the message
result	Yes	element that contains transaction result description

'result' Node of 'get_message_status' Response

The 'messages' array is a set of 'message' elements. Each 'message' element contains message fields. The list of fields is configured in the course of implementation:

Field	Required	Description
code	Yes	message removal (from the internal middleware queue) result code. Possible values are: NOT FOUND – message ID is unknown to the agent OK (desc = WAITING) – message sending has not yet been started OK (desc = SENDING) – message is being processed at the moment ERROR – an internal agent error occurred
desc	No	error description

'get_message_status' Response Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:toatech:agent">
 <soapenv:Header/>
 <soapenv:Body>
 <urn:get_message_status_response>
 <urn:message_response>
 <urn:message_id>2006</urn:message_id>
 <urn:result>
 <urn:code>OK</urn:code>
 <urn:desc>WAITING</urn:desc>
 </urn:result>
 </urn:message_response>
 <urn:message_response>
 <urn:message_id>2007</urn:message_id>
 <urn:result>
 <urn:code>OK</urn:code>
 <urn:desc>SENDING</urn:desc>
 </urn:result>
```

```
</urn:message_response>
<urn:message_response>
<urn:message_id>2006</urn:message_id>
<urn:result>
<urn:code>OK</urn:code>
<urn:desc>NOT FOUND</urn:desc>
</urn:result>
</urn:message_response>
</urn:get_message_status_response>
</soapenv:Body>
</soapenv:Envelope>
```

'set_message_status' Method

The 'set_message_status' method is the only method used by OFSC SOAP API. The method returns transaction results.

If as the result of 'send_message' method, Middleware has returned status = 'sending', OFSC SOAP API method 'set_message_status' is used to return the result after the actual end of the transaction. Middleware can also use this method to update fields of the message in the OFSC system. One call of this method can be used to set the status for several messages.

'set_message_status' Request

The 'set_message_status' request defines the message for which the status is to be set and contains the following fields:

Name	Required	Description
user	No	'user' structure
messages	Yes	array of 'message' elements each containing data for a single message

'set_message_status' Request 'messages' Array

The 'messages' array is a set of one or more 'message' nodes. Each 'message' node contains:

Field	Required	Description
message_id	Yes	'message_id' value from the 'send_message' request
status	Yes	new value of the 'status' to be set for the message; possible **case-sensitive** values are: **sending** – message still in the internal queue of the Middleware waiting to be delivered (another call of 'set_message_status' will be required later to set the final status of the delivery) **delivered** – message successfully transferred to the back office system **failed** – transaction failed

Field	Required	Description
		sent – message sent but there is no way to confirm that it has reached the final recipient (for example E-mails) **Note:** these are the default statuses to be returned, though for each specific project they may be agreed at the implementation phase.
description	No	new value of the 'desc' field customer-specific additional value that along with Message Status can influence the flow of a message scenario e.g Message Status 'failed' can differ subject to its descriptions.
data	No	new value of the 'data' field
external_id	No	ID of the message in the external system
duration	No	new value of the 'duration' field
sent	No	new value of the 'sent' field
fault_attempt	No	number of the remaining attempts to resend the message in case the notification has failed this way the external system can change the number of the remaining attempts (e.g.stop or continue resending until success) unless there is a particular need to use the functionality, the field should be omitted, so the fault attempt logic remains in accordance to the corresponding message scenario step **Note:** resending in this case does not create a new message, the same message (with the same id) is being resent.
stop_further_attempts	No	this field should be set to '1' which means that notification attempts are stopped. No other values should be used
time_delivered_start	No	time delivered interval (promised to the customer) in HH:MM:SS format if 'status' returned is 'delivered' or 'sent', the fields are updated for activity/visit
time_delivered_end	No	

'set_message_status' Request Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:toatech:agent">
 <soapenv:Header/>
 <soapenv:Body>
 <urn:set_message_status>
 <user>
 <now>2011-11-23T15:50:23+00:00</now>
 <login>user_name</login>
 <company>company_name</company>
 <auth_string>67c5900a04abc54132a52da8a2320be2</auth_string>
 </user>
```

```
<messages>
<message>
<message_id>2006</message_id>
<status>failed</status>
<description>WRONG_TIME</description>
<data>Night time</data>
</message>
<message>
<message_id>2007</message_id>
<status>delivered</status>
<description>COMPLETED</description>
<data></data>
<duration>14</duration>
<sent>2011-11-29T12:54:22+00:00</sent>
</message>
<message>
<message_id>2008</message_id>
<status>failed</status>
<description>WRONG_TIME</description>
<data>Night time</data>
</message>
</messages>
</urn:set_message_status>
</soapenv:Body>
</soapenv:Envelope>
```

'set_message_status' Response

The 'set_message_status' response is an array of 'message_response' nodes. Each 'message_response' node contains the following elements:

Name	Required	Description
message_id	Yes	ID of the message
result	Yes	node that contains transaction result description

'result' Node of 'set_message_status' Response

Each result node contains the following elements

Field	Required	Description
code	Yes	Message status retrieval result code. Possible values are: NOT FOUND – message ID is unknown to the agent OK – message has been updated
desc	No	code description

'set_message_status' Response Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:toatech:agent">
```

```
<soapenv:Header/>
<soapenv:Body>
<ns1:set_message_status_response>
<message_response>
<message_id>2006</message_id>
<result>
<code>OK</code>
</result>
</message_response>
<message_response>
<message_id>2007</message_id>
<result>
<code>OK</code>
</result>
</message_response>
<message_response>
<message_id>2007</message_id>
<result>
<code>NOT FOUND</code>
</result>
</message_response>
</ns1:set_message_status_response>
</soapenv:Body>
</soapenv:Envelope>
```

7 Updating Properties and Processing Activities with 'data'

Updating Properties and Processing Activities with 'data'

A message processing result returned by an agent in the 'send_message' response or via the 'set_message_status' call, can be processed by the OFSC system to perform the following actions:

- update all company-defined properties of activity, inventory, resource, user, as well as a specific set of activity fields. Properties are updated by entity-related triggers, for example, resource properties are updated by resource-related triggers, etc.
- cancel activities and set non-scheduled activities

The fields to be assigned and the corresponding values are passed in the 'data' field.

The **#params?** string is used as a delimiter between 'data' itself and the passed parameters. The format of the parameter line is similar to URL. The **&** character is used as a delimiter between different parameters. Names and values of parameters are URL encoded.

Updating Fields and Properties

All company-defined properties of inventory and activity can be updated with the agent's response:

- To update the company-defined property the 'data' node should contain the following string:

```
#params?property label=value to be set
```

- For example to set 'cconfirmed' property to '1' the data should contain:

```
#params?cconfirmed=1
```

Only a predefined set of activity fields can be updated with the agent's response:

- The fields that can be updated are (values in the list a labels of the fields):
- email
- sms
- cell (synonym for 'sms')
- phone
- appt_number
- customer_number
- customer_name
- address
- city

- state
- zip
- To update the field from the list, the 'data' node should contain the following string:

```
#params?field label=value to be set
```

- For example to set 'phone' field to '123456' the data should contain:

```
#params?phone=123456
```

Note: fields 'address', 'city', 'state' and 'zip' are used by geocoding and, therefore, must contain valid values of the customer's address, city of residence, state and zip/post code. Other values will not be resolved correctly by the geocoding server fields 'cell' and 'phone' should contain only numbers. Their values are validated, and if any strings other than numbers have been entered, such strings are removed. If a value is entered as a string with no numbers, an empty value is set for the field

Managing Activities

Activities can be cancelled or set unscheduled with the agent's response:

- To cancel an activity the 'data' node should contain the following string:

```
#params?action=cancel_activity
```

- To set an activity unscheduled the 'data' node should contain the following string:

```
#params?action=unschedule_activity
```

Bulk Action

One middleware response can contain several updates, delimited with the '**&**' sign. For example, to set 'cconfirmed' property to '1', 'phone' field to '123456' and make activity unscheduled, the 'data' node should contain the following string:

```
#params?cconfirmed=1&phone=123456&action=unschedule_activity
```

Note: The total length of the 'data' field cannot exceed 255 characters.If a submitted 'data' value exceeds the limit, it can be correctly processed but will be truncated in the database.

8 Previous Versions

Previous Versions

The Outbound API in Oracle 16.2 is fully compatible with the Outbound API of ETAdirect versions 4.2, 4.3, 4.4 and 4.5. The only change is that the 'device' field of the message engine transactions has gone obsolete. If sent, the field will be ignored.

9 Appendix A

Appendix A – Middleware_Simple.WSDL

The file Middleware_Simple.WSDL should be provided as part of the SDK.

10 Appendix B

Appendix B – Middleware_Advanced.WSDL

The file Middleware_Advanced.WSDL should be provided as part of the SDK.

OFSC Glossary Keys

Oracle Product Abbreviations Keyword Map