OrderFlow API Documentation

Realtime Despatch Software Ltd

This document and its content is copyright of Realtime Despatch Software Limited. All rights reserved.
You may not, except with our express written permission, distribute, publish or commercially exploit the content.
Any reproduction of part or all of the contents in any form is prohibited.

Introduction

This document describes the standard XML interfaces used by OrderFlow to support realtime data exchanges with external systems.

The authoritative version of this document is held in the Realtime Despatch version control system. The document is subject to continual change; please check the current version with Realtime Despatch before relying on the contents.

OrderFlow provides a sophisticated XML interface with third party systems, the interface provides the following functionality:

- Searchable logs of all incoming and outgoing XML transactions
- Event driven triggers and CRON type schedules for XML data exports
- Configurable parsing of third party responses to identify application level errors
- Automated retrying of failed transactions

OrderFlow also supports the import and export of information in CSV data files, either manually or as part of automated scheduled processes. The import and export of CSV files is not covered by this document.

Guidelines for API Implementation

The OrderFlow API has been designed around a number of basic assumptions about the way in which data will be exchanged with external systems. The purpose of these assumptions is to ensure that the interfaces are resilient in the event of system downtime and network failure. All integrators that use the OrderFlow API should design their implementations with these assumptions in mind.

System Availability

OrderFlow does not assume that external systems with which it communicates will always be available.

Similarly, implementation of the operations described in this document should make allowance for the possibility that OrderFlow will have short periods of downtime.

Receipt of Message Retries

For outgoing messages to third party systems OrderFlow has a *retry* mechanism to handle the situation in which message delivery cannot be confirmed within a specified time.

Messages that exceed the maximum number of retries are brought to the attention of an administrator who is responsible for assessing the business impact of the repeated failure and taking appropriate action.

The number of retries and the delay between each retry is designed to ensure that when an external system is slow to respond, OrderFlow does not immediately increase the load placed upon the external system by retrying too rapidly or too frequently.

© Realtime Despatch Software Limited 2019

Sending of Message Retries

Similarly, in order to ensure business continuity in the event of system downtime, third party systems interfacing with OrderFlow should include a mechanism for retrying messages which could not be delivered following the period of inavailability.

The recommended strategy is to retry delivery of messages at specified intervals. As with delivery from OrderFlow, if the message cannot be delivered after a configured number of retries, a mechanism should be in place which allows the third party system to alert the system administrator of this situation.

Order of Message Delivery

OrderFlow does not assume that incoming messages will always be received in the same order in which they were sent. Similarly, the use of message retries from OrderFlow means that the order of message delivery to third party systems cannot be guaranteed in every case.

Where the API contains information that allows the receiving system to identify that the data contained within a message has been made obsolete by a subsequent message, this obsolete message should be ignored. The OrderFlow to external system messages described in the section Operations FROM OrderFlow (PUSH) include a message id in each outgoing message, which can be used to determine the sequencing order of messages.

Idempotency

Third party systems which receive and trigger actions on the basis of notifications received from OrderFlow should be *idempotent*, in the sense that a second request to trigger an action that has *already been completed* should be ignored when appropriate.

The necessity for idempotent messages arises as a result of the retry mechanism. If a retry is triggered because a previous attempt to deliver a message timed out, the receiving system should be able to detect whether the retry message had in fact been received on the first attempt. It should not attempt to retry the action without taking into account the possibility that the action may have already been completed.

HTTP Authentication

The XML interface is password protected using HTTP BASIC authentication and/or custom authentication.

HTTP Basic Authentication

Implemented as described in RFC 2617 and on Wikipedia. Note that the security realm is "RTD", if this is required by the HTTP client. Custom Authentication is supported through the user and password which should be transferred using the HTTP request headers (user and password, respectively). The password should be base64 encoded. In both cases, communication is assumed to be via HTTPS, without which the security of these authentication schemes would be inadequate.

Channel and Organisation Authentication

In most environments it is necessary to authenticate a user by **channel**. For example, one channel might be represent orders associated with a particular retailer from a shopping cart backed web site, and another might represent orders coming from that retailer's Amazon site. In order to facilitate channel authentication, the channel code needs to be passed to OrderFlow in all requests.

© Realtime Despatch Software Limited 2019

On OrderFlow, a channel will belong to an organisation. For this reason for some operations it is necessary to be authenticated at the **organisation** level.

For *HTTP POST* operations which involve posting a request body rather than a www-urlencoded form, the recommended usage is to pass the channel using a HTTP header named `channel`, and to pass in the organisation using a header named `organisation`.

For *HTTP GET* operations, the same mechanism is available, although it is also possible to pass the channel and organisation via HTTP request parameters in the query string itself. Note that if the channel or organisation is supplied both as request parameters and as headers, then the values will be taken from the respective request parameters.

Message Conventions

URL: The form of a URL used to connect to OrderFlow is shown below:

`https://[host]/[instanceName]/[moduleName]/[messageName].xml`

The `host` is the HTTPS URL associated with the server on which the OrderFlow application is running.

The `instancename` is simply the particular instance of OrderFlow on the host, and will be environment-specific.

The `moduleName` is the qualifier which points to the part of the application in which the functionality lies.

The `messageName` is the name of the particular message, and may consist of more than one segment (e.g. for imports, all URLs end with `.xml`).

From this point onwards only the last part of the URL is shown, from the module name onwards: `/remoteorder/imports/importitems.xml`

The first part of the URL will be specific to the instance of OrderFlow being addressed.

Capitalization: applies to both URLs as well as the contents of the messages themselves, that is, both elements and attributes.

- First letter always begins with lower case.
- After this, camel case is used. Each new word which is part of the same identifier starts with an upper case letter.
- Hyphens and underscores are not used in XML elements or attributes. For example, and element containing the last modified date would be named `lastModifiedDate` rather than `last-modified-date`.

HTTP Operation types

PULL

PULL operations are invoked by the third party application (the client) on service interfaces published by OrderFlow.

These can be invoked by remote clients to OrderFlow. OrderFlow provides the endpoints to PULL operations.

See the Operations TO OrderFlow (PULL).

© Realtime Despatch Software Limited 2019

PUSH

PUSH operations are invoked by OrderFlow to push data to the third party application (the client), typically an e-commerce shopping cart application or accounts system.

For PUSH operations OrderFlow functions as the *client* in the interaction.

Push notifications can either be realtime or scheduled, or can be triggered by events on OrderFlow. For scheduled notifications, URLs and invocation schedule are configured per organisation or per channel for operation type.

The operations will involve notifications over a given period. The receiver is expected to either accept or reject the notification. If rejected, or if the notification target server is not available, then the same notifications will be attempted at the next notification schedule time. Push operation which involve posting of data are considered to have succeeded if the target server returns an error code of `200` or if the content of the acknowledgement returned by the client matches the rules defined in the remote message definition.

> **A note on existing third party APIs**
>
> As OrderFlow is not responsible for defining and hosting PUSH operations, the API operations as described below will only in certain circumstances. Specifically, developers can use this guide as a basis for implementing extensions to third party systems that interface with OrderFlow.
>
> OrderFlow is also able to integrate using pre-existing APIs defined by other third party applications through a configurable integration framework. The details for this kind of integration are not described here. The current document covers only the 'generic integration' for PUSH operations.

Operations TO OrderFlow (PULL)

Import Operations

Product Import

A mechanism for adding new products into the OrderFlow database, and for updating existing products (keyed on product code).

Products should first be defined within the third party application. The process that allows a user to create a new product should always associate it with a mandatory unique SKU (e.g. each size colour combination must have it's own SKU code).

"Bundles" of SKUs that might be represented in the third party system by a unique "Parent" SKU should not be passed to OrderFlow. The SKUs sent to OrderFlow should be the individual products rather than logical groupings of items which might be received and stored separately.

"Virtual" products that are not associated with physical items of stock in the warehouse should not be sent to OrderFlow.

When a new SKU is created within the third party system the SKU code, description and (optionally) any other relevant details should be passed to OrderFlow via the XML interface. The OrderFlow system should then create a new product code for the retailer (sales channel) and return an XML confirmation. The SKU code should be unique to the retailer, if the 'shared' flag is set as part of the product creation process the SKU code should be unique across the OrderFlow system.

Note that `multi-line` properties can be supported: simply use the string `[BR]` to represent line breaks.

© Realtime Despatch Software Limited 2019

Operation Summary
Invocation Called from client to OrderFlow as required
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header above will be the reference on OrderFlow for the organisation to which the product will belong. An example request body for the Product Import operation invocation is shown below.

© Realtime Despatch Software Limited 2019

```
<imports>
  <import type="product" operation="insert" externalReference="TEST_fullproduct">
    externalReference=TEST_fullproduct
    description=A description for the test product
    weight=100
    weightUnits=grams
    imageReference=TEST_fullproduct.gif
    type=default
    quantityOnOrder=10
    priceNet=10.50
    priceGross=11.50
    tax=1.50
    taxCode=T1
    currency=GBP
    currencyUnits=pounds
    costNet=10.50
    costGross=11.50
    costTax=1.50
    costTaxCode=T1
    costCurrency=GBP
    costCurrencyUnits=pounds
    userDefined1=User defined field value 1
    userDefined2=
    userDefined3=
    userDefined4=
    userDefined5=
    channel=MYCHANNEL
    type=default
    activated=true
  </import>
  <import type="product" operation="insert" externalReference="TEST_min_product">
    externalReference=TEST_min_product
    description=A description for the min product
    organisation=supershop
    type=default
    activated=true
  </import>
  <import type="product" operation="insert" externalReference="TEST_global_product">
    externalReference=TEST_global_product
    description=A description for the global product
    type=default
    activated=true
  </import>
</imports>
```

For a successful product import, OrderFlow will return output as shown below.

```
<importResult>
  <importSuccesses>
    <import type="product" operation="insert" externalReference="TEST_fullproduct"
      entity="rtd.domain.database.Product" item="rtd.domain.database.Product@172c384b"
      queryTime="2014-08-02 12:20:33.575" />
    <import type="product" operation="insert" externalReference="TEST_min_product"
      entity="rtd.domain.database.Product" item="rtd.domain.database.Product@c4983d8"
      queryTime="2014-08-02 12:20:33.596" />
    <import type="product" operation="insert" externalReference="TEST_global_product"
      entity="rtd.domain.database.Product" item="rtd.domain.database.Product@629bfaff"
      queryTime="2014-08-02 12:20:33.617" />
  </importSuccesses>
  <importFailures>
  </importFailures>
</importResult>
```

© Realtime Despatch Software Limited 2019

Note that failures are recorded per imported item. This happens when the data cannot be imported for some reason, for example if the imported item refers to data which is not present on the system.

The system also identifies and rejects *duplicates* when an attempt is made to reimport data which is already on the system. Similarly to failures, duplicates are also recorded separately using an `importDuplicates` element.

```
<importResult>
  <importSuccesses>
  </importSuccesses>
  <importFailures>
    <import type="product" operation="insert" externalReference="TEST_fullprosduct"
      queryTime="2014-08-02 12:31:35.582">
      <failureMessage>Unable to find
        any entity associated with identifier:
        ref:channel:missing_channel</failureMessage>
      <failureDetail>Unable to find any entity associated
        with identifier: ref:channel:missing_channel
        (rtd.exceptions.ImportMissingDataException)
        ...</failureDetail>
    </import>
  </importFailures>
  <importDuplicates>
    <import type="product" operation="insert" externalReference="TEST_min_product"
      queryTime="2014-08-02 12:31:35.717">
      <duplicateMessage>An instance
        of product with reference
        &amp;apos;TEST_min_product&amp;apos; is already present in
        the database.</duplicateMessage>
      <duplicateDetail>An
        instance of product with reference
        &amp;apos;TEST_min_product&amp;apos; is already present in
        the database.
        (rtd.exceptions.ValidationErrorException)
        ...</duplicateDetail>
    </import>
    <import type="product" operation="insert" externalReference="TEST_global_product"
      queryTime="2014-08-02
          12:31:35.784">
      <duplicateMessage>An instance
        of product with reference
        &amp;apos;TEST_global_product&amp;apos; is already present in
        the database.</duplicateMessage>
      <duplicateDetail>An
        instance of product with reference
        &amp;apos;TEST_global_product&amp;apos; is already present in
        the database.
        (rtd.exceptions.ValidationErrorException)
        ...</duplicateDetail>
    </import>
  </importDuplicates>
</importResult>
```

Note that temporary products can be imported directly using the Order Import operation. However, using this operation it is only possible to specify the product code (external reference) and the description.

See Product Import Fields for more details on fields which can be used for this operation.

© Realtime Despatch Software Limited 2019

Product Update

If a product description or other details are edited in the third party application, the new description or other values should be passed to OrderFlow using the Product Update operation.

The format of the message is the same as the product import but OrderFlow will apply a product **update** rather than an **insert**.

Operation Summary
Invocation Called from client to OrderFlow as required
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header will be the reference on OrderFlow for the organisation to which the product belongs.

The Product Update will make a request very similar to that of the Product Import, the key difference being the `operation` attribute.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
    <import type="product" operation="update" externalReference="TEST_fullproduct">
        externalReference=TEST_fullproduct
        description=An updated description for the test product
        weight=250
    </import>
</imports>
```

See Product Import Fields for more details on fields which can be used for this operation.

Order Import

The order import defines a mechanism for importing orders supplied as an XML document. The structure of the invocation is identical to the Product Import described earlier. All that differs is the URL used, and the contents of XML posted to the OrderFlow server.

© Realtime Despatch Software Limited 2019

Operation Summary
Invocation Called from client to OrderFlow as required
Method HTTP POST
URL /remoteorder/imports/importitems.xml
Header `channel`

The value for the `channel` HTTP header will be the reference on OrderFlow for the sales channel for the incoming order.

An example of a minimal Order Import input is shown below:

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="order" operation="insert" externalReference="TEST_order1">
    state=created
    validated=true
    paymentTransactionInfo=999
    customerComment=A comment from the customer
    totalPriceNet=10.0
    totalPriceGross=12.0
    totalTax=2.0
    totalTaxCode=T1
    shippingPriceNet=2.5
    shippingPriceGross=2.0
    shippingTax=2.0
    shippingTaxCode=T1
    currency=GBP
    currencyUnits=pounds
    placed=2014-08-31 07:16:02
    authorised=2014-08-31 08:16:10
    source=ebay
    channel=MYCHANNEL
    campaign=campaign_reference
    deliveryAddressLine1=Granvilla
    deliveryAddressLine2=Melton Hill
    deliveryAddressLine3=Woodbridge
    deliveryAddressLine4=Suffolk
    deliveryAddressLine5=
    deliveryAddressLine6=
    deliveryCountryCode=UK
    deliveryPostCode=IP12 1AX
    deliveryContactName=Phil Zoio
    deliveryEmailAddress=phil@realtimedespatch.co.uk
    deliveryDayPhoneNumber=01394 384181
    deliveryEveningPhoneNumber=01394 385000
    deliveryMobilePhoneNumber=07595 524200
    deliveryFaxNumber=01394 384181
    deliveryCompanyName=Realtime Despatch
    invoiceAddressLine1=Realtime Despatch
    invoiceAddressLine2=2 Manor Cottages
    invoiceAddressLine3=Swindon Road
    invoiceAddressLine4=Kington Langley
    invoiceAddressLine5=Chippenham
    invoiceAddressLine6=Wiltshire
```

© Realtime Despatch Software Limited 2019

```
invoiceCountryCode=UK
invoicePostCode=SN15 5ND
invoiceContactName=Charlie Armor
invoiceEmailAddress=charlie@realtimedespatch.co.uk
invoiceDayPhoneNumber=01249 750564
invoiceEveningPhoneNumber=01249 750564
invoiceMobilePhoneNumber=
invoiceFaxNumber=01249 750564
invoiceCompanyName=Realtime Despatch
userDefined1=ud1
userDefined2=ud2
userDefined3=ud3
userDefined4=ud4
userDefined5=ud5

shipment.externalReference=TEST_myref_1
shipment.state=ready
shipment.earliestShipDate=1999-12-31
shipment.deliveryInstruction=Please leave with neighbour if nobody at home
shipment.despatchComment=
shipment.despatchReference=
shipment.weight=120
shipment.weightUnits=grams
shipment.itemCount=1
shipment.addressLine1=
shipment.addressLine2=
shipment.addressLine3=
shipment.addressLine4=
shipment.addressLine5=
shipment.addressLine6=
shipment.countryCode=
shipment.postCode=
shipment.contactName=
shipment.emailAddress=
shipment.dayPhoneNumber=
shipment.eveningPhoneNumber=
shipment.mobilePhoneNumber=
shipment.faxNumber=
shipment.companyName=
shipment.userDefined1=
shipment.userDefined2=
shipment.userDefined3=
shipment.userDefined4=
shipment.userDefined5=
shipment.deliverySuggestionCode=express
shipment.deliverySuggestionName=Express
shipment.orderItem=entity:order

orderLine.1.product.externalReference=DVD-BELOVED
orderLine.1.quantity=10
orderLine.1.state=created
orderLine.1.totalPriceNet=4.0
orderLine.1.totalPriceGross=5.0
orderLine.1.totalTax=1.0
orderLine.1.totalTaxCode=T1
orderLine.1.unitPriceNet=
orderLine.1.unitPriceGross=
orderLine.1.unitTax=
orderLine.1.unitTaxCode=
orderLine.1.userDefined1=
orderLine.1.userDefined2=
orderLine.1.userDefined3=
orderLine.1.userDefined4=
orderLine.1.userDefined5=

orderLine.2.product.externalReference=DVD-MATR
```

© Realtime Despatch Software Limited 2019

```
    orderLine.2.quantity=20
    orderLine.2.state=created
    orderLine.2.totalPriceNet=6.0
    orderLine.2.totalPriceGross=7.0
    orderLine.2.totalTax=1.0
    orderLine.2.totalTaxCode=T2
    orderLine.2.unitPriceNet=
    orderLine.2.unitPriceGross=
    orderLine.2.unitTax=
    orderLine.2.unitTaxCode=
    orderLine.2.userDefined1=
    orderLine.2.userDefined2=
    orderLine.2.userDefined3=
    orderLine.2.userDefined4=
    orderLine.2.userDefined5=

    orderLine.1.shipment=entity:shipment
    orderLine.2.shipment=entity:shipment

    orderAttribute.1.name=TEST_att1
    orderAttribute.1.title=Attribute 1
    orderAttribute.1.value=Attribute Value 1
    orderAttribute.1.orderItem=entity:order

    orderAttribute.2.name=TEST_att2
    orderAttribute.2.title=Attribute 2
    orderAttribute.2.value=Attribute Value 2
    orderAttribute.2.orderItem=entity:order
  </import>

  <import type="order" operation="insert" externalReference="TEST_minorder">
  #additional order properties as above
  ...
  </import>


  <import type="order" operation="insert" externalReference="TEST_with_product">
  #additional order properties as above
  ...
  </import>
</imports>
```

Note that `campaign` is an optional field (introduced in 3.8.0) which can be used to associate the incoming purchase order with an existing campaign.

See Order Import Fields for more details on fields which can be used for this operation.

Example Output: As with Product Import.

REFERRING TO PRODUCTS

Note that arbitrary order attributes can be imported using name value pairs as shown in the example above.

As the above example shows, there are two ways of referring to products. One is to refer to existing products within the system, using a line such as

```
orderLine.1.product.externalReference=DVD-BELOVED
```

The other is to import products as part of the order import, as shown in the snippet below.

© Realtime Despatch Software Limited 2019

```
product.1.externalReference=TEST_orderproduct1
product.1.description=A description for the new product 1
product.1.type=default
product.1.activated=true

orderLine.1.product=entity:product.1
```

Note that products need to be given a type attribute. For products which are only temporary, that is, for which no stock is to be checked into the warehouse, the type entry might be written as:

`product.1.type=temporary`

The range of properties available for the product entries is the same as is available for the standalone product import. However, the imported product must also have the product (and possibly numbered index) prefix, as the default imported entity in this case is order, not product.

line items are indexed, as shown for example in the expression

`orderLine.1.quantity=10`

Note that the referenced products, delivery codes, packing options, etc. need to be present and active in the database for the import to work successfully.

As with products, if multiple orders are imported, it is possible for some to import successfully and others to fail import. Again as with products, errors for particular items are specified using the importFailures element.

Note that multi-line properties can be supported: simply use the string [BR] to represent line breaks.

MERGING ORDERS WITH PRODUCT DEFINITIONS

It is possible to merge existing product definitions using the Order Import operation. This is useful in the case where the products may or may not be known about at the time of order import. In this case, the merge operation should be used for the order import, as shown below.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="order" operation="merge" externalReference="TEST_myref">
    ...
    product.1.externalReference=TEST_orderproduct1
    product.1.description=A description for the new product 1
    product.1.type=temporary
    product.1.activated=true
    ...
  </import>
</imports>
```

In the example above, the product TEST_orderproduct1 may or may not be know to Realtime Despatch at the time of order import.

Note that orders and their associated order lines and shipments cannot be updated using the merge operation.

© Realtime Despatch Software Limited 2019

Supplier Purchase Order Import

The Supplier Purchase Order allows incoming deliveries to be associated with the purchase order that was used to order the incoming stock from a supplier. Multiple deliveries may be associated with one purchase order.

Operation Summary
Invocation From 3rd party system to OrderFlow
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header will be the reference on OrderFlow for the organisation to which the purchase order applies.

OrderFlow can be configured to return details of incoming deliveries to the external system from which the associated supplier purchase order was received.

The following optional restrictions can be enforced by the OrderFlow configuration if appropriate:

- Incoming deliveries that have been associated with a supplier purchase order can be restricted to the products contained in the order.
- Incoming deliveries that have been associated with a supplier purchase order can be restricted to the product quantities contained in the order.
- Supplier purchase orders associated with suppliers not already defined within OrderFlow can automatically create a new supplier definition.

Example **input** is shown below.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="purchaseOrder" operation="insert"
    externalReference="TEST_po">
    purchaseOrder.supplierReference=TEST_purchaseOrder
    purchaseOrder.supplier=HAMA
    purchaseOrder.site=WAREHOUSE_1
    purchaseOrder.campaign=TEST_campaign_reference

    purchaseOrderLine.1.product=DVD-ABUG
    purchaseOrderLine.1.quantity=1
    purchaseOrderLine.1.purchaseOrder=purchaseOrder
    purchaseOrderLine.1.externalReference=poLine1

    purchaseOrderLine.2.product=DVD-FRAN
    purchaseOrderLine.2.quantity=1
    purchaseOrderLine.2.purchaseOrder=purchaseOrder
    purchaseOrderLine.2.externalReference=poLine2
```

© Realtime Despatch Software Limited 2019

```
  </import>
</imports>
```

Note that supplier purchase orders can also be imported as CSV documents through the CSV import interface. In this case, the document is not restricted to using the native format above.

Note that `site` is an optional field which is only applicable in multi-warehouse environments when the target site for the purchase order is known prior at the point of placing the purchase order. For single warehouse environments, the default site or warehouse will be used automatically.

Note that `campaign` is an optional field (introduced in 3.8.0) which can be used to associate the incoming purchase order with an existing campaign.

For certain workflows, it is also possible to associate purchase orders with sales orders (shipments). This may include shipments that have been received already, as well as shipments that are still expected to be received in orderFlow.

Adding associated shipment references in this way is simple, as shown in the example below:

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="purchaseOrder" operation="insert" ...>
     <!-- ... example as above -->

     purchaseOrderShipment.1.shipmentReference=SHIP00011
     purchaseOrderShipment.2.shipmentReference=SHIP00012
  </import>
</imports>
```

Advanced Shipping Note (ASN) Import

The Advanced Shipping Note (ASN) defines a mechanism by which incoming deliveries can be associated with existing orders. This provides the opportunity to avoid the overhead of checking stock into the warehouse, as items contained within an ASN delivery can be fed directly into order processing. Two types of ASNs are supported: Just-in-time ASNs, for which the individuals lines are associated with orders to be processed at the time of delivery, and Stock ASNs, for which the delivered items will be checked into the warehouse.

Operation Summary
Invocation From 3rd party system to OrderFlow
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header will be the reference on OrderFlow for the organisation to which the ASN applies.

The format of the ASN import is very similar to that of the product and order import. The differences lie in the content of the data.

© Realtime Despatch Software Limited 2019

The example below is for a **stock ASN**, for which the associated delivery contents are expected to be placed in stock.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="asn" operation="insert">
    asn.supplierReference=TEST_asn
    asn.supplier=abdc
    asn.site=WAREHOUSE_1
    asnLine.1.product=DVD-BELOVED
    asnLine.1.quantity=10
    asnLine.1.advancedShippingNote=asn
    asnLine.2.product=DVD-UNSG
    asnLine.2.quantity=20
    asnLine.2.advancedShippingNote=asn
  </import>
</imports>
```

The example below is for a **Just-in-Time ASN**, for which the associated delivery contents are expected to be associated with outstanding orders through a *cross docking process*.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="asn" operation="insert">
    asn.supplierReference=TEST_asn
    asn.supplier=abdc
    asn.site=WAREHOUSE_1
    asnLine.1.product=DVD-BELOVED
    asnLine.1.quantity=10
    asnLine.1.advancedShippingNote=asn
    asnLine.1.orderItem=100
    asnLine.2.product=DVD-UNSG
    asnLine.2.quantity=20
    asnLine.2.advancedShippingNote=asn
    asnLine.2.orderItem=100
  </import>
</imports>
```

The third party system is responsible for providing the supplier reference for the ASN. The combination of supplier and supplier reference needs to be unique within OrderFlow. The supplier field itself is optional.

Note that the orderItem attribute used in the Just-in-Time ASN refers to the order which is identified using the `externalReference` attribute in the Order Import operation.

The following additional assumptions underlie the advance shipping note functionality. Some of these assumptions may be relaxed in future releases of OrderFlow:

- At the time when the ASN is received by OrderFlow, all of the products associated with the ASN will already be known to OrderFlow. This may either be through a Product Import_, or implicitly via an Order Import.
- If the ASN is for just-in-time orders, then all of the orders will be known to OrderFlow.
- For a single ASN, all the lines will either be purely for just-in-time orders, or purely for stock check-in. In other words, the ASN will not contain a mixture of just-in-time and stock products.
- For any order that the ASN covers, it should include in the same ASN the items required to satisfy all of the line items in that order. It should not, for example, be necessary to retrieve product items from stock to process an order contained within the ASN.

© Realtime Despatch Software Limited 2019

- Orders referred to using ASNs cannot have multiple line items using the same SKU/product code.
- The ASN is only supported for single shipment orders.
- The ASN will be received electronically prior to the physical delivery of the items.

Note that the `site` is only required in multi-warehouse environments. For single warehouse environments, the default site or warehouse will be used automatically.

When receiving the ASN, OrderFlow will make sure the above conditions are satisfied, and in doing so will determine to which order lines each of the ASN line entries should be applied.

Note on the physical processing of the ASN

The note that follows is not part of the XML interface, but gives some background to how the contents of the ASN document will be used.

When the goods are received, the paperwork which accompanies the physical delivery should contain the ASN reference, as in the example below.

```
ASN: ASN_BARCODE
prod1 (The description for prod1)
  Quantity: 10
prod2 (The description for prod2)
  Quantity: 2
prod1 (The description for prod1)
  Quantity: 4
prod3 (The description for prod3)
  Quantity: 1
```

For each ASN line item, the product code will be inputted or scanned in for each item in the delivery. For just-in-time ASNs, when all of the lines associated with a particular order have been received, the user is directed to the packing screen for that order.

Returns Import

The Return Item import defines a mechanism for importing customer returns remotely, for example, via a Point of Sale (PoS) system. Multiple Return Lines may be associated with a Return Item.

Operation Summary
Invocation From 3rd party system to OrderFlow
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header will be the reference on OrderFlow for the organisation to which the return applies.

© Realtime Despatch Software Limited 2019

The example below is for a Return Item with two Return Lines:

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
<import type="return" operation="insert">
    return.orderReference=order_reference
    return.authorisation=TEST_return
    return.authorised=true
    return.type=goods_not_delivered
    return.storeId=001
    return.returnDate=2015-01-21 16:35:17
    return.user=philz
    return.organisation=myco
    return.site=WAREHOUSE_1
    return.note=Customer did not like colour
    #The line below is optional
    #return.channel=MYCHANNEL
    returnLine.1.quantity=1
    returnLine.1.reason=A - Not wanted
    returnLine.1.condition=As new
    returnLine.1.product=DVD-BELOVED
    returnLine.1.returnItem=return
    returnLine.2.quantity=1
    returnLine.2.reason= B - Not as described
    returnLine.2.condition=Packaging damaged
    returnLine.2.product=DVD-FRAN
    returnLine.2.returnItem=return
</import>
</imports>
```

Common elements such as product and channel follow the standard OrderFlow API conventions. Some additional detail on the XML Return Item and Line fields:

- orderReference: The order number that this Return refers to.
- authorisation: Either the authorisation reference or order number (depending on OrderFlow configuration). Must have a value if no `orderReference` is specified.
- authorised: Whether the return is authorised.
- type: The type of the return (e.g. 'goods_not_delivered', 'goods_damaged', 'unknown'). Note that this is configurable.
- storeId: The ID of the store from which the return is being made.
- note: Note to be added to return.
- reason: Reason for the return, for example one of: [A - Not wanted, B - Not as described, C - Wrong size, D - Wrong product sent, E - Quality/Manufacturing fault, F - Damaged in transit, G - Late arrival, H - Other]. Note that this is configurable.
- condition: Condition of the item(s) being returned, for example one of: [As new, Packaging damaged, Product damaged (refurbishable), Product damaged (irreparable)]. Note that this is configurable.
  Note that the

site` is only required in multi-warehouse environments. For single warehouse environments, the default site or warehouse will be used automatically.

© Realtime Despatch Software Limited 2019

Campaign Import

The Campaign import defines a mechanism for importing campaigns remotely. Multiple Campaign Lines can be associated with a Campaign, and multiple Campaigns can be defined in a single import. Campaigns are only present in OrderFlow from version 3.7.9.

Operation Summary
Invocation From 3rd party system to OrderFlow
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `channel`

The value for the `channel` HTTP header will be the reference on OrderFlow for the sales channel to which the campaign will belong.

The example below is for a Campaign with two Campaign Lines:

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
<import type="campaign" operation="insert" externalReference="campaign_reference">
    externalReference=campaign_reference
    name=campaign_name
    description=TEST campaign description
    channel=acmeweb
    startDate=2017-06-20
    breakDate=2017-06-24
    endDate=2017-06-25
    state=created
    campaignLine.1.quantity=5
    campaignLine.1.product.externalReference=ipod5
    campaignLine.1.campaign=entity:campaign
    campaignLine.2.quantity=2
    campaignLine.2.product.externalReference=star_wars_4
    campaignLine.2.campaign=entity:campaign
</import>
</imports>
```

Common elements such as product and channel follow the standard OrderFlow API conventions. Some additional detail on the XML Campaign fields:

- startDate: The 'scheduled' start date of the campaign, when it effectively becomes active.
- breakDate: The 'scheduled' date by which the primary activity on the campaign needs to be completed. SLAs can be measured against this date.
- endDate: The date at which the campaign needs to be terminated.

© Realtime Despatch Software Limited 2019

Stock Move Task Import

The Stock Move Task import provides a mechanism for a third party system to initiate a sequence of stock moves within OrderFlow.

The stock moves may be for a variety of purposes. For example, they may involve the transfer of damaged stock from quarantine to outgoing locations prior to being returned to a supplier. In another example, the stock moves may entail the movement of items within an eCommerce warehouse to outgoing locations for subsequent shipment to stores.

Operation Summary
Invocation From 3rd party system to OrderFlow
Method HTTP POST
URL /remotewarehouse/imports/importitems.xml
Header `organisation`

The value for the `organisation` HTTP header will be the reference on OrderFlow for the organisation to which the products reference in the task belong.

The example below shows a third party task import to return some damaged stock to a supplier.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
<import type="stockMoveTask" operation="insert">
externalReference=Widgets_RTN037625
taskDefinition=return_damaged_to_supplier
supplier=WidgetsRUs
site=Default
stockMoveLine.1.product=DVD-MATR
stockMoveLine.1.suggestedQuantity=1
stockMoveLine.1.thirdPartyReference=037625-01
stockMoveLine.1.stockMoveTask=stockMoveTask
stockMoveLine.2.product=DVD-ABUG
stockMoveLine.2.suggestedQuantity=2
stockMoveLine.2.thirdPartyReference=037625-02
stockMoveLine.2.stockMoveTask=stockMoveTask
</import>
</imports>
```

And this example shows a third party task import to transfer stock to a different warehouse.

```
<?xml version="1.0" encoding="UTF8"?>
<imports>
<import type="stockMoveTask" operation="insert">
externalReference=Calais_095441
taskDefinition=outgoing_stock_transfer
site=Default
targetSite=Calais
```

© Realtime Despatch Software Limited 2019

```
stockMoveLine.1.product=DVD-MATR
stockMoveLine.1.suggestedQuantity=1
stockMoveLine.1.thirdPartyReference=095441-01
stockMoveLine.1.stockMoveTask=stockMoveTask
stockMoveLine.2.product=DVD-ABUG
stockMoveLine.2.suggestedQuantity=2
stockMoveLine.2.thirdPartyReference=095441-01
stockMoveLine.2.stockMoveTask=stockMoveTask
</import>
</imports>
```

The following assumptions underlie the stock move task import functionality. Some of these assumptions may be relaxed in future releases of OrderFlow:

- The third party system is responsible for providing its corresponding task reference (`externalReference`).
- At the time when the task is received by OrderFlow, all of the products associated with the task will already be known to OrderFlow. This may be either through a Product Import_, or implicitly via an Order Import.

Note that the `site` is only required in multi-warehouse environments. For single warehouse environments, the default site or warehouse will be used automatically.

The `taskDefinition` determines how the system treats the lines once they have been imported. This is explained in a bit more detail in the next section.

Note on the processing of the imported task

The note that follows is not part of the XML interface, but gives some background to how the imported task will be actioned on OrderFlow.

When the task is imported, OrderFlow will create a stock move task of the relevant type, and capture the `required lines` (supplied in the XML) for that task. A background job will then run that will match the required lines for each imported task with the appropriate source (picking) and target (putaway) locations, thereby creating the *source* and *target* lines for the task.

Once the source and target lines have been identified, OrderFlow marks the stock move task as `ready` , allowing it to be processed in the same way as other stock move tasks on the system.

Imports using Custom Formats

The imports described above all assume that the *native* OrderFlow format is used. OrderFlow also support the import of data via other text based formats. For example, it is possible to support imports of data supplied as CSV files. Custom XML-based data formats can be accommodated using an XSLT transformation. Indeed, almost any text based format can be accommodated.

© Realtime Despatch Software Limited 2019

	Operation Summary
	Invocation Called from client to OrderFlow as required
	Method HTTP POST
	Example URL /remoteorder/imports/transformitems.xml
Parameters	*content*: the content of the document being imported.
	handler: used to identify the transformation to be applied on the incoming data.
	operation: indicates the operation to be applied. Only required in situations where the operation is not already specified within the incoming document. If supplied, the value needs to be `insert`, `update` or `merge`.

Note that the 'magic' that takes place is in the setup of the *handler*, which needs to be correctly identified for the incoming data. Different handlers will be configured in different ways, depending on the format of the incoming data. For example, if the incoming format is CSV, the input handler may define mappings from the source fields generated by the third party system to the target fields understood by OrderFlow. The details of how the input handler is configured is outside of the scope of this document.

Note that the organisation scope of the operation above is determined using the `organisation` and/or `channel` header or parameters supplied as described in the Authentication section.

© Realtime Despatch Software Limited 2019

Product Operations

Inventory Pull

Third party applications can check the inventory level of a single of product or set of products within OrderFlow. The SKU codes and "available stock" figures are returned. Depending on the filters used, this operation can return the inventory level for just a single product, a particular set of products or for all products in the inventory for a particular retailer.

	Operation Summary
	Invocation Called from client to OrderFlow as required
	Method HTTP POST
	Example URL /remotewarehouse/inventory.xml?channel=ch1&externalReferences=pr1,pr2,pr3
Parameters	*channel*: retrieves all products for a particular channel, identified by reference (mandatory)
	site: if multi-site inventory is enabled, then identifies the site for which inventory is required (mandatory)
	externalReferences: product codes for which to retrieve inventories (optional). If no externalReferences are supplied the system will return all products associated with the channel, for which inventory exists.

The Inventory Pull operation will result in OrderFlow returning output as shown below.

```
<inventory>
<product inventoryId="615" externalReference="DVD-BLDRNDC" site="global"
         total="1" allocated="1" available="-1" frozen="0" .../>
<product inventoryId="616" externalReference="DVD-BLDRNDC" site="DEFAULT"
         total="1" allocated="1" available="-1" frozen="0"/>
<product inventoryId="617" externalReference="DVD-BLDRNDC" site="SECOND"
         total="0" allocated="0" available="0" frozen="0"/>
</inventory>
```

As indicated using `...` , additional fields have been added to the inventory output. As with 4.0.3, these include:

```
<product ... inspection="0" damaged="0" quarantined="0"
             onOrder="0" lastStockChangeId="0" lastLineRequirementChangeId="0"/>
```

The key figure is the available `available` quantity, which will be zero or negative where a product is out of stock.

Note that the *site* parameter is only used in multi-site environments for which multi-site inventory is enabled. In these environments, optional values for site include:

- *global*: returns the inventory values calculated across all sites on the system.

© Realtime Despatch Software Limited 2019

- *any*: returns the inventory values for the 'global' site as well as for any other site for which the user has access permission.
- by site reference, allows inventory values for a specific site to be retrieved.

Note that if multi-site inventory is supported, an no site parameter is supplied, then an inventory record calculated across all sites on the system is returned.

Order Operations

The OrderFlow API can be used to create orders and optionally, to specify whether the line items within an order are spread across multiple shipments.

Orders can be cancelled through the API if the order state (out of stock, picked, etc.) is one that allows order cancellation. This business logic is configurable within OrderFlow.

Order Detail Pull

The OrderFlow API can be used to query the details on a particular order. This operation can be used to pick up all relevant information which pertains to a single order at a particular point in time, including the current order and associated shipment states, the availability of individual line items, the earliest ship date, the despatch reference, etc.

Operation Summary
Invocation Called from client to OrderFlow as required.
Method HTTP GET
URL /remoteorder/order/detail.xml?externalReference=81

The main order state values are the following:

Order State	Description
created	The order has been added to the system but not yet processed.
despatched	All shipments within the order have been packed.
deleted	an order is marked for deletion. Note that deleted orders are not accessible via the API.

Note that additional order states may be defined to support specific requirements within a particular instance of OrderFlow

© Realtime Despatch Software Limited 2019

Most of the order processing state tracking is done through shipments. Every order must contain at least one shipment, but it is also possible to support partial despatch of orders through multiple shipments. A selection of the applicable shipments states are shown below:

Shipment State	Description
created	The order and shipment have been added to the system but not yet processed.
ready	The shipment is ready for processing, but processing of the shipment has not started. At this point, no stock will have been allocated to the shipment.
allocated	Stock has been allocated for all of the order lines in the shipment. Note that if one or more of the order lines is out of stock, the shipment will remain in the 'ready' state.
on_hold	The shipment has been placed on hold, because the earliest ship date is in the future.
picked	All order lines for the shipment have been picked, but the shipment has not yet been packed.
packed	The shipment has been packed but not yet despatched.
despatched	The shipment has been despatched.
deleted	The shipment has been marked for deletion. Note that deleted shipments are not accessible via the API.

In some circumstances it is necessary to track state changes at the order line level. Some of the states that apply for order lines:

Order Line State Description

---

created The order line has been added to the system but not yet processed. allocated The stock required for the order line has been allocated. picked The stock required for the order line has been picked. out_of_stock The stock required for the order line is not available. packed The order line has been packed. deleted The order line has been marked for deletion. Note that deleted order lines are not accessible via the API.

Example output is show below.

```
<order externalReference="81" state="despatched">
 <shipments>
  <shipment sequence="1" state="created" earliestShipDate="2014-02-11">
   <orderLines>
    <orderLine product="steepletoneroxy1pink" quantity="1"
      state="created">
    </orderLine>
   </orderLines>
  </shipment>
```

© Realtime Despatch Software Limited 2019

```
  </shipments>
</order>
```

Order Cancellation

The process of deleting or cancelling an order is usually driven by the third party application in which the order was first generated. The process should attempt to delete the order within the OrderFlow system before changing the status of the order within the third party system.

	Operation Summary
	Invocation From 3rd party system to OrderFlow
	Method HTTP POST
	URL /remoteorder/order/cancel.xml
Parameters	*externalReference*: the reference of the order to be cancelled
	cancelChangesExternalReference: whether cancellation should change the order reference

The OrderFlow system may refuse to cancel an order if has been packed or despatched, in which case an error message will be returned and cancellation process should fail in the third party application.

TEMPORARY VS PERMANENT CANCELLATION

One of the options on order cancellation is the use of the `cancelChangesExternalReference`. If set to false, the order reference will not be changed. Because OrderFlow does not allow duplicates of orders with the same reference it will not be possible to reimport the same order.

For some third party systems, a temporary cancellation of the order is required in order to support the reimport of a *modified* order. In this case, the old order needs to be cancelled in a way that will allow the same (but modified) order to be reimported into the system. In this case the `cancelChangesExternalReference` parameter is used with the value of true.

If not supplied, the implied value for the `cancelChangesExternalReference` is typically `false`, although this can be changed by configuration.

ERROR CASES

If an attempt is made to cancel an order, one of the following scenarios will apply:

- the cancellation will take place as expected. All shipments in the order that have not already been completed will be cancelled.
- the cancellation cannot take place because one of more of the shipments cannot be cancelled. For example, if a shipment has already been packed, it will need to be unpacked before it is eligible for cancellation.

© Realtime Despatch Software Limited 2019

The are situations when a cancellation cannot be attempted. If an attempt is made to cancel an order which is not on an authenticated channel, an `UnauthorizedDataAccessException` will be raised.

If the third party system attempts to cancel an order which has not yet been received by OrderFlow, then `MissingDataException` will be raised. The cancellation operation will return XML as in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
<message>
<![CDATA[Unable to find order for reference &apos;duff&apos;.]]>
</message>
<exception>rtd.exceptions.MissingDataException</exception>
<detail>
<![CDATA[... error detail below]]>
</detail>
</error>
```

When `MissingDataException` is raised, the most appropriate behaviour on the third party system is to allow the underlying cancellation to proceed, in contrast to other situations in which it would be more appropriate to block the cancellation.

Order Line Cancellation

As with orders, order line cancellation via the OrderFlow Remote API is typically initiated by the third party application from which the order line was received.

Once invoked, the process will attempt to delete the order line within OrderFlow. The shipment to which the order line belonged will be reset to the start of the despatch processing workflow. This is because paperwork, courier selections and other aspects of despatch processing will ususally need to change to reflect the modified shipment contents.

	Operation Summary
	Invocation From 3rd party system to OrderFlow
	Method HTTP POST
	URL /remoteorder/order/cancel.xml
Parameters	*orderReference*: the reference of the order to be cancelled.
	productReference: the product reference (SKU) within the order line to be cancelled.
	thirdPartyReference: any (optional) third-party reference associated with the order line to be cancelled. This is used to identify the correct order line if there are multiple order lines for the same product.

Note that the must always be at least one order line present in an order (the last line can not be cancelled). If all of the order lines in an order need to be cancelled, then the Order Cancellation API operation should be used instead.

© Realtime Despatch Software Limited 2019

If an attempt is made to cancel an order line, one of the following outcomes will apply:

- the cancellation will take place as expected. The order line will be removed from the order (and shipment).
- the cancellation cannot take place. This may occur for a number of reasons, as described in the next section.

ERROR CASES

OrderFlow may reject an order line cancellation attempt if the line has already been packed or despatched.

Order line cancellation is currently not supported for orders which contain more than one order lines with the same product reference.

If an attempt is made to cancel an order line which is not on an authenticated channel, an `UnauthorizedDataAccessException` will be raised.

In both cases, an error message will be returned and the cancellation process should fail in the third party application.

If the third party system attempts to cancel an order line which has not yet been received by OrderFlow (or is simply not present), then a `MissingDataException` will be raised.

The cancellation operation in the event of an error will return XML as in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
<message>
<![CDATA[Unable to find order line with product &apos;duff&apos;.]]>
</message>
<exception>rtd.exceptions.MissingDataException</exception>
<detail>
<![CDATA[... error detail below]]>
</detail>
</error>
```

When a `MissingDataException` is raised, the most appropriate behaviour for the third party system is to allow the underlying cancellation to proceed. This differs from other cases where blocking the cancellation would be more appropriate.

Pending Shipments

This API call can be used to get a list of pending shipments for a particular sales channel.

	Operation Summary
	Invocation From 3rd party system to OrderFlow
	Method HTTP GET
	URL /remoteorder/shipment/pending.xml
Parameters	*channel*: the channel under consideration

© Realtime Despatch Software Limited 2019

For each shipment listed, the created date, shipment reference, order reference, state and earliest ship date is shown.

```
<?xml version="1.0" encoding="utf-8" ?>
<shipments>
  <shipment>
    <created>2014-01-18 00:00:00</created>
    <reference>1001</reference>
    <orderReference>100</orderReference>
    <state>out_of_stock</state>
    <earliestShipDate>2014-01-18</earliestShipDate>
  </shipment>
  <shipment>
    <created>2014-01-17 00:00:00</created>
    <reference>1011</reference>
    <orderReference>101</orderReference>
    <state>move_pending</state>
    <earliestShipDate>2014-01-17</earliestShipDate>
  </shipment>
  ...
  <shipment>
    <created>2014-01-27 00:00:00</created>
    <reference>OSC200612300011</reference>
    <orderReference>OSC20061230001</orderReference>
    <state>out_of_stock</state>
    <earliestShipDate>2014-01-28</earliestShipDate>
  </shipment>
  <shipment>
    <created>2014-01-28 00:00:00</created>
    <reference>OSC200703190011</reference>
    <orderReference>OSC20070319001</orderReference>
    <state>move_pending</state>
    <earliestShipDate>2014-01-28</earliestShipDate>
  </shipment>
</shipments>
```

Note that usage of this API may be throttled for performance reasons, as in some environments a large volume of data may be returned with each call. Clients are expected to use the operation sensibly.

Despatched Shipments Per Time Period

This operation allows third party systems to query despatched shipments over a specified time period.

© Realtime Despatch Software Limited 2019

	Operation Summary
	Invocation From 3rd party system to OrderFlow
	Method HTTP POST
	URL /remoteorder/shipment/despatches.xml
Parameters	*channel*: the channel under consideration
	from: the date or time from which to consider
	to: the date or time to which to consider
	includeOrderLines: (optional, from 3.7.8) to include order lines in the shipment

Note that in the use of the from and to parameter, the date formats should be used as follows:

`yyyy-MM-ss HH:mm:ss` : use this to query within a specified time interval `yyyy-MM-ss` : use this to query within specified date intervals

Note that the `to` parameter value is exclusive. In order to get shipments from March 27 2014, use `2014-03-27` to `2014-03-28` . In order to get shipments from 3pm to 4pm on that day, you can use `2014-03-27 15:00:00` to `2014-03-27 16:00:00` .

Note that the `includeOrderLines` value can be either `true` or `false` , or omitted. If omitted, order lines will not be shown.

An example result set is shown below, which has been created with `includeOrderLines` set to true:

```
<?xml version="1.0" encoding="utf-8" ?>
<shipments>
    <shipment>
        <reference>831</reference>
        <orderReference>83</orderReference>
        <state>despatched</state>
        <carrier>royalmail_tracked</carrier>
        <service>TPN01</service>
        <despatchReference>TT222211109GB</despatchReference>
        <completed>2014-04-25 09:30:08</completed>
        <orderLines>
            <orderLine>
                <productReference>spa_sku1</productReference>
                <quantity>5</quantity>
                <thirdPartyReference></thirdPartyReference>
            </orderLine>
            <orderLine>
                <productReference>spa_sku2</productReference>
                <quantity>3</quantity>
                <thirdPartyReference></thirdPartyReference>
            </orderLine>
        </orderLines>
    </shipment>
```

© Realtime Despatch Software Limited 2019

```
    <shipment>
        <reference>851</reference>
        <orderReference>85</orderReference>
        <state>despatched</state>
        <carrier>royalmail_tracked</carrier>
        <service>TPN01</service>
        <despatchReference>TT222211090GB</despatchReference>
        <completed>2014-04-25 09:30:08</completed>
        <orderLines>
            <orderLine>
                <productReference>spa_sku1</productReference>
                <quantity>1</quantity>
                <thirdPartyReference></thirdPartyReference>
            </orderLine>
        </orderLines>
    </shipment>
    <shipment>
        <reference>1081</reference>
        <orderReference>108</orderReference>
        <state>despatched</state>
        <carrier>royalmail_tracked</carrier>
        <service>TPS01</service>
        <despatchReference>TT222211069GB</despatchReference>
        <completed>2014-04-25 09:30:08</completed>
        <orderLines>
            <orderLine>
                <productReference>spa_sku3</productReference>
                <quantity>2</quantity>
                <thirdPartyReference></thirdPartyReference>
            </orderLine>
        </orderLines>
    </shipment>
</shipments>
```

Note that the returned data includes the carrier and service code, as well as the despatch reference, for each outgoing shipment.

Operations FROM OrderFlow (PUSH)

As described earlier in the PUSH operations described here are from the 'generic' OrderFlow integration. All of the messages are XML-based.

In order to assist with sequencing, the document (top level) element in the XML document contains a `messageId` attribute. The value for this attribute is a sequence number. For successive messages that refer to the same entity, this sequence number can be used to determine the order in which these messages were generated.

In some cases, the operation is triggered by an event on OrderFlow. In these cases, some extra information on the context of the operation is passed through by adding an `event` element wrapper around the detail of the message. The event element passes through information such as the name of the event, the user, time, etc.

The `event` element wrapper takes the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="8"
  eventType="order_cancelled"
  userName="philz"
  eventTime="2015-01-21 10:29:51"
```

© Realtime Despatch Software Limited 2019

```
  entity="..."
  ... >
  <detail>
  ...
  </detail>
</event>
```

As with other push operations, the `event` element contains the ID for the OrderFlow message sent to the third party system.

We will see specific examples of messages that use the event wrapper in the sections below.

Product Operations

Inventory Push

The structure of the inventory push is the same as the inventory pull, it is used by OrderFlow to PUSH stock levels to Third Party applications when a change in stock levels triggers the event or as a scheduled background processes.

Operation Summary
Invocation Called from OrderFlow to client, periodic or event driven
Method HTTP POST
Example URL https://thirdpartyurl/productInventory.xml
Example input Body post as below

An example outgoing message is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<inventory messageId="15">
  <product
    inventoryId="453"
    sequenceId="453"
    externalReference="CD-SERGEANT"
    organisation="myco"
    site="global"
    total="186"
    allocated="0"
    available="186"
    frozen="0"
    onOrder="0"
    lastStockChangeId="86"
    lastLineRequirementChangeId="0" />
</inventory>
```

Note that each inventory push only includes products whose availability figure has modified since the last successful push operation. This is to allow for a more efficient inventory update notification process.

© Realtime Despatch Software Limited 2019

Note also that inventory notification is configured on a per organisation/channel basis. This configuration will include URLs, frequency, etc. The inventory notification can be periodic, or realtime. In the case of the latter, inventory notifications are triggered each time a change is made to the available quantity for a product.

The `sequenceId` value can be used to ensure that the latest inventory record is used for a product if multiple inventory messages are received out of sequence. For most products, the `sequenceId` will be the same as the `inventoryId`. However, if the product does not have an inventory record on the system, then the `inventoryId` will be zero, and the `sequenceId` will be set to the highest inventory ID value on the system at the time that the message was generated.

Stock Change Push

OrderFlow also supports a mechanism to notify third party systems of individual stock changes. This may be useful for third party systems that need an accurate audit of individual stock changes rather than simply the point in time inventory levels provided by the Product Inventory operation.

Examples of where an individual stock change feed may be required include:

- a third party system needs to maintain a record of stock adjustments and writeoffs for auditing purposes.
- OrderFlow needs to be integrated closely with another third party Warehouse Management System.

Stock Change Push notifications are tied to stock change events as they occur in OrderFlow, but can also be sent at regular intervals using a periodic report. A stock change event may generate one or more stock changes. For example, a move from one location to another will generate two stock changes: a `move_out` of one location, and a `move_in` to another.

Operation Summary
Invocation Called from OrderFlow to client on an event, or periodically
Method HTTP POST
Example URL https://thirdpartyurl/stockChanges.xml
Example input Body post as below

An example Stock Change Push entry is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<stockChanges
  messageId="14"
  organisation="myco"
  site="DEFAULT">
  <stockChange
    id="89"
    type="negative_adjustment"
    timestamp="2015-01-21 16:49:22"
    product="CD-SERGEANT"
    location="location_4"
    locationType="pickable"
    previousQuantity="187"
    changeQuantity="-1"
```

© Realtime Despatch Software Limited 2019

```
    newQuantity="186"
    user="philz">
    <note><![CDATA[Item dropped.]]></note>
  </stockChange>
</stockChanges>
```

For both the event and periodic report based approach, the system can be configured to only send particular types of stock changes. For example, internal stock moves within the system are generally interesting to a third party system. However, movements into locations for *damaged* items are more likely to be of interest.

© Realtime Despatch Software Limited 2019

A list of the most common types of stock changes is shown below:

Stock Change Types	Summary
Delivery	The receipt of an inbound item via an incoming Delivery to OrderFlow
Move in	The movement of an item into a location. Excludes *damaged* locations.
Move out	The movement of an item out of a location. Excludes *damaged* locations.
Damaged in	The movement of an item into a *damaged* location.
Damaged out	The movement of an item out of a *damaged* location. Useful if item had been incorrectly recorded as damaged.
Negative adjustment	Used if stock is not present in the current location as previously recorded.
Positive adjustment	Used if the stock is present on a location where not previously recorded.
Pack debit	debit Records a debit (reduction) in the stock holding of an item following the pack of a shipment.
Unpack credit	If a shipment is unpacked, handles the credit of stock back onto the system.
Return	Covers the receipt of incoming stock through the returns process.

Delivery and Purchase Order Operations

It is possible to configure OrderFlow to push back event notifications relating to incoming deliveries and purchase orders.

Each time a delivery line is recorded (when a given quantity of a particular product is added to the system), an event can be generated. Another point at which events can be triggered is when a delivery is completed by moving into the `applied` state. Finally, a purchase order (against which multiple deliveries can be recorded) can trigger an event notification when it is marked as completed; this will take place when no more deliveries are expected to be recorded against the purchase order.

One or more of these event types can be used by a third party application. Each of these is discussed in turn.

Delivery Line Push

The Delivery Line Push event is recorded when a new delivery line is added to the system.

Note that the event is generated only if a stock change is recorded against the delivery line; it is possible, for example, to record all the lines of the delivery and only apply the stock changes at the end in a single operation. In the latter case, no event will be generated at this stage.

© Realtime Despatch Software Limited 2019

Operation Summary
Invocation Called from OrderFlow to client on an event, or periodically
Method HTTP POST
Example URL https://thirdpartyurl/deliveryLine.xml
Example input Body post as below

An example delivery line is shown below. Note that as well as containing details on the product and quantity received, it includes the detail of the containing delivery. If the delivery was part of a purchase order, then details of the purchase order are included as well.

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="4"
  eventType="delivery_line_applied"
  userName="philz"
  eventTime="2015-01-21 16:35:17"
  entity="rtd.domain.DeliveryLine">
  <detail>
    <deliveryLine
      id="24"
      product="ipod5"
      variation="standard"
      quantity="1"
      state="applied">
      <delivery
        id="33"
        type="from_licence_plates"
        state="receiving"
        site="SECOND"
        organisation="acme"
        supplierReference="default"
        supplierDeliveryReference="acmedel_1"
        deliveryDate="2013-07-18"
        created="2013-07-18 15:11:32">
        <purchaseOrder
          id="6"
          externalReference="po_acme_1"
          state="created"
          supplierPurchaseOrderReference="po_acme_1"
          manuallyCompleted="false"
          purchaseOrderDate="2013-09-05">
        </purchaseOrder>
      </delivery>
    </deliveryLine>
  </detail>
</event>
```

Note that as with the Order Event Push operations, the delivery uses the `event` element wrapper.

© Realtime Despatch Software Limited 2019

Delivery Push

At the point when a delivery is completed or *applied*, an event message can be triggered.

The Delivery Push message contains details on all of the delivery lines included within the delivery. This makes it possible to identify any delivery line messages that have not been received, or indeed, to rely on the Delivery Push message for a complete record of the deliveries applied.

Operation Summary
Invocation Called from OrderFlow to client on an event, or periodically
Method HTTP POST
Example URL https://thirdpartyurl/delivery.xml
Example input Body post as below

An example Delivery Push message is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="6"
  eventType="delivery_applied"
  userName="philz"
  eventTime="2015-01-21 16:40:08"
  entity="rtd.domain.Delivery">
  <detail>
    <delivery
      id="33"
      type="from_licence_plates"
      state="applied"
      site="SECOND"
      organisation="acme"
      supplierReference="default"
      supplierDeliveryReference="acmedel_1"
      deliveryDate="2013-07-18"
      created="2013-07-18 15:11:32"
      completed="2015-01-21 16:40:08">
      <purchaseOrder
        id="6"
        externalReference="po_acme_1"
        state="partially_applied"
        supplierPurchaseOrderReference="po_acme_1"
        manuallyCompleted="false"
        purchaseOrderDate="2013-09-05">
      </purchaseOrder>
      <deliveryLines>
        <deliveryLine
          id="23"
          product="ipod5"
          variation="stock_only"
          quantity="1"
          state="applied" />
        <deliveryLine
```

© Realtime Despatch Software Limited 2019

```
          id="24"
          product="ipod5"
          variation="stock_only"
          quantity="1"
          state="applied" />
      </deliveryLines>
    </delivery>
  </detail>
</event>
```

The detail on the delivery includes a reference to the supplier, the supplier's reference for the delivery, as well as the date on which the delivery was expected, received and completed.

Purchase Order Push

A Purchase Order Push event can be triggered when a purchase order is completed on the system. When a delivery is completed on OrderFlow, the purchase order will automatically be completed if there are no further outstanding items on the purchase order. In this case, there is no need for a separate purchase order notification.

However, if the purchase does still contain outstanding items, it is possible to complete the purchase order *manually* if no further deliveries are expected against the purchase order, and in turn, generate a Purchase Order push notification.

Operation Summary
Invocation Called from OrderFlow to client on an event, or periodically
Method HTTP POST
Example URL https://thirdpartyurl/purchaseOrder.xml
Example input Body post as below

An example purchase order notification is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="7"
  eventType="purchase_order_completed_manually"
  userName="philz"
  eventTime="2015-01-21 16:41:09"
  entity="rtd.domain.PurchaseOrder">
  <detail>
    <purchaseOrder
      messageId="7"
      id="6"
      externalReference="po_acme_1"
      state="completed"
      organisation="acme"
      site="SECOND"
      supplierReference="HAMA"
      supplierPurchaseOrderReference="po_acme_1"
      manuallyCompleted="true"
      purchaseOrderDate="2013-09-05">
```

© Realtime Despatch Software Limited 2019

```
      <note><![CDATA[No more items to receive.]]></note>
      <purchaseOrderLines>
        <purchaseOrderLine
          id="9"
          product="ipod5"
          quantity="10"
          outstanding="8"
          externalReference="acme1_1"
          state="created">
        </purchaseOrderLine>
        <purchaseOrderLine
          id="10"
          product="woodworm_zoom"
          quantity="8"
          outstanding="8"
          externalReference="acme1_2"
          state="created">
        </purchaseOrderLine>
        <purchaseOrderLine
          id="11"
          product="cyclepro_cape"
          quantity="20"
          outstanding="20"
          externalReference="acme1_3"
          state="created">
        </purchaseOrderLine>
      </purchaseOrderLines>
      <deliveries>
        <delivery
          id="33"
          type="from_licence_plates"
          state="applied"
          supplierDeliveryReference="acmedel_1"
          deliveryDate="2013-07-18"
          created="2013-07-18 15:11:32"
          completed="2015-01-21 16:40:08">
          <deliveryLines>
            <deliveryLine
              id="23"
              product="ipod5"
              variation="stock_only"
              quantity="1"
              state="applied" />
            <deliveryLine
              id="24"
              product="ipod5"
              variation="stock_only"
              quantity="1"
              state="applied" />
          </deliveryLines>
        </delivery>
      </deliveries>
    </purchaseOrder>
  </detail>
</event>
```

The Purchase Order Push notification contains details of all the lines in the purchase order, together with the quantity outstanding for each line.

In addition, the notification contains details on all of the deliveries applied against the purchase order, as well as lines contained within these deliveries. This makes it possible to verify that all of the deliveries expected against the purchase order have been received and processed.

© Realtime Despatch Software Limited 2019

Order Operations

Event Push

OrderFlow can be configured to push notification of changes made to an order to the associated third party applicaiton. The main changes of interest are usually changes to the status of an order or its associated shipments, for example, which take place for example when shipments are marked as packed or despatched.

Each of these different status types may also return additional information. Generic information, such as the time and user name of the event instigator, will always be available. In other cases, relevant additional information may be available.

Order events can be triggered at the order, shipment and/or line item level. The main ways in which a status notification will be triggered as as follows:

- when a state change occurs, for example, when a shipment's status changes from picked to packed.
- when a non-state changing operation is executed. For example, when a shipment is printed, no change of the shipment takes place.

However, it is still possible for a notification to be sent to the third party system.

Order Event Pushes can be triggered from events that take place on an order, the shipments used to despatch lines within the order, or on the lines themselves.

Exactly which combination of order, shipment and line item events and operations result in event notifications can be controlled through the OrderFlow configuration.

Operation Summary
Invocation Called from OrderFlow to client, typically event driven
Method HTTP POST
Example URL https://thirdpartyurl/orderEvent.xml

An example order state notification is shown below, in this case for the despatch of a shipment.

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="4"
  eventType="shipment_despatched"
  userName="philz"
  eventTime="2015-01-22 11:34:08"
  entity="rtd.domain.Shipment"
  externalReference="141"
  operation="despatch"
  state="despatched">
  <detail>
    <order
      externalReference="141"
      state="despatched">
```

© Realtime Despatch Software Limited 2019

```
      <shipments>
        <shipment
          sequence="1"
          state="despatched"
          externalReference="1411"
          earliestShipDate="2009-12-07"
          courier="royalmail_dmo"
          despatchReference="12345678">
          <orderLines>
            <orderLine
              product="DVD-REDC"
              quantity="1"
              state="packed">
            </orderLine>
          </orderLines>
        </shipment>
      </shipments>
    </order>
  </detail>
</event>
```

An example for an order cancellation is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="8"
  eventType="order_cancelled"
  userName="philz"
  eventTime="2015-01-21 10:29:51"
  entity="rtd.domain.OrderItem"
  externalReference="SPA_MULTI_4"
  state="cancelled"
  operation="cancel">
  <detail>
    <order
      externalReference="SPA_MULTI_4"
      state="cancelled">
      <shipments>
        <shipment
          sequence="1"
          state="cancelled"
          externalReference="SPA_MULTI_4"
          earliestShipDate="2014-11-05"
          courier="generic">
          <orderLines>
            <orderLine
              product="spa_sku2"
              quantity="3"
              state="cancelled">
            </orderLine>
            <orderLine
              product="spa_sku3"
              quantity="1"
              state="cancelled">
            </orderLine>
            <orderLine
              product="spa_sku4"
              quantity="2"
              state="cancelled">
            </orderLine>
          </orderLines>
        </shipment>
      </shipments>
    </order>
```

© Realtime Despatch Software Limited 2019

```
  </detail>
</event>
```

Note that in both cases, the operation has been triggered by event on the system. This is clear from the `event` element that wraps the detail of the message.

Shipment Operations

Payment Request

OrderFlow is able to send requests to third party applications to request that payment to be taken for a shipment. This operation only applies for configurations for which up front payment does not occur, In these cases, payment is only taken when the lines for a particular shipment are verified to be in stock. The Payment Request operation is typically invoked within OrderFlow prior to picking to limit the operational consequences of a failed payment request.

Operation Summary
Invocation PUSH from OrderFlow to shopping cart at the point a shipment can be processed
Method HTTP POST
Example URL https://thirdpartyurl/requestPayment.xml

The data provided as part of the request includes the shipment and its containing order reference, as well as the line items associated with the payment request.

Example body text for the Payment Request invocation is shown below:

```
<paymentRequest date="2014-05-10 19:10:21">
  <order externalReference="multishipment">
    <shipments>
      <shipment externalReference="multishipment2">
        <orderLines>
          <orderLine product="DVD-BELOVED" quantity="3">
          </orderLine>
          <orderLine product="DVD-MATR" quantity="4">
          </orderLine>
        </orderLines>
      </shipment>
    </shipments>
  </order>
</paymentRequest>
```

Invocation of this operation by OrderFlow should trigger a corresponding Payment Response operation described below. However, these two operations are not tied to each other in a synchronous fashion.

Following receipt of the Payment Request, the third party application should return a success response to OrderFlow, at which point, OrderFlow will change the status of the relevant shipment to `awating_payment_confirmation`. Subsequently, the third

© Realtime Despatch Software Limited 2019

party application will trigger payment to be taken via an interaction with the relevant payment gateway, after which the Payment Response will be used to communicate the result back to OrderFlow.

Payment Response

This service should be used by a third party applications to report the payment status of a shipment following earlier receipt of a Payment Request call from OrderFlow. The call to this method is invoked at some point after the third party application has had a chance to process the shipment referred to in the Payment Request call.

Operation Summary
Invocation Invoked by the 3rd party application responsible for interfaces with the payment gateway
Method HTTP POST
Example URL https://thirdpartyurl/shipmentresponse.xml

For successes it should simply replay the contents of the payment request, but indicate using a result attribute that the payment confirmation is successful.

For a **success** operation, the body of the Payment Response invocation would be as below.

```
<paymentResponse result="success">
  <order …>
    … contents as per Payment Request
  </order>
</paymentResponse>
```

Note the use of the `success` flag to indicate successful completion of the operation.

For failed payments, the body would be as shown below.

```
<paymentResponse result="failed">
  <message>Message from payment gateway</message>
  <code>XYZ</code>
  <order …>
    … contents as per Payment Request
  </order>
</paymentResponse>
```

Note that for both successful and failed payments the order line contents of the shipment is replayed back to OrderFlow. In the case of payment failure, a message and optional response code is passed back to OrderFlow. In this case, OrderFlow will set the state of the shipment to `payment_request_failed`. Only if payment is successful will the shipment be put back into the despatch workflow.

© Realtime Despatch Software Limited 2019

Return Operations

Return Push

A Return Push event can be triggered when a return is applied on the system.

Operation Summary
Invocation Called from OrderFlow to client on an event, or periodically
Method HTTP POST
Example URL https://thirdpartyurl/return.xml
Example input Body post as below

An example return notification is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<event
  messageId="243"
  eventType="return_item_applied"
  userName="philz"
  eventTime="2018-01-21 14:46:45"
  entity="rtd.domain.Return">
  <detail>
    <return
      id="10035468782"
      authorised="true"
      type="stock"
      site="DEFAULT"
      authorisation="10035468782"
      orderReference="10025908733"
    >
      <returnLine
        product="DVD-REDC"
        quantity="1"
        reason="D - Wrong product sent"
        condition="As new"
        refund="true"/>
      <returnLine
        product="DVD-BELOVED"
        quantity="2"
        reason="E - Quality/Manufacturing fault"
        condition="Product damaged (irreparable)"
        refund="true"/>
    </return>
  </detail>
</event>
```

The Return Push notification contains details of all the lines in the return.

© Realtime Despatch Software Limited 2019

Appendix

Import Field Definition Detail

The next section provides detail on the field definitions used in the product, order and other import operations.

Product Import Fields

More detail on the fields which can be imported using the Product Import and Product Update are shown below.

© Realtime Despatch Software Limited 2019

PRODUCT FIELDS

© Realtime Despatch Software Limited 2019

Name	Mandatory	Format	Example
externalReference	YES	VARCHAR(120)	`DVD-UNSG`
description	YES	VARCHAR(120)	`Under Siege`
type	YES	VARCHAR(120) The 'logical' type of the product. Products of different types will be used by the system in different ways.	Implementation-dependent
quantityType	NO	Denotes what the quantity means for the product. Defaults to 'unit' if omitted.	`unit`, `weight`, `volume`
displayUnits	NO	The units to be displayed on the user interfaces for this product, if quantity type is not 'unit'. Will be abbreviated where appropriate.	`kilogram`, `gram`, `milligram`, `litre`, `millilitre`
barcode	NO	VARCHAR(120)	`112112113455`
thirdPartyReference	NO	VARCHAR(100)	`654913223164`
imageReference	NO	VARCHAR(120) Used to provide link to an external system holding images of the product. Can be a relative path.	`images/dvd-unsg.gif`
packagingDescription	NO	VARCHAR(255) Text describing how item is packaged and quantity thereof.	`Cartons of 12`

© Realtime Despatch Software Limited 2019

PRODUCT ATTRIBUTE FIELDS

OrderFlow supports the import of arbitrary product attributes as name value pairs.

Name	Mandatory	Format	Example
name	NO	VARCHAR(120)	`card.tx.ref`
title	YES	VARCHAR(120)	`Card Transaction Reference`
value	NO, but normally supplied	VARCHAR(5000)	`XYZ-123`

Order Import Fields

A more complete list of available order, shipment and order line fields used the Order Import operation are shown below:

© Realtime Despatch Software Limited 2019

ORDER FIELDS

© Realtime Despatch Software Limited 2019

Name	Mandatory	Format	Example
externalReference	YES	VARCHAR(80)	`ORDREF_123000034`
state	Project specific	One of a predefined list of values which may be used to control how an order is processed. Normally, it is best not to set this value, instead leaving this job to the OrderFlow import mapping script	`created`
partialOrder	NO	BOOLEAN Flag indicating whether order has been divided upstream of OrderFlow.	`false`
paymentGatewayIdentifier	NO	VARCHAR(120) Used to pass a payment gateway identifier to OrderFlow, if required	`google_checkout`
paymentTransactionInfo	NO	VARCHAR(1024)	Environment dependent
placed	NO	TIMESTAMP Time the order was placed by the customer	`2014-11-31 13:24:06`
authorised	NO	TIMESTAMP Time the payment was approved (typically by a payment gateway)	`2014-11-31 15:24:09`
customerComment	NO	VARCHAR(1024)	`As discussed, please fix UK plug`
totalPriceNet	NO	FLOAT	`10.20`
totalPriceGross	NO	FLOAT	`11.99`
totalPriceTax	NO	FLOAT	`1.79`
totalTaxCode	NO	One of a list of values which may be used to control the information shown on customer paperwork	`T0`, `T1`, `T2`
shippingPriceNet	NO	FLOAT The net amount for the shipping charge for the order	`2.98`
shippingPriceGross	NO	FLOAT	`3.50`
shippingTaxTotal	NO	FLOAT	`0.52`
shippingTaxCode	NO	One of a list of values which may be used to	`T0`, `T1`, `T2`

© Realtime Despatch Software Limited 2019

ORDER ATTRIBUTE FIELDS

OrderFlow supports the import of arbitrary order attributes as name value pairs.

Name	Mandatory	Format	Example
name	NO	VARCHAR(120)	`card.tx.ref`
title	YES	VARCHAR(120)	`Card Transaction Reference`
value	NO, but normally supplied	VARCHAR(5000)	`XYZ-123`

© Realtime Despatch Software Limited 2019

SHIPMENT FIELDS

© Realtime Despatch Software Limited 2019

Name	Mandatory	Format	Example
externalReference	NO	VARCHAR(100) Can be omitted, in which case, order reference will typically be used to provide implicit value.	`ORDREF_123000034`
paidFor	NO	BOOLEAN Flag indicating whether shipment payment has been taken.	`true`
earliestShipDate	NO	DATETIME The earliest date the shipment should be shipped	`2014-11-31`
site	NO	VARCHAR(120) Optional field used to explicitly select site or warehouseto be used for fulfilling shipment. Only used for multi-site environments.	`WAREHOUSE_1`
priority	NO	INTEGER A positive number representing the priority of the shipment	`100`
priorityName	NO	VARCHAR(30) A human-readable string representing the priority	`Urgent`
weight	NO	FLOAT	`100`
weightUnits	NO	VARCHAR(10) The weight unit, which defaults to 'gram'	`gram`
addressLine1	NO	VARCHAR(255)	
addressLine2	NO	VARCHAR(255)	
addressLine3	NO	VARCHAR(255)	
addressLine4	NO	VARCHAR(255)	
addressLine5	NO	VARCHAR(255)	
addressLine6	NO	VARCHAR(255)	
countryCode	NO	VARCHAR(2)	
postCode	NO	VARCHAR(10)	
contactName	NO	VARCHAR(255)	

© Realtime Despatch Software Limited 2019

Note that if the shipment address fields or contact fields are set, they will override any values set for the delivery address fields in the containing order. This allows multiple shipments within the same order to be directed to different addresses.

© Realtime Despatch Software Limited 2019

ORDER LINE FIELDS

© Realtime Despatch Software Limited 2019

Name	Mandatory	Format	Example
product.externalReference	YES	VARCHAR(120)	`DVD-MATR`
quantity	YES	INTEGER	`2`
description	NO	VARCHAR(1024) Typically used for a local language order line description. If not set, then product description is used instead	
totalPriceNet	NO	FLOAT	`10.20`
totalPriceGross	NO	FLOAT	`11.99`
totalPriceTax	NO	FLOAT	`1.79`
totalTaxCode	NO	VARCHAR(10)	`T0`
unitPriceNet	NO	FLOAT	`10.20`
unitPriceGross	NO	FLOAT	`11.99`
unitPriceTax	NO	FLOAT	`1.79`
unitTaxCode	NO	VARCHAR(10)	`T0`
promotionCode	NO	VARCHAR(80) The promotion code for the order if present	`summer_2014`
promotionPriceDescription	NO	VARCHAR(150) The description of the promotion price	`£9.99 (was £12.99)`
userDefined1	Project specific	VARCHAR(255)	
userDefined2	Project specific	VARCHAR(255)	
userDefined3	Project specific	VARCHAR(255)	
userDefined4	Project specific	VARCHAR(255)	

© Realtime Despatch Software Limited 2019

Note that either the total price or the unit price should be specified. OrderFlow will not derive order line prices if none are supplied, but it can, for example, derive a total price value from unit prices.

Prices are typically required for despatch note paperwork, and for international shipment, for customs documentation.

HTTP Example

An example HTTP request response pair shows how HTTP is used in the Product Import operation, described in in the next section.

Note that in this example, the XML is passed in the body of the HTTP POST. In other POST operations, such as the Order Cancellation operation, the body will consist of request parameters as URL encoded name value pairs.

Request

```
POST /rtd2-host/remoteorder/imports/importitems.xml HTTP/1.1
channel: MYCHANNEL
user: philz
password: cGhpbHo=
User-Agent: Jakarta Commons-HttpClient/3.1
Host: 127.0.0.1:8080
Content-Length: 967

<?xml version="1.0" encoding="UTF8"?>
<imports>
  <import type="product" operation="insert" externalReference="TEST_fullproduct">
    externalReference=TEST_fullproduct
    description=A description for the test product
    weight=100
    weightUnits=grams
    type=default
    quantityOnOrder=10
    imageReference=TEST_fullproduct.gif
    priceNet=10.50
    priceGross=11.50
    tax=1.50
    taxCode=T1
    currency=GBP
    currencyUnits=pounds
    userDefined1=User defined field value 1
    userDefined2=
    userDefined3=
    userDefined4=
    userDefined5=
    channel=MYCHANNEL
    type=default
    activated=true
  </import>
  <import type="product" operation="insert" externalReference="TEST_min_product">
    description=A description for the min product
    organisation=altco
    type=default
    activated=true
  </import>
  <import type="product" operation="insert" externalReference="TEST_global_product">
    externalReference=TEST_global_product
    description=A description for the global product
    type=default
```

© Realtime Despatch Software Limited 2019

```
    activated=true
  </import>
</imports>
```

Note how the channel external reference is passed in using the channel header.

In the example above the user name and password (in this case `philz`) are passed as headers, with the password being Base64 encoded. The other authentication method uses HTTP Basic Authentication (as described on Wikipedia). In this case, the user name and password headers would be replaced by the following:

```
Authorization: Basic cGhpbHo6cGhpbHo=
```

where `cGhpbHo6cGhpbHo=` is the base Base64 encoding of `philz:philz`.

Below we show both the normal and the error response to a this request.

Response

NORMAL RESPONSE

```
HTTP/1.1 200 OK
Content-Language: en-US
Content-Type: text/xml; charset=utf-8
Transfer-Encoding: chunked
Server: Jetty(6.1.14)

<?xml version="1.0" encoding="UTF-8"?>
<importResult>
  <importSuccesses>
    <import type="product" operation="insert" externalReference="TEST_fullproduct"
      entity="rtd.domain.database.Product" item="TEST_fullproduct"
      queryTime="2014-09-28 20:32:34.620" />
    <import type="product" operation="insert" externalReference="TEST_min_product"
      entity="rtd.domain.database.Product" item="TEST_min_product"
      queryTime="2014-09-28 20:32:34.636" />
    <import type="product" operation="insert" externalReference="TEST_global_product"
      entity="rtd.domain.database.Product" item="TEST_global_product"
      queryTime="2014-09-28 20:32:34.658" />
  </importSuccesses>
  <importFailures></importFailures>
</importResult>
```

Notice how the result is returned in the body of the HTTP request, normally in XML format.

ERROR RESPONSE

```
HTTP/1.1 500 Internal Server Error
Content-Language: en-US
Content-Type: text/xml; charset=utf-8
Transfer-Encoding: chunked
Server: Jetty(6.1.14)

<?xml version="1.0" encoding="UTF-8"?>
<error>
  <message>
    <![CDATA[Cannot authenticate &apos;MYCHANNEL&apos;.
```

© Realtime Despatch Software Limited 2019

```
        Does not match list of authenticated channel]]></message>
  <detail>
       <![CDATA[Cannot authenticate &apos;MYCHANNEL&apos;.
        Does not match list of authenticated channel(rtd ...]]>
  </detail>
</message>
</error>
```

For most of the operations, an error which prevents successful completion of the operation results in a HTTP response with a non-200 response code. In this case, more details can be extracted from the message and detail elements.

ERROR RESPONSES

When an error occurs in one of the XML operations, OrderFlow will respond in a standard way. Typically, it will return a non 200 HTTP return code. For example, if the user is not authenticated then a 401 return code will be returned. If the server is unable to complete the request due to some other application error, a 500 return code will be returned.

OrderFlow will return error text in the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
  <message>high level error message</message>
  <detail>stack trace of error message</detail>
</error>
```

Error text is generated dynamically, with the intention being to return meaningful text that will help diagnose and correct problems. This approach means that it is not possible to provide an exhaustive list of errors that may be returned by the API.

Examples of the most common errors found in the element of are API response are given below.

© Realtime Despatch Software Limited 2019

HTTP Code	Error Text	Explanation
HTTP 500 Internal Server Error	Unable to load XML document from resource	The message structure or XML content cannot be parsed, check the XML formatting.
HTTP 500 Internal Server Error	Expecting value for 'channel' request header or parameter	The channel value was not found in the HTTP request header
HTTP 500 Internal Server Error	Cannot authenticate 'channelexample' Does not match list of authenticated channel	The channel contained in the HTTP request header is not valid
HTTP 412 Precondition failed	No user for supplied user name and password combination	Username / password combination is invalid
HTTP 200	Duplicate entity in import of order with reference: 03340002 Rejected instance rtd.domain.database.OrderItem with reference '03340002'.	The order '03340002' already exists for the channel
HTTP 200	No product found for external reference: WRONGPRODUCT. If this product is present, make sure that it has been activated.	A product code contained within the order is unrecognized
HTTP 200	Failed to convert property value of type 'java.lang.String' to required type 'java.lang.Double' for property 'totalPriceGross' nested exception is java.lang.NumberFormatException For input string: '19t6.99'	The value 'totalPriceGross' could not be stored as a numeric value
HTTP 200	Unable to find any entity associated with identifier:ref:courier:worldship	The courier identifier “worldship” does not exist or is inactive

© Realtime Despatch Software Limited 2019