Web API Endpoint Reference

Version 2023.001

Contents

Web API Endpoint Reference

Our Web API endpoints give external applications access to Spring Systems transactional data.

Usage of web API's is much preferred over file based integration.

While most transactions are supported in both directions, we find it most efficient for the *transaction originator* to initiate the call. E.g. while we support an external party calling into our system to 'Get Ship Request (940) from Spring', we have found it faster and more efficient for our system to instead push the Ship Request (940) out to the receiving system (e.g. your WMS). Our system is the originator of the Ship Request (940) transaction so it is faster if we immediately push it out to the WMS (warehouse management system). Similarly, we prefer for the WMS to push back to us the completed shipment.

Web API Base URL:

PRODUCTION https://portalapp.springsystems.com/api/

TEST https://portalapp-staging.springsystems.com/api/

OPERATION	DESCRIPTION	ENDPOINT
Collect Purchase Order from Spring	Returns order data based on the values that you specify.	/po-outgoing/export
Send Shipment to Spring	Creates a Shipment within the Spring Systems Portal.	/api/ship-incoming/send
Send Invoice to Spring	TBD	TBD

Authentication:

Method 1:

https://***api_user***@***api_key***:***<insert endpoint here>***

example: https://***api_user***@***api_key***:portalapp.springsystems.com/api/po-outgoing/export

Method 2:

https:// ***<insert endpoint here>***/api_user/***usernamegoeshere***/api_key/***keygoeshere***

example: https://portalapp.springsystems.com/api/po-outgoing/export/api_user /***usernamegoeshere***/api_key/***keygoeshere***

ID fields:

Throughout our data, you will see ID and KEY fields (xxxxx_ID -and- xxxx_KEY). These are internally assigned identification numbers given by our system to a particular object (e.g. a vendor, a location, a purchase order, etc). These are Spring Systems internally assigned values which you would receive from our system, e.g. on a Purchase Order. You can use these ID' or KEY's to very distinctly identify an object when communicating with our system. If you don't know the ID or KEY of the object, you can identify it back to us using a combination of other fields.

(NOTE: xxx_id are numeric (auto-incremented), xxx_key are auto-generated strings)

Response Codes

It is the sender's responsibility to verify that each API call receives a successful 200 level response. Below is an excerpt from https://developer.mozilla.org and https://restfulapi.net

Web API's typically use the Status-Line section of an HTTP response message to inform clients of their request's overarching result.

HTTP defines these standard status codes that can be used to convey the results of a client's request. The status codes are divided into the five categories.

- **1xx: Informational** – Communicates transfer protocol-level information.
- **2xx: Success** – Indicates that the client's request was accepted successfully.
- **3xx: Redirection** – Indicates that the client must take some additional action in order to complete their request.
- **4xx: Client Error** – This category of error status codes points the finger at clients.
- **5xx: Server Error** – The server takes responsibility for these error status codes.

Spring systems follows this approach. As with all API communications, **it is the responsibility of the sending system to ensure that every call receives a successful 200 level response**. Spring Systems does not log non successful transactions.

Examples of successful transactions:

Status: 200 OK

```
<messages>Invoices(s) successfully saved</messages>
<success>1</success>
<transaction>
  <transaction_key>810609189d9f1993</transaction_key>
  <transaction_created>2021-05-04 13:52:27</transaction_created>
  <transaction_additional>
    <attributes/>
  </transaction_additional>
</transaction>
```

Examples of failed transactions:

Status: 401 Unauthorized (401)

```
{"errors":["Invalid API credentials. Invalid data passed"]}
```

Status: 400 Bad Request (400)

```
<?xml version="1.0" encoding="UTF-8"?>
<pos>
  <errors>Vendor is a required field and is empty or null</errors>
</pos>
```

Status: 400 Bad Request (400)

```
<?xml version="1.0" encoding="UTF-8"?>
<shipments>
  <errors>Your request is missing filter data.  Please send filter parameters with your request</errors>
</shipments>
```

Request Parameters

When making a web service call, please pass along any variables or settings within the call. Some variables are mandatory such as api_user or api_key, or Request Parameters Filters such as show below in the Get Purchase Order from Spring example. Additionally some settings or limits can be overridden via an API call as well. Please try overriding such settings by simply passing the setting within the call for example setting.name/value (e.g. pagination.limit/x) within the call to override.

Throttling

Overview

Spring Systems PortalApp uses an API throttling to ensure availability and consistent throughput. Please make sure you are transmitting more than one transaction per API call to avoid any throttling issues.

- **Request Limit** - This is the number of API calls allowed per *Request Time Period*. Default setting is 500 requests (calls) per 86400 seconds (24h)
- **Request Time Period** - Default setting is 86400 seconds (24 hours)
- **Transaction Limit** - Default limit is 5,000 created or updated transactions per period.
- **Requested Entities Limit** - This applies to get requests only and controls the number of transactions or item returned per *Request Time Period*. For example, when requesting orders or items, please use filters to limit the number of possible results in your API query. Default is 50,000. (also see pagination)
- **Burst Time Period** - Default is 1 call per 1 sec

In addition to the traditional limits detailed above, Spring also utilizes a "Leaky Bucket" approach to further allow burst calls, yet control the pace of these bursts.

- **Leaky Bucket Capacity** – Each api_user has a "bucket capacity" per endpoint. The default capacity is 100. The bucket starts with 0 calls. Each call adds one to the bucket. When the bucket is full, no further calls will be allowed.
- **Leak per second** – each bucket will remove one per second to rebuild the allowed call count.

If these limits are reached, your API user will be prevented from making additional calls until the subsequent time period and/or appropriate limit is reached. Once again, please make sure you are transmitting ***more than one transaction per API call*** to avoid any throttling and transmission limit issues.

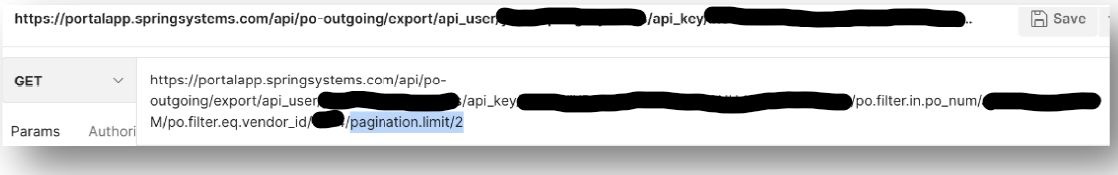
Please try to schedule and design your API calls to stay within these limits. Contact your account manager if you find the need to extend these limits (additional charges may apply). Limits cannot be extended when API calls are being made with only one transaction per call.

Pagination

Spring Systems uses the API cursor approach for pagination when making get requests. Your initial API response will include the *total records* to be returned as well as the default *limit*. (You may override this limit up to the *max limit* which is also provided).

When the total results exceed the *limit*, you will also be presented with the *next cursor* so that your system may create the appropriate URL to obtain the next page of results. You will also be presented with a pre-built URL if you find that easier.

You may override the per page quantity up to the max value. To do so, simply pass pagination.limit/x within the call to override.



Response

Throttling and Pagination detail is provided within the response header. Pagination detail is provided in both the response body and header.

Body Response Example: No Pagination:

```
<response_metadata>
    <pagination>
        <total_records>1</total_records>
        <total_pages>1</total_pages>
        <limit>25</limit>
        <max_limit>100</max_limit>
        <next_cursor/>
    </pagination>
</response_metadata>
```

Body Response Example: Yes Pagination:

```
<response_metadata>
    <pagination>
        <total_records>29</total_records>
        <total_pages>2</total_pages>
        <limit>25</limit>
        <max_limit>100</max_limit>
        <next_cursor>eyJwb19pZCI6MjczODE3fQ</next_cursor>
        <next_page>http://portalapp-staging.springsystems.com/api/po-
        outgoing/export/po.filter.gt.po_id/273788/pagination.next_cursor/eyJwb19pZ
        CI6MjczODE3fQ</next_page>
    </pagination>
</response_metadata>
```

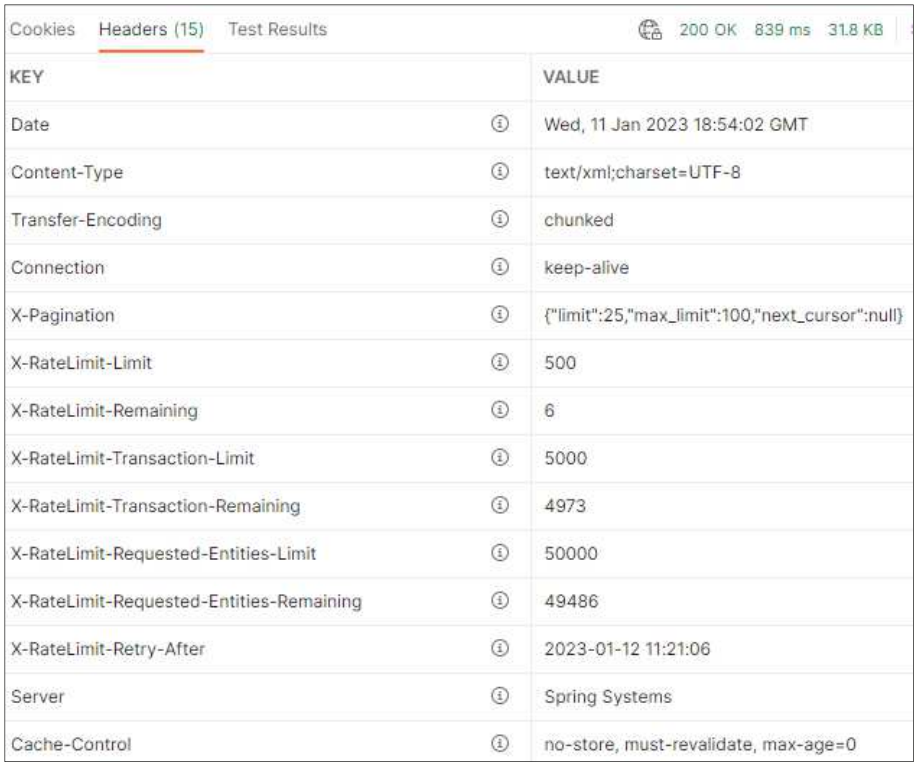
Body Response Example: Throttle Limit Reached:

```
"errors": [
    "Rate limit exceeded"
]
```

Body Response Example: Burst Time Limit Exceeded:

```
"errors": [
    "Too many requests"
]
```

Header Response Example:



Cookies Headers (15) Test Results 200 OK 839 ms 31.8 KB

KEY	VALUE
Date	Wed, 11 Jan 2023 18:54:02 GMT
Content-Type	text/xml;charset=UTF-8
Transfer-Encoding	chunked
Connection	keep-alive
X-Pagination	{"limit":25,"max_limit":100,"next_cursor":null}
X-RateLimit-Limit	500
X-RateLimit-Remaining	6
X-RateLimit-Transaction-Limit	5000
X-RateLimit-Transaction-Remaining	4973
X-RateLimit-Requested-Entities-Limit	50000
X-RateLimit-Requested-Entities-Remaining	49486
X-RateLimit-Retry-After	2023-01-12 11:21:06
Server	Spring Systems
Cache-Control	no-store, must-revalidate, max-age=0

Field List & Mandatory Fields by Retailer

Please follow this line for a list of:

- PortalApp Attributes – first tab in sheet
- Retailer Mandatory Fields (required fields vary by retailer) – 2nd tab in sheet.

List can be accessed at:

https://docs.google.com/spreadsheets/d/15ctAyYUw0IKGiUb0abqZGYjgh4P1i4nhVooSnzG75N0/

Please notify integration@springsystems.com if you find any missing fields. Sometimes new data is added by a retailer without our knowledge. This will be passed with the transaction data but not yet defined and documented in our system. Thank you for your help!

Get Purchase Order from Spring

Request Parameters Filters

Can filter on any of these. At least one is mandatory.

- po_created
- po_updated
- po_num
- po_rel_num
- po_acknowledge_status
- po_ship_status
- po_invoice_status
- po_ship_open_date
- po_ship_close_date
- po_id
- vendor_id
- retailer_id
- mark_for_location_id
- ship_to_location_id
- po_original_num
- po_type

Request Parameters Qualifiers

eq	equal
neq	not equal
lt	less than
lte	less than or equal
gt	greater than
gte	greater than or equal
like	like
nlike	not like
in	in
nin	not in

Request Parameters Format

Usage of one or more parameter qualifiers is mandatory.

po.filter.<qualifier>.<value>

example using more than one parameter...

https://api_user@api_key:portalapp.springsystems.com/api/po-outgoing/export/po.filter.gt.po_created/2017-09-01T00:00:00Z/po.filter.lt.po_created/2017-09-25T14:28:35-0500

Endpoint

Get POs: /api/po-outgoing/export

XSD – Collect Purchase Order from Spring

https://portalapp.springsystems.com/xsds/transactions/850/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/850/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/850/general.json

Send Purchase Order to Spring

Endpoint

Create POs: /api/po-incoming/send

XSD – Send Purchase Order to Spring

https://portalapp.springsystems.com/xsds/transactions/850/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/850/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/850/general.json

Get Ship Request (940) from Spring

While most transactions are supported in both directions, we find it most efficient for the *transaction originator* to initiate the call. E.g. while we support an external party calling into our system to 'Get Ship Request (940) from Spring', we have found it faster and more efficient for our system to instead push the Ship Request (940) out to the receiving system (e.g. your WMS). Our system is the originator of the Ship Request (940) transaction so it is faster if we immediately push it out to the WMS (warehouse management system). Similarly, we prefer for the WMS to push back to us the completed shipment.

Request Parameters

Can filter on any of these. At least one is mandatory.

shipment_created
shipment_updated
po_num
shipment_num
shipment_ship_status
shipment_ship_open_date
shipment_ship_close_date
shipment_id
vendor_id
retailer_id
ship_to_location_id

Request Parameters Qualifiers

eq	equal
neq	not equal
lt	less than
lte	less than or equal
gt	greater than
gte	greater than or equal
like	like
nlike	not like
in	in
nin	not in

Request Parameters Format

Usage of one or more parameter qualifiers is mandatory.

shipment.filter.<qualifier>.<value>

example using more than one parameter...

https://api_user@api_key:portalapp.springsystems.com/api/ship-request-outgoing/export/ship_info.filter.gt.ship_info_created/2017-09-01T00:00:00Z/ship_info.filter.lt.ship_info_created/2017-09-25T14:28:35-0500

Endpoint

Get Shipments: /api/ship-request-outgoing/export

XSD – Collect Ship Request (940) from Spring

https://portalapp.springsystems.com/xsds/transactions/940/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/940/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/940/general.json

Send PO Acknowledgement to Spring

Endpoint

Create Acknowledgements: /api/acknowledgement-incoming/send (under construction)

PO Acknowledgement from Spring

https://portalapp.springsystems.com/xsds/transactions/855/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/855/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/855/general.json

Get PO Acknowledgement from Spring

While most transactions are supported in both directions, we find it most efficient for the *transaction originator* to initiate the call. E.g. while we support an external party calling into our system to 'Get PO Acknowledgement from Spring', we have found it faster and more efficient for our system to instead push the PO Acknowledgement out to the receiving system

Endpoint

Get Acknowledgements: /api/acknowledge-outgoing/export (under construction)

PO Acknowledgement from Spring

https://portalapp.springsystems.com/xsds/transactions/855/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/855/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/855/general.json

Send Shipment to Spring

Usage Notes

Please provide as much data as possible about the Shipment and the Purchase Orders. At a minimum, we need to identify the Purchase Order and the Items.

Purchase Order Level

Purchase Orders for your shipment must already exist in our system. In order for us to identify the Purchase Order(s) in your shipment, you must provide back to Spring Systems ***either*** our `<po_id>` (given to you in the initial PO data), ***or*** retailer_id, retailer:tp_name, and po_num. Note that po_rel_num, mark_for_location:tp_location_id, and/or mark_for_location:tp_location_code are also mandatory if they are used for a particular PO.

Item Level

Items for your shipment should already exist in our system and in your Purchase Order. In order for us to identify the Items(s), you must provide back to Spring Systems ***either*** our:

`po_item_id` or…..

`our product_id` or…..

`product:product_additional:identifiers:gtin` or…..

`product:product_additional:vendor_item_num` (Must be unique. Only use if you do not have a GTIN.

These items would have been given to you in our initial Purchase Order data.

Trigger outgoing transaction

Please always indicate whether to trigger subsequent outgoing transactions such as sending an Advanced Ship Notice (ASN/EDI856) by passing a value of 1 within the tag:
`<send_outgoing_transaction_after>`

Passing an empty tag will indicate to not trigger subsequent transactions.

Updating a preexisting shipment

If the Spring Systems `<ship_info_id>` tag is provided and matches an existing shipment then the shipment with that specific id will be updated. If `<ship_info_id>` is not provided, then a new shipment will be created.

Update carton level

If a carton number is provided, this will update that carton, such as changing the contents or quantity. To remove a carton from a shipment, pass the tag *delete_carton_flag* with a value of 1 within the ship carton section. This will remove that carton from the shipment. Example:

```
<ship_carton>
<delete_carton_flag>1</delete_carton_flag>
</ship_carton>
```

Externally Assigned Carton Numbers

Maintaining the uniqueness `ship_carton_number` is a critical concept, and especially important for GS1-128 labels and EDI ASN's (856). External systems that are sending shipment data to PortalApp can provide these numbers or can request PortalApp to generate these numbers for them. If external system will be assigning carton numbers, these fields are mandatory:

`ship_carton:ship_carton_number` (must NOT include check digit)
`ship_carton:warehouse_carton_number`
`ship_carton:warehouse_carton_number_has_checkdigit`

Do not include the check digit when passing `ship_carton_number` to PortalApp. Failure to properly do this will risk duplication of carton numbers and incorrect labels if a user attempts to re-print labels from PortalApp.

If external system will be assigning carton numbers, PortalApp needs to know exactly what label number was printed on the GS1-128 label and whether this number already includes a check digit.

Third party applications can provide carton numbers to PortalApp (`ship_carton_number`) or this could be left blank and PortalApp will generate a carton number. If PortalApp generates the carton number, however, the third party application must store this number as it will be mandatory for processing a carton update or delete procedure.

For more detail refer to Appendix.

Request Shipping Documents

This endpoint can also be used to request shipping documents from Spring Systems. See "Appendix I – Auto Delivery of Shipping Documents" for more information.

Mandatory Fields

Once the Purchase Order and Items have been identified, you must of course also provide all mandatory shipment information (retailer dependent) such as

`ship_info:ship_info_ship_date`
`ship_info:ship_info_delivery_date`
`ship_info:ship_info_carrier_name`
`ship_info:ship_info_carrier_scac`
`ship_info:ship_info_carrier_code`
`ship_info:ship_info_tracking`
`ship_info:bill_of_lading`
`ship_info:master_bill_of_lading`
`ship_info:load_number`
`ship_info:trailer_number`
`ship_info:seal_number`
`ship_info:ship_pay_method`
`ship_info:weight.value` (and/or carton weight)
`ship_info:weight.units_of_measure` (and/or carton weight)
`ship_info:ship_to_location_id`
`ship_info:ship_to_locat`ion (all necessary detail)

`ship_info:ship_from_location` (all necessary detail)
`ship_carton:ship_carton_number`
`ship_carton:warehouse_carton_number`
`ship_carton:warehouse_carton_number_has_checkdigit`
`ship_carton:po_item_pack:po_item_pack_qty`
`mark_for_location:tp_location_id` -and/or-
`mark_for_location:tp_location_code`

And anything else that may be required on the ASN for that particular Trading Partner.

Endpoint

Create Shipments: /api/ship-incoming/send

XSD – Send Shipment to Spring

https://portalapp.springsystems.com/xsds/transactions/856/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/856/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/856/general.json

Generate Shipping Documents from Spring

Usage Notes

This endpoint can be used to request shipping documents from Spring Systems. See "Appendix I – Auto Delivery of Shipping Documents" for more information.

Endpoint

Generate Shipment Docs /api/ship-docs/generate

Sample Data

https://portalapp-staging.springsystems.com/xsds/transactions/856/general.xsd

Get Shipment from Spring

Endpoint

Get Shipments: /api/ship-outgoing/export

XSD – Send Shipment to Spring

https://portalapp.springsystems.com/xsds/transactions/856/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/856/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/856/general.json

Usage Notes

Similar types of Request Parameters and Request Parameters Qualifiers as specified earlier also apply for this endpoint.

Example

https://portalapp-staging.springsystems.com/api/ship-outgoing/export.json/ship_info.filter.gte.ship_info_created/2020-02-10%2013:45

Send Invoice to Spring

Usage Notes

Please provide as much data as possible about the Invoice and the Purchase Orders. At a minimum, we need to identify the Purchase Order.

Purchase Order Level

Purchase Orders for your shipment must already exist in our system. In order for us to identify the Purchase Order(s) in your shipment, you must provide back to Spring Systems ***either*** our po_id (given to you in the initial PO data), ***or*** retailer_id, retailer:tp_name, and po_num. Note that po_rel_num, mark_for_location:tp_location_id, and/or mark_for_location:tp_location_code are also mandatory if they are used for a particular PO.

Invoice Item Data

First priority, if you include detailed invoice data within the `<invoice_po>` and it's child `<invoice_po_item>`, this is what will be used to build the detailed invoice line item data.

Second priority, if you do not include <invoice_po_item> but do include <invoice_ship_info>, then all relevant data including invoice line item data will be pulled directly from the shipment.

Third priority, if you do not include either of the above, the your invoice will built directly from the Purchase Order specified with <po>. It will be assumed that you are invoicing the PO completely, e.g. no adjustments to quantity or price.

Invoice Numbering

If <invoice_num> is provided it will take priority. If not provided, our system will increment by one from the last invoice number used. It will skip over any previously used invoice numbers.

Consolidated vs. Non-Consolidated Invoices

When dealing with multi-location Purchase Orders (multiple mark for's), most retailers prefer non-consolidated invoices. E.g. one invoice per store (mark for location). However some retailers mandate that you consolidate invoices by Purchase Order & Shipment (e.g. a multi-location (mark for) Purchase Order could get one invoice per PO & Shipment. For retailers that ask for consolidated invoices, you should list one invoice number and multiple <po_id>. Another approach could be one invoice number and one <ship_info_id>. Note that if this shipment contains multiple base PO numbers, we would need to split this into multiple invoices and we would increment sequentially above the invoice number provided.

Mandatory Fields

Once the Purchase Order and Shipment and Items have been identified, you must of course also provide all mandatory Invoice information (retailer dependent). If this detail is provided in your Invoice data, that will take priority. If not provided in your Invoice data then we will attempt to pull the data from the Shipment. An error will be thrown if mandatory fields are not provided and can not be pulled from the Shipment or PO.

Endpoint

Create Invoices: /api/invoice-incoming/send

XSD – Send Invoice to Spring

https://portalapp.springsystems.com/xsds/transactions/810/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/810/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/810/general.json

Get Invoice from Spring

Endpoint

Get Invoices: /api/invoice-outgoing/export

XSD Invoice from Spring

https://portalapp.springsystems.com/xsds/transactions/810/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/810/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/810/general.json

Product Catalog

Usage Notes

Users can download, upload or update their product catalog using the following webservice endpoints.

XSD – Product Catalog

https://portalapp.springsystems.com/xsds/transactions/832/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/832/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/832/general.json

Endpoint

Send Product Catalog to Spring: /api/catalog-incoming/send

Receive Product Catalog from Spring:: /api/catalog-outgoing/export

Product Inventory

Usage Notes

Users can upload or update their product inventory using the following webservice endpoints.

XSD – Product Inventory

https://portalapp.springsystems.com/xsds/transactions/846/general.xsd

Sample Data

XML Sample: https://portalapp.springsystems.com/xsds/transactions/846/general.xml

JSON Sample: https://portalapp.springsystems.com/xsds/transactions/846/general.json

Endpoint

Send Product Inventory to Spring: /api/ inventory-incoming/send

Appendix I – Auto Delivery of Shipping Documents

Shipping documents can be auto generated by the Spring Systems PortalApp and delivered to a third party application. This prevents the need for our partner applications to design the different label and packing slip formats required by each retailer. Shipping documents can include:

- GS1-128 labels
- Logo Packing Slip
- FedEx Labels
- UPS Labels

Usage Notes

METHOD - Shipping documents can be delivered by the PortalApp three different ways:

- Delivered within the 940
- Returned as a response to a 945
- Returned as a response to an ad-hoc request

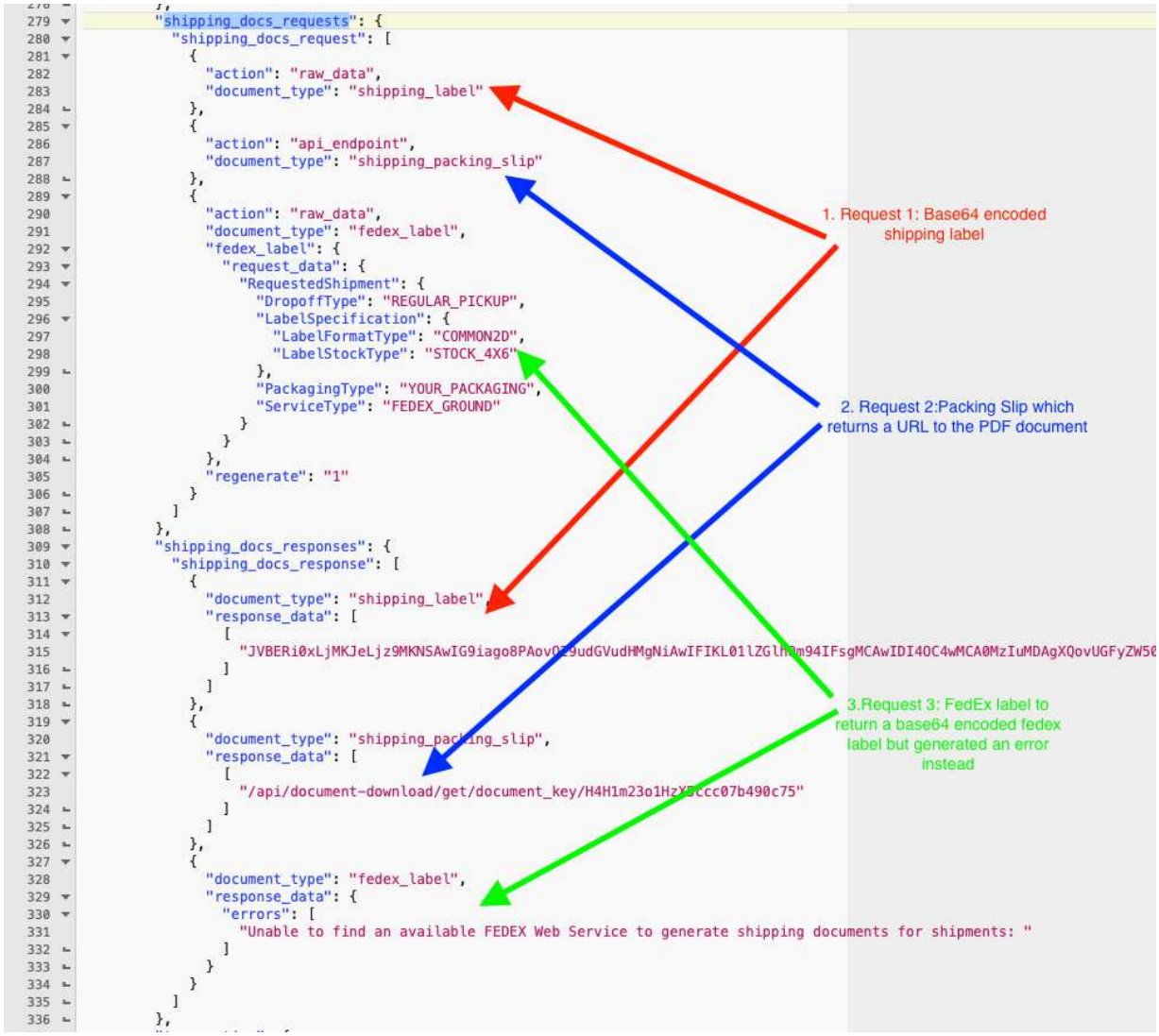
FORMAT - Shipping documents can be delivered by the PortalApp API response in two different formats:

- Unique URL to hosted PDF file, Receiving system would use your API credentials and this unique string to build the URL.
- PDF file represented in Base64 text string. Receiving system would convert this text data back into a PDF file.

Delivered Within the 940

Internal PortalApp settings must be pre-configured for a 940 to include shipping documents. Additionally the items within the 940 must be pre-packed into the appropriate carton configuration.

The following sample provides several scenarios of a 940 that includes shipping documents:



Returned As a Response to a 945 or an Ad-Hoc Request

When sending a 945 (Send Shipment to Spring) that needs to have shipping documents returned, or when using the standalone “Generate Shipping Documents from Spring” endpoint, specific requests and settings need to be provided.

(New Document)

```
//Return Base64 encode shipping label
{
  "shipping_docs": {
      "action": "raw_data",
      "document_type": "shipping_label"
    }
}

OR

//Return URL to download packing slip PDF
{
  "shipping_docs": {
      "action": "api_endpoint",
      "document_type": "shipping_packing_slip"
    }
}

OR

//Return Base64 Encoded string of the PDF for Fedex Label
{
  "shipping_docs": {
      "action": "raw_data",
      "document_type": "fedex_label",
      "fedex_label": {
        "request_data": {
          "RequestedShipment": {
            "DropoffType": "REGULAR_PICKUP",
            "LabelSpecification": {
              "LabelFormatType": "COMMON2D",
              "LabelStockType": "STOCK_4X6"
            },
            "PackagingType": "YOUR_PACKAGING",
            "ServiceType": "FEDEX_GROUND"
          }
        }
      },
      "regenerate": "1"
    }
}
```

Examples - Returned As a Response to a 945 or an Ad-Hoc Request

In all examples below, `ship_carton_number` can be provided or PortalApp can provide this back to in the response. This number must be maintained for update or delete options.

Scenario 1 - Build shipment, WMS tells full packing info, send ASN now (yes, works now)

- Step 1 (e.g. first API call) - Create a shipment, WMS provides all packing, Trigger outgoing transaction = 1; Request Shipping Documents = yes. PortalApp replies with the labels and sends the ASN.

Scenario 2 - Build shipment, WMS tells full packing info, send ASN later

- Step 1 - Create a shipment, WMS provides all packing, Trigger outgoing transaction = 0; Request Shipping Documents = yes. PortalApp replies with the labels, shipment number, and does not send the ASN.
- Step 2 – Shipment update and send ASN for this specific shipment number (shipment number from step 1 must be provided), Trigger outgoing transaction = 1.

Scenario 3 - Build shipment, WMS tells each carton, send later

- Step 1 - Create a shipment, WMS provides packing of first carton (or not, maybe just create shipment), Trigger outgoing transaction = 0; Request Shipping Documents = yes; PA replies back with shipment number
- Step 2 - Shipment update. Update carton contents of an existing carton. Important, must give us back the shipment number that PortalApp provided in step 1.
- Step 3 - Shipment update. Add carton to this shipment. Important, must give us back the shipment number that PortalApp provided in step 1.
- Step 4 - Shipment update. Delete carton from existing shipment. Important, must give us back the shipment number that PortalApp provided in step 1.
- Step 5 - Shipment update. Add final tracking info and send ASN. Important, must give us back the shipment number that PortalApp provided in step 1.Trigger outgoing transaction = 1.

For each Shipment update step, the third party application does not need to provide carton detail. If carton detail is provided it will update those cartons. Carton numbers from step 1 must be provided. If carton detail is sent again but carton number are not included, new (and possibly duplicate) cartons will be created and added to your shipment and ASN.

Appendix II – Carton Numbering

Third party applications can provide carton numbers to PortalApp or this could be left blank and PortalApp will generate a carton number. If PortalApp generates the carton number, however, the third party application must store this number as it will be mandatory for processing an carton update or delete procedure.

Third party applications that generate label numbers should follow the GS1 -128 standard convention. A full explanation is provided below, but in summary:

- 19 digit number (normally the 20th digit is a check digit number, but if you leave that off, PortalApp will append it)
- Starts with 4 leading 0's
- First half is the vendor's GS1 prefix (see below)
- 2nd half is a counter, NEVER repeat

The combination of the prefix and the counter makes these numbers globally unique.

If leading 0'sare not provided, PortalApp will add them back in. So a valid values would be:

- 0000111111012345678
- Or… 111111012345678

Red 1's represent the vendor's GS1 prefix, blue represents is the counter, incrementing for each label.

What is a GS1 prefix?

Given by the GS1. It is a 6-10 digit code that you use to assign a UPC-12 code to your products in order for those products to be scanned/sold

How can I acquire a GS1 prefix? Go to the GS1 website and fill out the forms required

What is a GS1-128 / SSCC / SSCC-18 / Serial Shipping Container Code

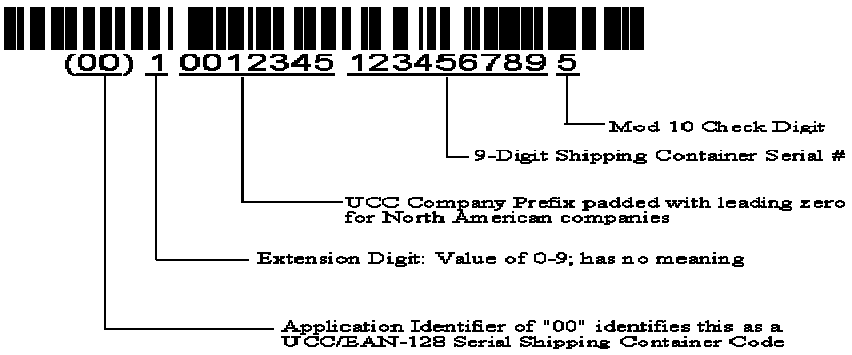
The EAN.UCC System also defines a method of serializing cartons (Serial Shipping Container Code or SSCC), so the contents can be traced to a specific line item (or items) on a specific purchase order. The standard format for serializing cartons and shipping containers is an 18 digit number encoded in UCC/EAN-128 symbology shown in the following illustration. This number **is not generally added to the box or container until the time of shipment**.

Serial numbering of cartons provides unique identification. This is important, especially to identify cartons containing variable quantities or non-standard mixtures of product and as a reference number for EDI (Electronic Data Interchange) transactions. When used in conjunction with EDI, serial numbering cartons and shipping containers eliminates the need to physically inspect the contents of every carton.

Eliminating this procedure has improved the productivity of some warehouse receiving operations by 100% or more!

The Serial Shipping Container Code (SSCC) is different from the EAN/UCC-14 assigned to intermediate packs and shipping containers of consumer units. The same EAN/UCC-14 is assigned to all identical packaging configurations of the same product. The Serial Shipping Container Code (SSCC) is a unique number assigned to each carton as it is shipped. Because it is unique, it can be used as a reference number tying the contents of a specific carton to information about the shipment including the purchase order(s) it should be applied to, carrier, date of shipment, etc. It is used in conjunction with the EDI 856 Advance Shipping Notice (ASN).

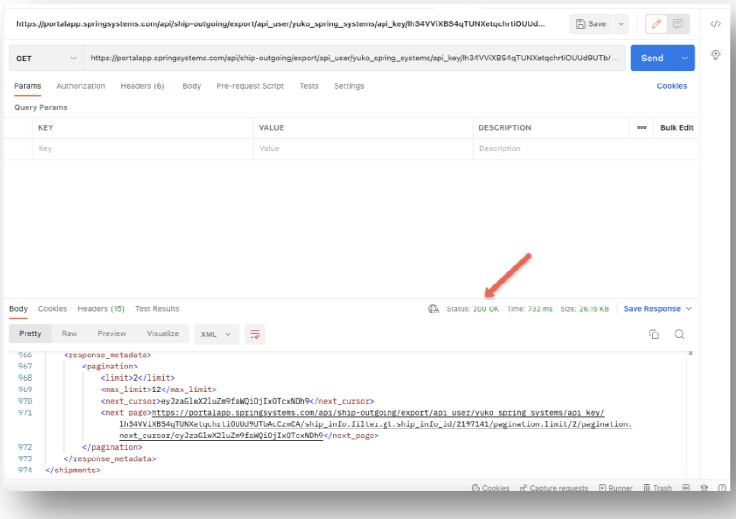
The 18 digit number consists of the following items:



Appendix III – API Response Codes

To ensure successful communication, the sending system must verify a successful API or web service response code.

Example 1 - Postman tool displays a response code as follows:



Example 2 - This is how it looks like on raw api call, the code is in response, how to find and interpret the code depends on the system that sent the request.