API Reference

Transact APIs Reference
Transact Gateway Service

AWS

Authored by: Nicholas Leuci

API Version 2 2024-06-21 **PRELIMINARY**

AWS Transact Gateway Service: Transact APIs Reference

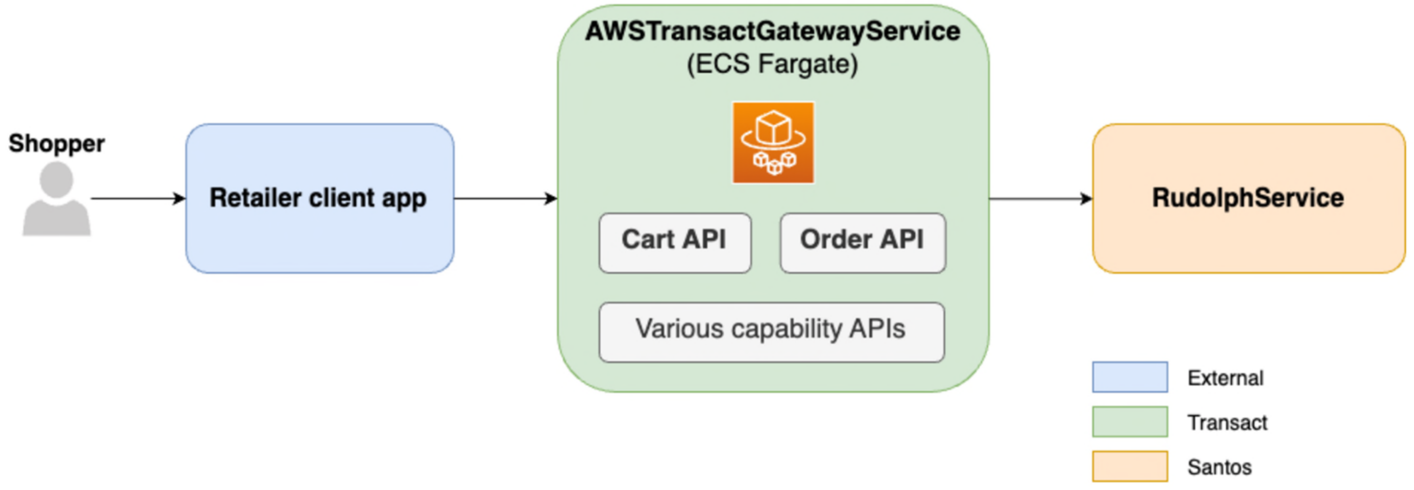
Table of Contents

Introduction and Needed Services

The AWS Transact Gateway Service is a cloud-based set of composable server system solutions available for the Retailer Developer intended as the primary user. This document defines and establishes API standards for AWS Transact Order Management APIs to enable the Retailer Developer to create a shopping experience for shoppers to capture a cart, checkout their cart, place, cancel and return an order. These APIs are intended for both online eCommerce and in-person Point of Sale shopping experiences.

All AWS Transact Gateway Service API operations are Amazon-authenticated and certificate-signed. They not only require the use of the AWS SDK, but also allow for the exclusive use of AWS Identity and Access Management users and roles to help facilitate access, trust, and permission policies. Alternatively, retailers can make REST calls to Transact APIs and build the **`aws auth sig`**v4 headers themselves, although it would be more work than using the provided AWS Transact SDKs.

This document describes the Cart and Order APIs that are at the core of the Transact services for the retailer client apps to call. These APIs are developed in **smithy**, an Amazon AWS internal Interface Definition Language (IDL). For both Cart and Order, Transact will call various internal APIs to create, read, update, delete, and list carts and orders. For simplicity, this document will bias towards modeling Cart and Order domain models uniformly as much as possible.



Here is a summary of the needed services for the Cart and Order APIs described in this document.

Transact Cart functional services:

1. **Create a cart:** A retailer should be able to use Transact to create a cart with a minimum of one product item.
2. **Add a line item to cart:** A retailer should be able to use Transact to add one or more line items to a cart.
3. **Delete a line item from cart:** A retailer should be able to use Transact to delete a line item from a cart.
4. **Edit cart line item quantity:** A retailer should be able to use Transact to edit the line item quantity.

5. **Get cart by ID:** A retailer should be able to get a cart object from Transact by the Cart ID.
6. **Search / list carts:** Retailer should be able to get a list of carts meeting certain criteria.
7. **Edit cart metadata:** Retailer should be able to edit cart metadata (shopper ID, session ID, client ID, brand ID, store ID).
8. **Delete cart:** Retailer should be able to discard a cart.
    1. **Delete stale orders:** Transact must provide a service to:
        1. periodically mark for deletion those carts which have not been updated for longer than some maximum time configured by the retailer.
        2. periodically delete carts that have been marked for deletion.
9. **Label cart:** A retailer should be able to label a cart with a retailer-provided arbitrary label (e.g. “Archived”, “Saved”, “Active”, “Wishlist”, “Cart”, “Grocery”, “John’s Cart”, etc.).
10. **Validate cart:** A retailer should be able to pass in a cart object to be validated by Transact, in case the client has reasons for creating or managing carts outside of Transact.

Transact Order functional services:

1. **Create an order:** A retailer should be able to create an order so that a retailer and shopper can agree on the terms of a “sales contract”.
2. **Update orders:** A retailer should be able to update an order’s data or add new data so that negotiation failures can be resolved and so that item quantities can be adjusted.
3. **Negotiate order (create/update/validate order):** A retailer should be able to negotiate an order, so that it can attempt to validate the contents of the order and generate a negotiated contract it can eventually finalize / sign on behalf of the customer.
4. **Discard order:** As a client, I need to be able to discard a “Created” or “Negotiated” order that I know will not become Finalized, so that it can be deleted and so that I’m not charged for it’s storage.
    1. **Discard stale orders:** Transact must provide a service to:
        1. periodically mark for deletion those orders which have been “Created” and not updated for longer than some maximum time.
        2. periodically mark for deletion those orders which have been “Negotiated” but have not been finalized prior to their TTL.
        3. periodically delete orders that have been marked for deletion.
5. **Finalize order:** As a client, I need to be able to “finalize” or “sign” and order, so that I can reflect a mutual agreement between shopper and retailer, and so that I can fulfill the order and charge the shopper for it.
6. **Cancel order & line items:** As a client, I need to be able to cancel an order that is not yet in fulfillment (no items shipped yet), so that shoppers can reverse their decision to make a purchase before it’s fulfilled.
    1. (Private Preview) Additionally, I need to be able to cancel a line item that is not yet in fulfillment from an order, so that shoppers can reverse their decision to make a purchase before it’s fulfilled.
7. **Update fulfillment status (Private Preview):** As a client, I must be able to consume an event stream to which the fulfillment provider publishes fulfillment events, so that I can update the fulfillment state of the order and line items on the order.
8. **Search / list orders:** Clients must be able to list/search for orders that meet specific criteria, so that they can list them for shoppers.

9. **Get order by ID:** Clients must be able to retrieve a specific order by ID, so that they can show that order to the shopper.
10. **Return Line item (Private Preview):** As a client, I need to be able to initiate a return for a line item on an order.
11. **Refund Line item (Private Preview):** As a client, I need to be able to initiate a refund for a line item on an order.

The Transact **Catalog API** is intended to provide product information necessary to be shown in a checkout page and to validate product existence during order orchestration. This service differs from a Product Information Management (PIM) tool that is used by retailers to add, update or delete product information. In order to add, delete or update the product information itself, the retailer will have to use their PIM/Catalog service directly (outside of Transact). Succinctly, the Transact Catalog API does not provide catalog maintenance, but it does provide the means for shoppers to find and select products for ordering from the retailer's catalog.

The Transact **Pricing API** is intended to provide the functional pricing information viewable in the checkout page and to validate the accuracy of product prices used during order orchestration. Thr Pricing API will only provide the product base and sale price. It will be able to support line-item level strike through pricing. Retailers can also process tired pricing based on the way they setup their pricelist. Promotions & Loyalty lie outside of the Pricing API, but it will exist in a separate capability. Transact allows retailers to have providers for each of these capabilities outside of the Pricing API. This also allows retailers the choice to maintain pricing separate from these capabilities.

In the beta release, Transact will not support authentication, account creation, updating or deletion of shopper profiles. For these services, the retailer will need integrate directly with their shopper capability provider. Only providing shopper profile and address look-up through Transact would require additional effort for the retailer to integrate their shopper capability directly and through Transact. In order to have a single point of interaction with the shopper capability, Transact will keep shopper capability out of the scope during beta. From an ordering standpoint, (for checkout page and beyond), Transact assumes the retailer will provide all the information needed for an order (shopperID, shipping address and billing address).

The Transact **Tax API** will calculate the applicable tax information to show to a shopper for both e-commerce and physical store orders. The Transact core ordering workflows will focus on tax calculations, but the tax calculations performed during ordering will also help drive retailer-owned shopper facing experiences tied to post purchase functions (e.g., receipts, order history, order details). These same tax calculations will be used to support retailer-owned tax reporting and remittance processes, in conjunction with 3P tax solution providers. Retailers are responsible for establishing contractual relationships with their 3P tax providers/partners, as part of the capability offerings in the Transact AWS marketplace. Retailers will need to utilize their 3P tax solution front-end capabilities to setup/register their accounts and continuously manage their tax data needs.

API Actions

The following Cart and Order actions and their primary data types are supported and described:

Cart API

- CreateCart
- CreateCartLineItems
- GetCart
- UpdateCart
- DeleteCart
- ListCarts

Order API

- CreateOrder
- CreateOrderLineItems
- GetOrder
- UpdateOrder
- UpdateOrderFulfillments
- UpdateOrderReturns X
- UpdateSignedOrder X
- DeleteOrder
- ListOrders
- SignOrder
- CancelOrder X

Catalog APIs

- GetProduct
- BatchGetProduct

Pricing APIs

- GetProductPrices
- GeProductPricesForLineItems

Tax API

Data Types

Cart APIs

Note that carts may be created, managed, and stored using Transact via these Cart APIs, but retailers may choose to create, manage, and store carts outside of Transact as well. Transact.engineId will be obtained from the request URI to pass to Transact for all Cart APIs. There is internal caching to map from **`engineId`** to **`orderManagerId`** and **`contractDefinitionId`**, both of which are needed in Transact.

CreateCart

`CreateCart` calls the internal Transact API. This initiates order negotiation for the purpose of getting a list of constraint violations. It returns a Cart object and its status.

Request Syntax

```
POST /carts
Content-type: application/smithy

@idempotent
@http(code: 201, method: "POST", URI: "/carts")
operation CreateCart {
    input: CreateCartInput
    output: CreateCartOutput
    errors: [
        ValidationException
        AccessDeniedException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs:

- Required
  - List of line item inputs, each minimally containing:
    - Product ID
    - Quantity
      - Unit
      - Amount
- Optional:
  - Retailer ID
  - Store ID

- Client ID
- Shopper ID
- Session ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Cart created in Transact, and is assigned a Cart ID unique to a Transact engine
- Cart gets assigned Transact engine metadata specifying the Transact engine and Transact engine version info.
- The "Add item(s) to cart" workflow gets triggered using the provided list of line items as input. See CreateCartLineItems.

If the above actions are successful, cart creation date reflects when the API was called.

Response Structure

```
Status Code: 201 Created
{
    cart: Cart
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The output is a newly created cart object with corresponding line item(s) and calculated (sub)totals. The following data is returned in smithy format by the Transact service.

Cart

The newly instantiated Cart object with corresponding line item(s) and calculated (sub)totals

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException

Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

CreateCartLineItems

This service calls the internal Transact APIs. This API service is for creating/adding new line items to the instantiated cart. It returns the updated Cart with added LineItem(s) and gets the status and details of CreateCartLineItems.

Request Syntax

```
POST /carts/{cartId}/lineItems
Content-type: application/smithy

{
  lineItems: LineItem[]
}
@http(code: 201, method: "POST", URI: "/carts/{cartId}/line-items")
operation CreateCartLineItems {
    input: CreateCartLineItemsInput
    output: CreateCartLineItemsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ConflictException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
```

URI Request Parameters

The request uses the following URI parameters as inputs:

- Required:
  - Cart ID
  - List of line item inputs, each minimally containing:
    - Product ID
    - Quantity
      - Unit
      - Amount

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- The “add line item” workflow is triggered

- Sample basic "add line item" workflow:
    - Transact confirms that the Cart ID exists. If not, returns an error
    - For each line item input in the list of line item inputs:
        - Transact calls Catalog provider to get details about the Product associated with the Product ID.
        - If product ID doesn't exist or is not sellable, return an error.
        - If the product's base unit does not match the unit specified by the client, return an error.
            - Unit
            - Amount
        - Add the following product information to the line item:
            - Product Name
            - Product Description
            - Product Thumbnail
            - Price
                - Unit
                - Currency
                - Amount
        - A new line item with a line item ID unique to the cart is created, with the product ID provided as input and the product information from the product provider included, and added to the list of line items in the cart.
    - The "last updated date" of the order is updated to the current time.
    - Transact calculates and updates the line item subtotal.
    - Transact calculates and updates the subtotals for cart based on the newly added line items.

Response Structure

Status Code: 201 Created

```
{
    cart: Cart
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. It returns the updated Cart object with new line items added. The following data is returned in smithy format by the Transact service.

Cart

The updated Cart object with instantiated LineItem(s).

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

GetCart

This service calls the internal Transact API. This API service is for reading / returning a Cart object and any associated line items for the instantiated cart. It returns a Cart and gets the status and details of GetCart.

Request Syntax

```
GET /carts/{cartId}
Content-type: application/smithy

{
  lineItems: LineItem[]
}
@readonly
@http(code: 200, method: "GET", URI: "/carts/{cartId}")
operation GetCart {
    input: GetCartInput
    output: GetCartOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
```

URI Request Parameters

The request uses the following URI parameters as inputs:

- Required:
    - Cart ID
- Optional:
    - Refresh cart flag

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check if the cart ID provided exists. If not, return an error.
- If the refresh cart flag is not provided:
    - Return the cart object with the ID provided
    - The “last updated date” of the order is updated to the current time.
- If the refresh cart flag is provided:
    - The “refresh cart” workflow is triggered
    - Sample basic “refresh cart” workflow:

        - Check if the cart ID provided exists. If not, return an error.
        - For each line item, get product catalog data for its product ID and update the line item with the new data if there is any.
        - For each line item, get updated pricing data for its product ID and update the line item with the new price data if there is any.
        - The "last updated date" of the order is updated to the current time.
    - The subtotals for the cart are calculated based on the new line item data.

Response Structure

Status Code: 200 OK
```
{
    cart: Cart
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The Transact service returns the updated Cart object with instantiated LineItem(s) in smithy format.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

UpdateCart

This service calls the internal Transact API. A constraint violation is returned if the request contains a line item that is not already in the cart. For a Cart object**, i**t gets the status and details of UpdateCart. This API service should address the following requirements:

- delete cart lineitem
- edit lineitem quantity
- edit cart metadata
- label cart
- validate cart

Request Syntax

```
PATCH /carts/{cartId}
Content-type: application/smithy

{
  lineItems: LineItem[]
}
@idempotent
@http(code: 200, method: "PATCH", URI: "/carts/{cartId}")
operation UpdateCart {
    input: UpdateCartInput
    output: UpdateCartOutput
    errors: [
        ValidationException
        AccessDeniedException
        ConflictException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

Delete line item:

- Required:
    - Cart ID
    - Line item ID

Edit line item:

- Required:
    - Cart ID
    - Line item ID
    - New quantity
        - Unit
        - Amount

Edit cart metadata:
- Required: Cart ID
- Optional:
    - Shopper ID
    - Session ID
    - Client ID
    - Merchant ID
    - Store ID

Label cart:
- Required:
    - Cart ID
    - Label key-value pair.
    - Add or Remove Flag

Validate cart:
- Required: Cart object (without Transact-generated cart ID, Transact engine metadata)

Request Actions

The request does not have a request body. The Transact service performs the following actions:

Delete line item:
- The “delete line item” workflow is triggered
- Sample basic “delete line item” workflow:
    - Check if the cart ID provided and if the line item ID provided exist. If not, return an error.
    - If the line item provided is the only line item remaining, return an error, e.g., “use ‘delete cart service instead.”
    - The line item matching the line item ID provided as input is deleted from the cart ID specified
    - The “last updated date” of the order is updated to the current time.
    - The subtotals for the cart are calculated based on the deleted line item having been removed.
- In future stages of development, the above workflow may be modified to include additional capabilities as they are made available via Transact and/or additional steps to reverse actions of the “add to cart” workflow, such as:
    - Removing inventory hold using Inventory provider.

Edit line item:
- The “edit line item quantity” workflow is triggered

- Sample basic “edit line item quantity” workflow:
    - Check if the cart ID provided and if the line item ID provided exist. If not, return an error.
    - Check if the quantity unit matches that of the product. If not, return an error.
    - Set the quantity amount based on the amount specified by the client as input.
    - The “last updated date” of the order is updated to the current time.
    - The subtotals for the cart are calculated based on the deleted line item having been removed.
- In future stages of development, the above workflow may be modified to include additional capabilities as they are made available via Transact and/or additional steps to reverse actions of the “add to cart” workflow, such as:
    - Update inventory hold using Inventory provider, to reflect the new quantity.

Edit cart metadata:
- Check if cart ID exists. If not, return an error.
- Update metadata values with those specified in the input
- The “last updated date” of the order is updated to the current time.

Label cart:
- Check if cart ID exists. If not, return an error.
- If “add” flag:
    - Check if label key-value pair is already present on cart. If so, return an error.
    - Add key-value pair to the list of labels.
- If “remove” flag:
    - Check if label key value pair is already present on cart. If NOT, return an error
    - The “last updated date” of the order is updated to the current time.

Validate cart:
- Check if all non-Transact generated but required fields are present. If not, return an error.
- Check whether at least one line item is present. If not, return an error.
- Check whether all line item IDs are unique. If not, return an error.
- Check whether all units in line items match for quantity, and price. If not, return an error.
- Check whether sum of line item costs match subtotal. If not, return an error.
- Check whether sum of line item costs, taxes & fees, and discounts match the total. If not, return an error.

Response Structure

```
Status Code: 200 OK
{
    cart: Cart
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service.

Delete line item:

- Cart object with specified line item deleted

Edit line item:

- Cart object with updated line item quantity.

Edit cart metadata:

- Cart object with updated metadata values.

Label cart:

- Cart object with updated labels list.

Validate cart:

- Return a success response if cart object is valid.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

DeleteCart

`DeleteCart` calls the internal Transact API. This service discards (soft deletes) the Cart, meaning that it won't show up in any of the Cart APIs, but would still exist in the underlying cart database in Transact. Note: Transact does not currently support programmatic hard deletes Request Syntax.

Request Syntax

```
DELETE /carts/{cartId}
Content-type: application/smithy

@idempotent
@http(code: 204, method: "DELETE", URI: "/carts/{cartId}")
operation DeleteCart {
    input: DeleteCartInput
    output: DeleteCartOutput
    errors: [
        ValidationException
        AccessDeniedException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - Cart ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check if cart ID exists. If not, return an error.
- Mark cart ID for deletion, to be garbage collected by Transact. Retailer customer should stop being metered for storage of that cart object.

Response Structure

```
Status Code: 204 No Content
{
    cart: Cart
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service{ Cart ID of deleted cart

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

ListCarts

`ListCarts` calls the internal Transact API. This service gets a list of Cart objects meeting certain search criteria. Additional work is needed to be able to filter by certain fields (**`shopperId`** and **`sessionId`**). See **ListOrders** for more details.

Request Syntax

```
GET /carts
    ?shopperId={shopperId} // optional
    &sessionId={sessionId} // optional
    &nextToken={nextToken} // optional, for pagination
    &maxResults={maxResults} // optional, for pagination
    /* Additional filters Transact needs to support later.
        - brandId
        - storeId
        - clientId
        - afterCreatedDate
        - beforeCreatedDate
        - afterLastUpdatedDate
        - beforeLastUpdatedDate
    */
Content-type: application/smithy

@readonly
@idempotent
@http(code: 200, method: "GET", URI: "/carts")
operation ListCarts {
    input: ListCartsInput
    output: ListCartsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Optional:
  - MaxResults (Integer) (see AWS API Standards for pagination) (default X; upper bound: Y)
  - Pagination identifier

- Query Filters (each requires a “filter type”-“filter value” tuple) (one or more filters may be provided, with filters combining using an AND operator)
    - Shopper ID
    - Session ID
    - **Beyond Private Beta:**
        - Brand ID
        - Store ID
        - Client ID
        - Label String
        - Created datetime range
        - Last Updated datetime range

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- If the “MaxResults” input is specified, the maximum number of records to return should be that number.
    - If this input exceeds the upper bound that Transact can return at any one time, return an error.
- If the total number of records for a query exceeds the number of records to return, return the first set of records, and a pagination identifier which allows the client to get the next set of matching cart IDs for that query.
- Returned carts should be sorted in descending order by cart creation datetime (most recently created carts first).
- If the pagination identifier is included as an input, return the next set of records from the query which provided that pagination identifier.
    - If the pagination identifier is not recognized, return an error.
    - If other filters are specified in addition to the pagination identifier, return an error.
- If no query filters are specified return all carts being stored by a given Transact engine
- If multiple query filters are specified, they should be combined in the query using an “AND” operator.
- If shopper ID filter is specified, narrow the results to include only carts associated with that shopper ID.
- If session ID filter is specified, narrow the results to include only carts associated with that session ID.
- If “creation” / “last-updated” datetime filter is specified, narrow the results to only include carts which have a timestamp for the corresponding field within that time range.
- If the label filter is specified, narrow the results to only return carts that include the query string in the list of label strings.
- If “brand ID” / “store ID” / “client ID” filter is specified, narrow the results to only return orders that have the specified filter value for the corresponding field.

Response Structure

```
Status Code: 200 OK
{
    carts: String{} // list of cartIds
    nextToken: String
}
```

R

If

smithy format by the Transact service.

- List of cart IDs

- Pagination identifier (if applicable)

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Order APIs

The Transact.engineId will be obtained from the request URI to pass into Transact for all Order APIs. There is caching to map from the transact engine id to the orderManagerId and contractDefinitionId, both of which are needed to call the internal Transact APIs.

CreateOrder

This API service calls the internal Transact API. This conducts order negotiation, enabling a shopper to create an Order so that a retailer and shopper can agree on the terms of a “sales contract” An order must have at least one line item. The number of line items in an order is not constrained by Transact, although chosen capability providers (e.g., promotions, order fulfillment, taxes, etc.) may be limited in the number of line items or units they can support..

Request Syntax

```
POST /orders
Content type: application/smithy

{
 order: Order
    constraintViolations: ConstraintViolation[]
  lineItems: LineItem[]
}

@idempotent
@http(code: 201, method: "POST", URI: "/orders")
operation CreateOrder {
    input: CreateOrderInput
    output: CreateOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - If retailer is using Transact to create, manage, and store carts:
        - Transact cart ID

- If retailer is **NOT** using Transact to create, manage, and store carts (i.e. they are managing carts outside of Transact):
    - Transact-compatible cart object
    - Shopper ID (if not already included in the cart object, otherwise optional).

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Transact creates an order, which is assigned an order ID unique to the Transact engine.
- If Transact cart ID is provided as input, Transact checks whether cart ID exists. If cart ID does not exist, return an error. If it exists, record the Transact cart ID in the order.
- If a cart object is provided as input, Transact validates that it is in the Transact-compatible schema. If cart object is not a Transact-compatible object, return an error.
- Check if cart input contains at least one valid line item. If not, return an error.
- Check if shopper ID is provided as input OR cart input contains a shopper ID. If both are false, return an error.
- Add available cart data to the order (i.e., line items, shopper ID, payment info, fulfillment info, as available).
- If shopper ID is provided as input, Transact adds or updates order’s shopper ID.
- Transact adds the Transact engine metadata to the order.
- Transact internally triggers a Negotiate order workflow.

Response Structure

```
Status Code: 201 Created
{
   order: Order
   constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service: Order document, either “Created” or “Negotiated”, depending on result of the negotiate order workflow.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException

Updating or deleting a resource can cause an inconsistent state.

HTTP Status Code: 409

InternalServerException

Unexpected error during processing of request.

HTTP Status Code: 500

ThrottlingException

Request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

The input does not satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

CreateOrderLineItems

Calls the internal Transact APIs. This API service is for creating/adding new line items to an instantiated Order. It gets the status and details of a CreateOrderLineItem object.

Request Syntax

```
POST /orders
Content-type: application/smithy

{
    order: Order
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@http(code: 201, method: "POST", URI: "/orders/{orderId}/line-items")
operation CreateOrderLineItems {
    input: CreateOrderLineItemsInput
    output: CreateOrderLineItemsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - If retailer is using Transact to create, manage, and store carts:
        - Transact cart ID
        - Order ID
    - If retailer is **NOT** using Transact to create, manage, and store carts (i.e. they are managing carts outside of Transact):
        - Transact-compatible cart object
        - Shopper ID (if not already included in the cart object, otherwise optional).
    - List of line item inputs, each minimally containing:
        - Product ID
        - Quantity
            - Unit
            - Currency

        - Amount

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- The “add line item” workflow is triggered
- Sample basic “add line item” workflow:
    - Transact confirms that the Order ID exists. If not, returns an error
    - For each line item input in the list of line item inputs:
        - Transact calls Catalog provider to get details about the Product associated with the Product ID.
        - If product ID doesn’t exist or is not sellable, return an error.
        - If the product’s base unit does not match the unit specified by the client, return an error.
            - Unit
            - Amount
        - Add the following product information to the line item:
            - Product Name
            - Product Description
            - Product Thumbnail
            - Price
                - Unit
                - Currency
                - Amount
        - A new line item with a line item ID unique to the order is created, with the product ID provided as input and the product information from the product provider included, and added to the list of line items in the order.
    - The “last updated date” of the order is updated to the current time.
    - Transact calculates and updates the line item subtotal.
    - Transact calculates and updates the subtotals for the order based on the newly added line items.

Response Syntax

```
Status Code: 201 Created
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service: Order object with added line items

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

You do not have the required privileges to perform this action.

HTTP Status Code: 403

ConflictException

Updating or deleting a resource can cause an inconsistent state.

HTTP Status Code: 409

InternalServerException

Unexpected error during processing of request.

HTTP Status Code: 500

ResourceNotFoundException

Request references a resource which does not exist.

HTTP Status Code: 404

ThrottlingException

Request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

The input does not satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

GetOrder

This service calls the internal Transact API. This API is for reading / returning an Order object and its associated line items. It gets the status and details of GetOrder.

Request Syntax

```
GET /orders/{orderId}
Content-type: application/smithy

{
  lineItems: LineItem[]
}
@readonly
@http(code: 200, method: "GET", URI: "/orders/{orderId}")
operation GetOrder {
    input: GetOrderInput
    output: GetOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
`
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - Order ID
    - Cart ID
- Optional:
    - Refresh cart flag

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check if the order ID provided exists. If not, return an error.
- Check if the cart ID provided exists. If not, return an error.
- If the refresh cart flag is not provided:
    - Return the cart object with the ID provided
    - The “last updated date” of the order is updated to the current time.

- If the refresh cart flag is provided:
    - The “refresh cart” workflow is triggered
    - Sample basic “refresh cart” workflow:
        - Check if the cart ID provided exists. If not, return an error.
        - For each line item, get product catalog data for its product ID and update the line item with the new data if there is any.
        - For each line item, get updated pricing data for its product ID and update the line item with the new price data if there is any.
        - The “last updated date” of the order is updated to the current time.
    - The subtotals for the order and the cart are calculated based on the new line item data.

Response Structure

```
Status Code: 200 OK
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The Transact service returns the updated Order object with instantiated LineItem(s) in smithy format.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

UpdateOrder

Calls the internal Transact API. This API service attempts Order renegotiation by allowing a shopper to specify fulfillment and payments and to make other changes in the order. A constraint violation is returned if the request contains a line item that is not already in the order. It gets the status and details of the UpdateOrder object.

Request Syntax

```
PATCH /orders/{orderId}
Content-type: application/smithy

{
    order: Order
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@idempotent
@http(code: 200, method: "PATCH", URI: "/orders/{orderId}/line-items")
operation UpdateOrder {
    input: UpdateOrderInput
    output: UpdateOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
  - Transact order ID
- Optional:
  - Update line item quantity
    - Line item ID (required)
    - Quantity (required)
  - Add / update line item fulfillment options
    - Line item ID (required)
    - Fulfillment Option (required)
  - Remove line item

        - Line item ID (required)
    - Add / update recipient information
        - Line item ID (required)
        - Recipient information (required)
    - Add / update payment method
        - Payment card identifier (required)
    - Add / update shopper ID
        - Shopper ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check whether Transact order ID exists, if not, return an error.
- Update cart according to inputs.
    - Update line item quantity
        - Check if line item ID exists. If not, return an error.
        - Check if quantity unit and amount is allowed by the product. If not, return an error.
        - Update line item to specified quantity.
    - Add / update line item fulfillment options
        - Check if line item ID exists. If not, return an error.
        - Confirm that fulfillment option is valid. If not, return an error.
        - Update line item to specified fulfillment option.
    - Remove line item
        - Check if line item ID exists. If not, return an error.
        - Remove line item from order.
    - Add / update line item recipient information
        - If recipient information already exists, replace existing recipient information with supplied recipient information.
        - Else: Add supplied recipient information to order.
    - Add / update payment method
        - If payment method already exists, replace existing payment method with supplied payment method.
        - Else: Add payment method to order.
    - Trigger negotiate order workflow.

Response Syntax

```
Status Code: 200 OK
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service.

- Order updated with new data from input, and depending on the outcome of the negotiate order workflow:
    - in “negotiated” state
    - in “created” state, with list of errors that must be resolved by the client.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

UpdateOrderFulfillments

UpdateOrderFulfillments calls the internal Transact API. This updates the Order fulfillment statuses. This service is intended to be called by fulfillment providers once their asynchronous fulfillment process is complete for updating fulfillment statuses in the order document. This service should be able to take care of updating fulfillment status for particular line item(s) as well. This encompasses the following process:

- Events in the event stream should be published by the Fulfillment Provider when it changes the fulfillment state of a line item that was sent to the provider for fulfillment by Transact. The event should include, at a minimum, the Order ID and Line Item ID for the line item whose state has been updated by the Fulfillment Provider, and the new fulfillment state of that line item.
- Transact should monitor the event stream and update the state of the corresponding line item to reflect the latest update from the Fulfillment provider.
- Fulfillment states should be determined by the Fulfillment capability spec:
    - Currently assuming: Picked → Packed → Shipped → Out for Delivery → Delivered
- If an order has at least one item whose state has been updated to “Shipped” by the provider, the state of the order should be updated to “In Fulfillment”.
- If all of an order’s line items have been updated to “Delivered”, update the state of the order to “Complete”.

Request Syntax

```
PUT /orders/{orderId}/fulfillments
Content-type: application/smithy
{
    order: Order
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@idempotent
@http(code: 200, method: "PUT", URI: "/orders/{orderId}/fulfillments")
operation UpdateOrderFulfillments {
    input: UpdateOrderFulfillmentsInput
    output: UpdateOrderFulfillmentsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - Transact order ID
    - Shopper ID
    - Line item ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Trigger Negotiate Order workflow.
- Sample default Negotiate Order workflow:
    - Check whether Transact order ID exists, if not, return an error.
    - Confirm shopper ID is present in the order. If not, return an error.
    - Confirm order has at least one line item. If not, return an error and mark order for deletion.
    - Confirm order has shipping address if fulfillment options for any line item are “shipment” / “delivery”.
    - If payment card identifier is not specified, return an error.
    - For each line item:
        - Confirm product ID exists and is sellable in Catalog provider. If not, return an error.
        - Update line item product data with latest data for product ID from Catalog Provider.
        - Confirm line item quantity unit and amount type (int/float) matches expected unit and amount type for product ID. If not, return an error.
        - Get / update price for product ID from Pricing provider.
        - Confirm line item’s price unit matches expected unit and amount type for product ID. If not, return an error.
        - Get / update tax amount for product ID from Tax provider.
        - If fulfillment method is specified, confirm it is still valid for the line item with Fulfillment Provider. If not, return an error.
        - If fulfillment method is specified, get or update fulfillment charges for the line item with the Fulfillment provider.
        - If fulfillment method is specified, get or update fulfillment promise from the Fulfillment provider.
        - If fulfillment method is not specified, get available fulfillment options and add them to the line item for the client to consume. Return an error.
        - Calculate order subtotal. Sum of line item subtotals.
        - Calculate order fulfillment charges. Sum of line item fulfillment charges
        - Calculate order discounts. Sum of line item discounts.
        - Calculate order taxes and fees. Sum of line item taxes and fees.
        - Calculated order total. Sum of subtotal, fulfillment charges, discounts, and taxes and fees.
    - If no errors are found (all required data is present, valid, and up-to-date), set the order state to “Negotiated”.
    - If errors are found (required data missing, data invalid), set the order state to “Created” and collect all errors to be returned to client.

- In future stages of development, the above workflow may be modified to include additional capabilities as they are made available via Transact and/or additional steps such as:
    - Honoring price overrides if present.
    - Confirm shopper ID is present and in good standing in Shopper Identity provider. If not, return an error.
    - If payment card identified is specified, confirm that payment card identifier exists in Payment provider. If not, return an error.
    - Confirming inventory availability using Inventory provider.
    - Placing an inventory hold using Inventory provider
    - Associating order with held inventory using Inventory provider.
    - Passing order to Promotions provider to get promotional discounts.
    - Passing order to Fraud & Abuse provider to validate shopper and/or payment card identifier.

Response Syntax

```
Status Code: 200 OK
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service.

- Updated Order, depending on the outcome of the negotiate order workflow:
    - in "negotiated" state
    - in "created" state, with list of errors that must be resolved by the client.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException

Request references a resource which does not exist.

HTTP Status Code: 404

ThrottlingException

Request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

The input does not satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

UpdateOrderReturns

UpdateOrderReturns calls the internal Transact API. This service updates the Order return statuses and calls the internal Transact API with targetAmendmentState set to SIGNED. This service should be able to take care of updating return status for particular line item(s) as well. This API should throw an exception for returning order units that are already in the process of being returned.

Request Syntax

```
PUT /orders/{orderId}/returns
{
    order: Order
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}
Content-type: application/smithy

@idempotent
@http(code: 200, method: "PUT", URI: "/orders/{orderId}/returns")
operation UpdateOrderReturns {
    input: UpdateOrderReturnsInput
    output: UpdateOrderReturnsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

To be implemented in a later release.

Request Actions

The request does not have a request body. The Transact service performs the following actions:

To be implemented in a later release.

Response Syntax

```
Status Code: 200 OK
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service.

To be implemented in a later release.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

UpdateSignedOrder

UpdateSignedOrder calls the internal Transact API. This service is for updating a post-signed Order to support use cases like **item substitutions**. The intent is to have a separate API for this so that there can be separate permissions for this action. The details are pending.

Request Syntax

```
PATCH /orders/{orderId}
Content-type: application/smithy

{
    order: Order
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@idempotent
@http(code: 200, method: "PATCH", URI: "/orders/{orderId}/line-items")
operation CreateSignedOrderLineItems {
    input: CreateSignedOrderLineItemsInput
    output: CreateSignedOrderLineItemsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

To be implemented in a later release.

Request Actions

The request does not have a request body. The Transact service performs the following actions:

To be implemented in a later release.

Response Syntax

```
Status Code: 200 OK
{
    order: Order
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service.

To be implemented in a later release.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

DeleteOrder

DeleteOrder calls the internal Transact API. This service discards (soft deletes) the Order, meaning that it won't display in any of the Order APIs, but would still exist in the underlying Cart database in Transact. This service equally applies to negotiated or created orders. Note: Transact does not currently support programmatic hard deletes.

Request Syntax

```
DELETE /orders/{orderId}
Content-type: application/smithy

{
    order: Order
}

@idempotent
@http(code: 204, method: "DELETE", URI: "/orders/{orderId}")
operation DeleteOrder {
    input: DeleteOrderInput
    output:DeleteOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ConflictException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required: Order ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Trigger "Discard order" workflow.
- Sample default "Discard order" workflow:
  - Check if the order ID exists. If not, return an error.
  - Check if the order state is "Created" or "Negotiated". If not, return an error.
  - Update order state to "marked for deletion"
- In future stages of development, the above workflow may be modified to include additional capabilities as they are made available via Transact and/or additional steps such as:

- Release inventory holds for this order with the inventory provider.

Response Syntax

```
Status Code: 204 No Content
{
    order: Order
}
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response. The following data is returned in smithy format by the service: updated Order document of order that has been discarded (marked for deletion).

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ConflictException
Updating or deleting a resource can cause an inconsistent state.
HTTP Status Code: 409

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

ListOrders

ListOrders calls the internal Transact API. This service is intended to fetch and display Orders by specific search criteria. The implementation of specific search keywords is still in development.

Request Syntax

```
GET /orders
Content-type: application/smithy

   ?shopperId={shopperId} // optional
   &nextToken={nextToken} // optional, for pagination
   &maxResults={maxResults} // optional, for pagination
   /* Additional filters that may be needed to support later.
     Some (i.e. the date filters) require setting up an internal search engine.
      - brandId
      - storeId
      - clientId
      - associateId
      - afterCreatedDate
      - beforeCreatedDate
      - afterLastUpdatedDate
      - beforeLastUpdatedDate
      - afterNegotiatedDate
      - beforeNegotiatedDate
      - afterSignedDate
      - beforeSignedDate
      - orderState
      - paymentToken
      - productId
   */

@readonly
@http(code: 200, method: "GET", URI: "/orders")
operation ListOrders {
    input: ListOrdersInput
    output: ListOrdersOutput
    errors: [
        ValidationException
        AccessDeniedException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required:
    - Shopper ID
- Optional:
    - MaxResults (Integer) (see AWS API Standards for pagination) (default X; upper bound: Y)
    - Pagination identifier
    - Post Beta: Query Filters (each requires a “filter type”-“filter value” tuple) (one or more more filters may be provided, with filters combining using an AND operator)
        - Creation Datetime Range
        - Last updated Datetime Range
        - Negotiated Datetime Range
        - Signed Datetime Range
        - Order state
        - Payment card identifier
        - ProductID
        - Brand ID
        - Store ID
        - Client ID
        - Associate ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- If the “MaxResults” input is specified, the maximum number of records to return should be that number.
    - If this input exceeds the upper bound that Transact can return at any one time, return an error.
- If the total number of records for a query exceeds the number of records to return, return the first set of records, and a pagination identifier which allows the client to get the next set of matching order IDs for that query.
- Returned orders should be sorted in descending order by Order creation datetime (most recently created orders first).
- If the pagination identifier is included as an input, return the next set of records from the query which provided that pagination identifier.
    - If the pagination identifier is not recognized, return an error.
    - If the pagination identifier does not correspond with the supplied shopper ID, return an error.
    - If other filters are specified in addition to the pagination identifier, return an error.
- Narrow the results to include only orders associated with that shopper ID.
- If no query filters are specified return all orders being stored by a given Transact engine
- If multiple query filters are specified, they should be combined in the query using an “AND” operator.
- If “creation” / “last-updated” / “negotiation” / “signed“ datetime filter is specified, narrow the results to only include orders which have a timestamp for the corresponding field within that time range.
- If order state filter is specified, narrow the results to only return orders that currently match that order state.
- If payment card identifier filter is specified, narrow the results to only return orders that have that payment card identifier as the payment method.

- If the product ID filter is specified, narrow the results to only return orders that have that product ID in at least one line item.
- If “brand ID” / “store ID” / “client ID” / “Associate ID” filter is specified, narrow the results to only return orders that have the specified filter value for the corresponding field.

Response Syntax

```
Status Code: 200 OK
{
    orders: String[] // list of orderIds
    nextToken: String
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following Order data is returned in smithy format by the service.

- List of order IDs for the specified shopper ID.
- Pagination identifier (if applicable)

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException

Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException

Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException

The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

SignOrder

This API service calls the internal Transact API. This service enables a shopper to be able to confirm, or “sign” an Order, allowing a mutual agreement between shopper and retailer, and so fulfill the order and charge the shopper for it.

Request Syntax

```
POST /orders/{orderId}/sign
Content type: application/smithy
{
 order: Order
    constraintViolations: ConstraintViolation[]
  lineItems: LineItem[]
}

@idempotent
@http(code: 200, method: "POST", URI: "/orders/{orderId}/sign")
operation SignOrder {
    input: SignOrderInput
    output: SignOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required: Order ID
- Optional: Payment Transaction Reference ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check if the current datetime is **<=** Negotiated Date + TTL duration. If not, return an error.
- Trigger the Sign order workflow.
- Sample default Sign order workflow:
    - Mark any line items whose fulfillment method is “take-with” as fulfilled.
    - If any line items are not “take-with”, send order to fulfillment provider. If successful, record fulfillment reference in the order. If unsuccessful, return an error.

  - Set order state to “Signed”
  - If there is a Transact Cart ID associated with the order, mark the cart for deletion.
- In future stages of development, the above workflow may be modified to include additional capabilities as they are made available via Transact and/or additional steps such as:
  - Authorize payment card identifier for order total. If this fails, return an error.

Response Structure

```
Status Code: 200 OK
{
   order: Order
   constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service: “Signed” Order document.

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

CancelOrder

This API service calls the internal Transact API. This service cancels an Order before it is in fulfillment. This service should also handle cancelling particular line item(s).

Request Syntax

```
POST /orders/{orderId}/cancel
Content type: application/smithy
{
 order: Order
    constraintViolations: ConstraintViolation[]
  lineItems: LineItem[]
}

@idempotent
@http(code: 200, method: "POST", URI: "/orders/{orderId}/cancel")
operation CancelOrder {
    input: CancelOrderInput
    output: CancelOrderOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required: Order ID

Request Actions

The request does not have a request body. The Transact service performs the following actions:

- Check if the order ID exists. If not, return an error.
- Check if the order status is not “Signed”.
  - If “Created” or“Negotiated”, return an error (e.g., “order not yet Signed”).
  - If “In Fulfillment”, “Completed”, “Cancelled”, or “Reversed”, return an error (e.g., “order cannot be cancelled”)
- Send cancellation request to Fulfillment Provider for order, including all order line items. If provider rejects the request or is unavailable, throw an error.
- If Fulfillment provider confirms cancellation, mark order and all order line items “cancelled”

Response Structure

```
Status Code: 200 OK
{
   order: Order
   constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service: updated Order in cancelled state with cancelled line items

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Catalog APIs

The Transact Catalog APIs offer the services described below. These services are focused on the Retailer Product ID (RPID), which is defined as the **ProductId** in the request/response. This is the unique ID created by a retailer to identify the Product and used to invoke information related to the product from other capabilities (SKU is a comparable attribute). It is the smallest granular level identifier in a catalog. Every RPID is a sellable item to a shopper.

GetProduct

`GetProduct` calls the internal Transact API. It returns a Product object and its status. This service is used to get the product information required for an order. This request can be invoked by the retailer client directly to get all the product info required for displaying the information in the discovery/product detail page and also by the Transact core during ordering to get the updated product information to add to the order document.

Request Syntax

```
GET /products/{productId}"
Content-type: application/smithy

@readonly
@http(code: 200, method: "GET", URI: "/products/{productId}")
operation GetProduct {
    input: GetProductInput
    output: GetProductOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs:

- Required
  - **retailerId** (string) - unique Retailer
  - **productid** (string)- unique Catalog item
- Optional
  - **shopperlocale** (locale) – localization information
  - **catalogId** (string) – unique identifier of specific catalog to search

Request Actions

The request does not have a request body. The Transact service performs the following actions:

1 The retailer client can directly invoke this service. Transact can also call this service during ordering to get the updated product information to add to the order document. This service is used to search the specified catalog and retrieve the product information required for an order.
2 The retrieved product information is displayed in the discovery/product detail page.
3 If the above actions are successful, a catalog item and its associated information is retrieved and ready to be selected and added to the shopper's cart.

Response Structure

```
Status Code: 200 OK
{
    product: Product
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The output is a retrieved Product object with corresponding catalog details. The following data is returned in smithy format by the Transact service.

- Required
  - **Productid** (string)
  - **productTaxCode** (string) - maintained by the retailer's Tax Vendor
  - **unitOfMeasure** (string) – Unit Type for the product item
- Optional
  - **unitType** (enum) – variable depending on the item and locale
  - **catalogId** (string)
  - **productShippingInfo** (structure) – fulfillment information and options
    - **packageDimensionInfo (**structure)
      - **Length** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Width** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Height** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Weight** (structure)
        - **Value** (numeric)

      - **Unit** (string)
    - **packageContentInfo (**structure)
      - **isLithiumBatteryIncluded** (Boolean)
      - **containsFoodOrBeverage** (Boolean)
      - **hasmatClass** (string)
      - **isFragile** (Boolean)
    - **isSignedShippingMethodRequired** (Boolean)
- **ageRestriction** (structure)
  - **isAgeCheckRequired** (Boolean)
  - **minimumAge** (numeric)
- **brand** (string) – attribute of the product item
- **description** (string) – attribute of the product item
- **variant** – attribute of the product item
- **unitPrice** (structure) – **Amount**
  - **value** (numeric)
  - **currencyCode** (string)
- **isPriceTaxInclusive** (Boolean) – attribute of the product item
- **media** (structure)
  - **url** (string)
  - **mimeType** (string)
  - **label** (string)
  - **description** (string)
- **status** (enum) – attribute of the product item
- **attributes** (structure) –
  - **name** (string)
  - **value** (string)
  - **description** (string)
- **lastUpdatedAt** (datetime) – attribute of the product item

Errors

For information about the errors that are common to all actions, see Common Errors.

ValidationException

The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

AccessDeniedException

You do not have the required privileges to perform this action.
HTTP Status Code: 403

ResourceNotFoundException

Request references a resource which does not exist.
HTTP Status Code: 404

InternalServerException

Unexpected error during processing of request.
HTTP Status Code: 500

ThrottlingException

Request was denied due to request throttling.
HTTP Status Code: 429

BatchGetProduct

This service calls the internal Transact APIs. This API service is for retrieving mulitple Products with added information and it gets the status and details of the BatchGetProduct call. This service can be used to get the product information for multiple products in the same request. This will be exposed for the client to directly call the catalog service and by the Transact core to invoke as part of the various order and cart workflows.

Request Syntax

```
POST /products
Content-type: application/smithy

@readonly
@http(code: 200, method: "POST", URI: "/products”
operation BatchGetProduct {
    input: BatchGetProductInput
    output: BatchGetProductOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs:

- Required
    - **retailerId** (string) - unique Retailer
    - **productid** (string)- unique Catalog item
- Optional
    - **shopperlocale** (locale) – localization information
    - **catalogId** (string) – unique identifier of specific catalog to search

Request Actions

The request does not have a request body.   The Transact service performs the following actions:

1 The retailer client can directly invoke this service. Transact can also call this service during ordering to get the updated product information to add to the order document. This service is used to search the specified catalog and retrieve all the product information required for multiple Products in an order
2 Each retrieved product information set is displayed in the discovery/product detail page.
3 If the above actions are successful, a set of catalog items and their associated information is retrieved and ready to be selected and added to the shopper's cart.

Response Structure

```
Status Code: 200 OK
{
    product: Products
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The output is a set of one or more retrieved Product objects with corresponding catalog details. The following data is returned in smithy format by the Transact service for each Product object.

- Required
  - **Productid** (string)
  - **title (string) - product description (label) for product**
  - **productTaxCode** (string) - maintained by the retailer's Tax Vendor
  - **unitOfMeasure** (string) – Unit Type for the product item
- Optional
  - **unitType** (enum) – variable depending on the item and locale
  - **catalogId** (string)
  - **productShippingInfo** (structure) – fulfillment information and options
    - **packageDimensionInfo (**structure)
      - **Length** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Width** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Height** (structure)
        - **Value** (numeric)
        - **Unit** (string)
      - **Weight** (structure)
        - **Value** (numeric)
        - **Unit** (string)
    - **packageContentInfo (**structure)
      - **isLithiumBatteryIncluded** (Boolean)

            - **containsFoodOrBeverage** (Boolean)
            - **hasmatClass** (string)
            - **isFragile** (Boolean)
        - **isSignedShippingMethodRequired** (Boolean)
    - **ageRestriction** (structure)
        - **isAgeCheckRequired** (Boolean)
        - **minimumAge** (numeric)
    - **brand** (string) – attribute of the product item
    - **description** (string) – attribute of the product item
    - **variant** – attribute of the product item
    - **unitPrice** (structure) – **Amount**
        - **value** (numeric)
        - **currencyCode** (string)
    - **isPriceTaxInclusive** (Boolean) – attribute of the product item
    - **media** (structure)
        - **url** (string)
        - **mimeType** (string)
        - **label** (string)
        - **description** (string)
    - **status** (enum) – attribute of the product item
    - **attributes** (structure) –
        - **name** (string)
        - **value** (string)
        - **description** (string)
    - **lastUpdatedAt** (datetime) – attribute of the product item

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException

The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Pricing APIs

The Pricing API provides the pricing information to show in the checkout page and validate if the accurate product price is used during order orchestration. This capability will only provide the product base and sale price. The current way the pricing capability is defined will be able to support line-item level strike through pricing. Retailers can even process tired pricing based on the way they setup their pricelist. Promotions and Loyalty are outside of the pricing capability but they will be a separate capability. Transact handles these capabilities outside of pricing capability, since retailers may want to maintain pricing separate from these capabilities.

GetProductPrices

This API service calls the internal Transact API. This conducts order negotiation, enabling a shopper to create an order so that a retailer and shopper can agree on the terms of a "sales contract" An order must have at least one line item. The number of line items in an order is not constrained by Transact, although chosen capability providers (e.g., promotions, order fulfillment, taxes, etc.) may be limited in the number of line items or units they can support.

This request can be used to get the price of the product across different pricelists that are available. It returns all the pricelist and associated product prices in those pricelists. The retailer's client can directly invoke this request to get the prices across different pricelists for the retailer to decide what they need to display to the shopper.

Request Syntax

```
Get /prices
Content type: application/smithy

{
   Pricing: Pricing
     constraintViolations: ConstraintViolation[]
  lineItems: LineItem[]
}

@readonly
@http(code: 200, method: "GET", URI: "/ prices")
operation GetProductPrices {
    input: GetProductPricesInput
    output: GetProductPricesOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required
  - **retailerId** (string) – used to uniquely identify a retailer
  - **productId** (string) – the retailer product id (RPID) to identify the product and get its data
  - **productIdList** (structure) – a list of RPID strings
- Optional
  - **currencyCode** (string) – represents the base currency to compute prices
  - **shopperId** (string) – uniquely identifies the retailer's shopper

Request Actions

1. The retailer client can directly invoke this service. Transact can also call this service during ordering to get the updated product information to add to the order document. This service is used to search the specified pricelist(s) and retrieve the product prices for the products in an order
2. The retrieved product price is displayed in the discovery/product detail page.
3. If the above actions are successful, a product's price and any associated information is retrieved and added to the discovery/product detail page for selection in the shopper's cart. The displayed price is used in calculating the cart's subtotals and order totals for the shopper.

Response Structure

```
Status Code: 200 OK
{
   pricing: Pricing
   constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the Transact service:

- Required
  - **productPriceDetails** (structure) – product Item with price details
    - **productId** (string) – the retailer product id (RPID) to identify the product and get its data
    - **prices** (structure) – price(s) from the retailer's pricelist(s)
      - **value** (numeric) – price in the currency used for the item in the order

        - **unit** (string) – currencyCode for the item in the order
      - **priceListId** (string) – unique identifier for the pricelist(s) used to obtain shopper's prices
      - **unitBasePrice** (structure) – starting price of a product item
        - **value** (numeric) – price in the currency used for the item in the order
        - **unit** (string) – currencyCode for the item in the order
      - **unitSalePrice** (structure) – offered price of a product item to calculate actual cost
        - **value** (numeric) – price in the currency used for the item in the order
        - **unit** (string) – currencyCode for the item in the order
      - **isPriceTaxInclusive** (Boolean) - Defines if tax is included or excluded as part of the item price
- Optional
  - **catalogId** (string) – identifies the catalog from which the price(s) were retrieved
  - **priceListName** (string) – the name of the pricelist
  - **shopperId** (string) – uniquely identifies the retailer's shopper

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

GetProductPricesForLineItems

This service calls the internal Transact APIs. This API service is for creating/adding new Pricing to line item(s). It gets the status and details of a Pricing object.

This API will be used during ordering by the Transact core to get the updated price for the lineitem product. When processing this request, Transact passes in the right pricelist from which the price information needs to be returned.

Request Syntax

```
POST /prices/lineItems
Content-type: application/smithy

{
    pricing: Pricing
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@http(code: 200, method: "POST", URI: "/ prices/lineItems")
operation GetProductPricesForLineItems {
    input: GetProductPricesForLineItemsInput
    output: GetProductPricesForLineItemsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required
    - **retailerId** (string) – used to uniquely identify a retailer
    - **priceLineItemList** (structure) - List of lineitems for which prices must be retrieved.
    - **productId** (string) – the retailer product id (RPID) to identify the product and get its data
    - **productIdList** (structure) – a list of RPID strings
- Optional
    - **currencyCode** (string) – represents the base currency to compute prices

- **quantity** (structure) - Lineitem quantity within the cart.
  - **value** (numeric) – Amount of the given quantity unit
  - **unitOfMeasure** (string) - quantity unit type
- **catalogId** (string) - - identifies the catalog from which the price(s) were retrieved
- **shopperId** (string) – uniquely identifies the retailer’s shopper

Request Actions

The request does not have a request body. The Transact service performs the following actions:

1 The retailer client can directly invoke this service. Transact can also call this service during ordering to get the updated pricing information to add to the order document. This service is used to search the specified pricelist(s) and retrieve the product prices for the lineitems in an order
2 The retrieved product prices are displayed in the discovery/product details page.
3 If the above actions are successful, the product lineitem prices and any associated information are retrieved and added to the discovery/product detail page for selection in the shopper’s cart. The displayed prices are used in calculating the cart’s subtotals and order totals for the shopper.

Response Syntax

```
Status Code: 200 OK
{
    pricing: Pricing
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service: Pricing object with added line items

- Required
  - **pricingLineItemsDetailsList** (structure) – product lineItems list with price details
    - **productId** (string) – the retailer product id (RPID) to identify the product and get its data
    - **prices** (structure) – price(s) from the retailer’s pricelist(s)
      - **value** (numeric) – price in the currency used for the item in the order
      - **unit** (string) – currencyCode for the item in the order
    - **priceListId** (string) – unique identifier for the pricelist(s) used to obtain shopper’s prices
    - **unitBasePrice** (structure) – starting price of a product item
      - **value** (numeric) – price in the currency used for the item in the order
      - **unit** (string) – currencyCode for the item in the order
    - **unitSalePrice** (structure) – offered price of a product item to calculate actual cost
      - **value** (numeric) – price in the currency used for the item in the order
      - **unit** (string) – currencyCode for the item in the order

        - **isPriceTaxInclusive** (Boolean) - Defines if tax is included or excluded as part of the item price

- Optional
    - **Quantity** (string) - quantity of the item
    - **totalBasePrice** (structure) – Total price of all items without calculating discounts. This is the unitBasePrice multipled by the quantity. This field will only be populated if the quantity is passed in as part of the request.
        - **value** (numeric) – price in the currency used for the item in the order
        - **unit** (string) – currencyCode for the item in the order
    - **catalogId** (string) – identifies the catalog from which the price(s) were retrieved
    - **priceListName** (string) – the name of the pricelist
    - **shopperId** (string) – uniquely identifies the retailer's shopper

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Tax API

The Transact Tax capability calculates the applicable tax information to show to a shopper for both e-commerce and physical store orders. Transact's core ordering workflows focus on tax calculations, but the tax calculations performed during ordering also help drive retailer-owned shopper facing experiences tied to post purchase functions (e.g., receipts, order history, order details).

Also, these same tax calculations are used to support retailer-owned tax reporting and remittance processes, in conjunction with the 3P tax solution providers. Retailers are responsible for establishing contractual relationships with the 3P tax providers/partners, as part of the capability offerings in the Transact AWS marketplace. Retailers utilize the 3P tax solution front-end capabilities to setup/register their accounts and continuously manage the tax data needs ('back office' activities) for running their respective businesses. There is a further intention to 'commit' transactions to each tax solution provider, in an effort to support retailer-owned tax reporting/remittance processes.

GetTaxesForLineItems

This API service calls the internal Transact and is used for updating/adding new line item taxes to an instantiated Order, and adjusting the subtotals and total for an order with the applied taxes. This request covers the core needs for tax calculation, including tax breakdown details for each line item in a transaction. The response details received from this request, via the tax solution provider, will be persisted to the ordering document as part of the Core Transact ordering workflows.

It gets the status and details of a GetTaxesForLineItems object.

Request Syntax

```
POST /taxes/lineItems
Content-type: application/smithy

{
    tax: Tax
    constraintViolations: ConstraintViolation[]
   lineItems: LineItem[]
}

@http(code: 200, method: "POST", URI: "/taxes/lineItems")
operation GetTaxesForLineItems {
    input: GetTaxesForLineItemsInput
    output: GetTaxesForLineItemsOutput
    errors: [
        ValidationException
        AccessDeniedException
        ResourceNotFoundException
        ThrottlingException
        InternalServerException
    ]
}
```

URI Request Parameters

The request uses the following URI parameters as inputs.

- Required
    - **retailerId** (string) – used to uniquely identify a retailer
    - **timestamp** (date/time) – when the tax calculation is done for the order
    - **taxGroup** (structure) - all taxable lineitem elements in the order.
        - **billingAddress** (structure)
            - **line1** – (optional)
            - **line2** – (optional)
            - **postalCode**– (string) – Postal or Zip Code of the Address
            - **city** – (optional)
            - **state** – (optional)
            - **country** (CountryCode) – Country of the Address
        - **shipFromAddress** (structure)’
            - **line1** – (optional)
            - **line2** – (optional)
            - **postalCode**– (string) – Postal or Zip Code of the Address
            - **city** – (optional)
            - **state** – (optional)

        - **country** (CountryCode) – Country of the Address
    - **shipToAddress** (Address structure)
        - **line1** – (optional)
        - **line2** – (optional)
        - **postalCode**– (string) – Postal or Zip Code of the Address
        - **city** – (optional)
        - **state** – (optional)
        - **country** (CountryCode) – Country of the Address
    - **LineItem** (structure) – Taxable Line Items
        - **lineItemID** (string) Unique identifier of line item needing tax calculation
        - **lineitemType** (enum) Category of line item needing tax calculation
            - **PRODUCT**
            - **SHIPPING**
            - ,**SERVICE**
        - **quantity** (optional Quantity) – Quantity of the line item
            - **unitOfMeasure** (Unit of Measure) - quantity unit type
            - **value** (numeric) - amount of the given quantity unit
        - **unitSalePrice** (optional Amount) - Price per unit after promotions
            - **value** (string) - Amount of the given currency
            - **unit** (CurrencyCode) - Currency code
        - **aggregatePrice** (Amount) - Price of line item after promotions applied
            - **value** (string) - Amount of the given currency
            - **unit** (CurrencyCode) - Currency code
        - **isPriceTaxInclusive** (Boolean) - Is tax included or excluded in item price
        - **productTaxCode** (string) - Retailer-specific Product Tax Code (PTC)
        - **productTitle** (optional string) - Product Title

- Optional
    - **shopperLocale** (Locale) - - Data used for localizing response strings.
    - **shopperId** (string) – uniquely identifies the retailer's shopper

Request Actions

The request does not have a request body. The Transact service performs the following actions:

1. The retailer client can directly invoke this service. Transact can also call this service during ordering to get the updated tax information to add to the order document. This service is used to search the specified order line item(s), retrieve the retailer's tax service rates and calculate the tax amounts.
2. The retrieved line items taxes are displayed in the discovery/product details page.
3. If the above actions are successful, the order subtotal(s) and total amounts are recalculated and the entire order is updated accordingly in the discovery/product detail pages for the shopper.
4. When the order is placed, there may be an additional call to the retailer's tax service to report the transaction's tax data as required by the taxing authority's jurisdiction.

Response Syntax

```
Status Code: 200 OK
{
    tax: Tax
    constraintViolations: ConstraintViolation[]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in smithy format by the service: Tax object with updated line items.

- Required
  - **overallTaxAmount** (Amount) - Total tax amount payable as part of the order
    - **value** (string) – Tax Amount of the given currency
    - **unit** (CurrencyCode) - Currency code
  - **detailedTaxLineItems** (DetailedTaxLineItemsList structure) – Line items list with their individual tax details
    - **lineItemID** (LineItem string) - Unique line item identifier requiring a tax calculation
    - **appliedTaxRate** (numeric decimal) - Effective tax rate applied to the line item
    - **lineItemTaxableAmount** (Amount) - Taxable amount for tax calculation
      - **value** (string) – Tax Amount of the given currency
      - **unit** (CurrencyCode) - Currency code
    - **lineItemTaxAmount** (Amount) - Total tax payable for the line item
      - **value** (string) – Tax Amount of the given currency
      - **unit** (CurrencyCode) - Currency code
    - **taxInfo** (taxInfoList structure) - Tax information for the current line item
      - **amountForType** (Amount) - Tax amount for the current tax type
        - **value** (string) – Tax Amount of the given currency
        - **unit** (CurrencyCode) - Currency code
      - **rateForType** (numeric decimal) - Tax rate for the current tax type
      - **name** (string) – Name of the applied tax
      - **classification** (string) – Tax or Fee classification
      - **subclassification** (string) – Subclassification providing detailed tax type information
      - **origin** (enu) - Source of the tax, corresponding to input Location type
        - ORIGIN
        - DESTINATION
      - **jurisdictionName** (string) - Name of the jurisdiction
      - **jurisdictionType** (string) - Jurisdiction type, e.g., city, state, special district
      - **jurisdictionLocation** (Address structure) – Jurisdiction Location information
        - **line1** – (optional)
        - **line2** – (optional)
        - **postalCode**– (string) – Postal or Zip Code of the Address

- **city** – (optional)
- **state** – (optional)
- **country** (CountryCode) – Country of the Address

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException
You do not have the required privileges to perform this action.
HTTP Status Code: 403

InternalServerException
Unexpected error during processing of request.
HTTP Status Code: 500

ResourceNotFoundException
Request references a resource which does not exist.
HTTP Status Code: 404

ThrottlingException
Request was denied due to request throttling.
HTTP Status Code: 429

ValidationException
The input does not satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Data Types

The AWS Transact Gateway Service API contains essential data types that various actions use. This section describes each data type in detail.

Note

The order of each element in a data type structure is not guaranteed. Applications should not assume a particular order.

The following data types are supported:

Cart Data Type

A representation of the items a shopper is considering for purchase. A cart also contains sales context, e.g., date/time, order capture channel, shopper identity (when identified), and store location. At a minimum, a Transact cart contains a Transact-generated Cart ID, Transact engine metadata, datetimes for the cart's creation and last update, and at least one line item. Multiple order negotiations can be initiated from a single cart.

Transact is designed to support cart creation, management and storage, but if required, retailer developers may choose to create, manage, and store carts outside of Transact in the future and pass cart objects into Transact for use in order orchestration. Cart is the primary object used to capture and contain a retail customer's intended order.

Properties

Cart Property	Type	Category	Required?	Comments
CartId	String	Metadata	Required	Transact-generated ID. Must be unique at the Transact Engine leve., Does not need to be shopper-facing.
EngineId	String	Metadata	Required	Transact generated to specify the Transact engine ID and the Transact engine version ID (if applicable).
CreationDateTime	Datetime	Metadata	Required	Transact generated at cart creation.
UpdateDateTime	Datetime	Metadata	Required	Transact generated at last cart update activity.
Labels	String	Metadata	Optional	List of strings that may be provided or updated by a client. Stored in Transact, but not used by Transact. e.g., "Active", "Archived", "Cart", "Wishlist", "John's Cart", "Grocery", etc.
BrandId	String	Client Info	Optional	Client-provided ID specifying the brand associated with the cart. May be used to distinguish between multiple retailer brands that may share a Transact engine (e.g., a customer might have two Brand IDs, e.g. "Saks 5th Avenue", "Saks Off 5th", for two different brands that may use a single Transact engine.)
StoreId	String	Client Info	Optional	Client-provided ID specifying the store associated with the cart. May be used to identify a specific physical

Cart Property	Type	Category	Required?	Comments
				store location "store-XYZ", or to distinguish variations across stores, such as prices for different store locations or online vs offline.
ClientId	String	Client Info	Optional	Client-provided ID specifying the shopper-facing client application creating the cart. May be used to identify a specific application, e.g., "Website" vs "iOS App" vs "Android App", or potentially even the specific in-store POS client device ID.
ShopperId	String	Shopper Info	Optional	Client-provided ID specifying the shopper associated with the cart. May be checked against Shopper Identity provider if workflows are configured to include that check.
SessionId	String	Shopper Info	Optional	Client-provided ID specifying the session associated with the cart. May be used to associate carts with browsing sessions for logged-out shoppers.
Location	String	Shopper Info	Optional	Client-provided DEFAULT address or postal code used to estimate values for taxes & fees, available fulfillment options and charges, etc.
LiteItem	String	Product Info	Yes	Each cart must have at least one line item.
Subtotal	Currency (decmal)	Calculation Info	Required	Transact-generated sum of the costs of the line items in the cart, not including taxes, fees or discounts.
Discounts	Currency (decmal)	Calculation Info	Optional	Transact-generated sum of the discounts associated with the line items in the cart.
Fulfillment Charges	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the costs of fulfillment charges (e.g., shipping fees).
Tax	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the taxes and fees associated of the line items in the cart, including fulfillment charges.
Total	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the costs of the line items in the cart, including discounts, taxes, and fees.
Payment card identifier	String	Payment Info	Optional	Client-provided non-PCI token ID that maps to a payment method stored in a Payment provider.
Payment last-4	String	Payment Info	Optional	Client-provided or provider-supplied. Type of cart & last 4 digits of a card-based payment instrument, for displaying on receipts (e.g. VISA-1234, or AMEX-6789

ShopperId

The AWS Transact Service Model unique shopper identifier to associate the shopper placing the order with a cart.

@httpQuery("shopperId")

$shopperId

Type: String

Length Constraints: Fixed length of 36.

@pattern("^[A-Za-z0-9_-]+$")

`string ShopperId`

Required: Yes'

fulfillerAddress

Address the line item(s) need to be shipped to.

$fulfillerAddress

fulfillerAddress: Address

```
`   structure Address {}
```

Required: Yes'

lineItems

```
structure LineItem {
    id: String
}
list LineItemList {
    member: LineItem
}
lineItems: LineItemList
```

Required: Yes'

s3uri

The S3 URI from which the API service is invoked.

Type: String

Length Constraints: Minimum length of 10.

Pattern: [sS]3://[a-z0-9][a-z0-9.-]{1,61}[a-z0-9]/.+

Order Data Type

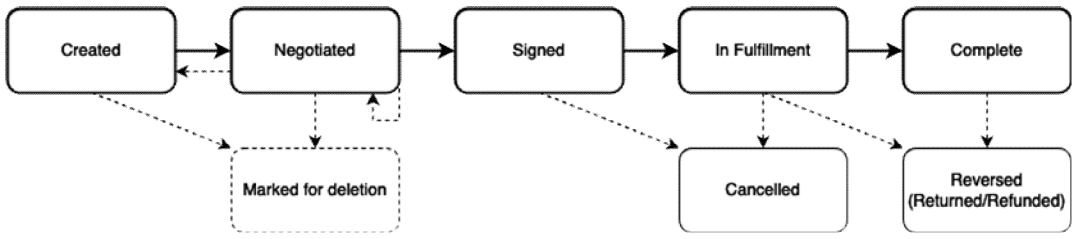
A representation of the items a shopper intends to purchase, the terms associated with how the shopper intends to purchase them, and the details of a retailer's offer to sell the requested items to the shopper according to the terms the shopper has requested ("sales contract"). Like a cart, an order also contains sales context, e.g., date/time, order capture channel, shopper identity (when identified), and store location.

Order is the object used to process a Cart's contents into a customer order for a completed transaction. Orders must be created, managed, and stored in Transact.

Order Workflow

A Transact Order has a specific workflow associated with it, signified by the Order Status. This status is Transact-generated, based on client requests and completion of order workflow steps. Possible status values are:

- **Created** — The order was successfully created, but is not ready to be signed by the client, e.g., if Transact's latest attempt to obtain and validate all necessary information was unsuccessful.
- **Negotiated** — The order is ready to be signed by the client. Transact succeeded its latest attempt to validate and obtain all necessary information for an order to be signed.
- **Marked for Deletion** — The order was discarded by the client or by Transact as it became stale.
- **Signed** — A negotiated order was successfully signed by the client, and will be or has been handed off for fulfillment. Contents of the order are now locked, except for changes in line-item state, order state, and amendments.
- **In Fulfillment** — At least one line item from a Signed order has begun fulfillment.
- **Cancelled** — All line items in the order have been cancelled prior to having been fulfilled.
- **Complete** — All non-cancelled line items in the order have been fulfilled successfully.
- **Reversed** — All non-cancelled line items in the order have been Returned or Refunded.



Properties

Order Property	Type	Category	Required?	Comments
OrderId	String	Metadata	Required	The Transact-generated ID must be unique at the Transact Engine level, but may be more unique (e.g., unique to retailer customer or globally unique). This may be shopper-facing.
CartId	String	Metadata	Optional	Transact-generated ID of the Transact cart from which this order was created.
EngineId	String	Metadata	Required	Transact generated to specify the Transact engine ID and the Transact engine version ID (if applicable).
CreationDateTime	Datetime	Metadata	Required	Transact generated at cart creation.

Order Property	Type	Category	Required?	Comments
UpdateDateTime	Datetime	Metadata	Required	Transact generated at last cart update activity.
NegotiatedDateTime	Datetime	Metadata	Optional	Transact-generated, based on the datetime that the order was last successfully negotiated.
SignedDateTime	Datetime	Metadata	Optional	Transact-generated, based on the datetime that a “Negotiated” order was signed by the client to became a “Signed” order.
Time-to-Live	Datetime	Metadata	Required	Transact-generated, based on the Order TTL configuration for the Transact engine. Stipulates the amount of time from the “Negotiated datetime” within which a “Negotiated” order can be "Signed" by a client.
OrderStatus	String	Metadata	Optional	Transact-generated, based on client requests and completion of order workflow steps. Possible values are: “Created”, “Negotiated”, “Signed”, “Marked For Deletion”, :’Cancele’:, “In-Fulfillment”, “Canceled”, and “Reversed.”.
BrandId	String	Client Info	Optional	Client-provided ID specifying the brand associated with the order. May not be relevant for all retailers, but will be used to distinguish between multiple retailer brands that may share a Transact engine (e.g., a customer like Saks Cloud Services might have two brand IDs, e.g. “Saks 5th Avenue”, “Saks Off 5th”, representing two different brands that may use a single Transact engine.)
StoreId	String	Client Info	Optional	Client-provided ID specifying the store associated with the order. May be used to identify something like “Webstore” a specific physical store location “store-XYZ”.
ClientId	String	Client Info	Optional	Client-provided ID specifying the shopper-facing client application creating the order. May be used to identify a specific application, e.g., “Website” vs “iOS App” vs “Android App”, or potentially even the specific in-store POS client device ID.
Associate ID	String	Client Info	Optional	Client-provided ID specifying the associate that was responsible for a transaction (e.g., for an in-store transaction).
ShopperId	String	Shopper Info	Required	Client-provided ID specifying the shopper associated with the order. May be checked against Shopper Identity provider if workflows are configured to include that check. Guest customer orders also require a shopper ID to complete checkout.
LineItem	String	Product Info	Optional	Each order must have at least one line item. See Line Item Object for the schema.
Promotional Claim Code	String	Discount Info	Optional	Not relevant until after Private Preview. Client-provided promotional claim code to be evaluated by the Promotions provider (i.e., Coupon-code or Discount-code) to determine applicability of certain promotions.
Payment card identifier	String	Payment Info	Optional	Client-provided non-PCI token ID that maps to a payment method stored in a Payment provider. **Required for successful negotiation and signing.**

Order Property	Type	Category	Required?	Comments
Payment transaction reference ID	String	Payment Info	Optional	Client-provided or provider-supplied record of the payment transaction for an order whose payment has been authorized.
Payment last-4	String	Payment Info	Optional	Client-provided or provider-supplied. Last 4 digits of a card-based payment instrument, for displaying on receipts.
Subtotal	Currency (decmal)	Calculation Info	Required	Transact-generated sum of the costs of the line items in the order, not including taxes, fees or discounts.
Discounts	Currency (decmal)	Calculation Info	Optional	Transact-generated sum of the discounts associated with the line items in the cart.
Fulfillment Charges	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the costs of fulfillment charges (e.g., shipping fees).
Tax	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the taxes and fees associated of the line items in the cart, including fulfillment charges.
Total	Currency (decmal)	Calculation Info	Optional	Transact generated sum of the costs of the line items in the order, including discounts, taxes, and fees.

orderId

```
orderId: String
```

Required: Yes

lineItems

```
structure LineItem {
    id: String
}
list LineItemList {
    member: LineItem
}
lineItems: LineItemList
```

Required: Yes

shopperId

@httpQuery("shopperId")

$shopperId

Type: String

Length Constraints: Fixed length of 36.

@pattern("^[A-Za-z0-9_-]+$")

```
string ShopperId
```

Required: Yes'

fulfillerAddress

Address the line item needs to be shipped to.

$fulfillerAddress

fulfillerAddress: Address

` structure Address {}

payments

$payments

```
payments: PaymentList
list PaymentList {
   member: Payment
}
```

[recipients]

$recipients

recipients: RecipientList

```
list RecipientList {
    member: Recipient
}
```

[discountApplications]

$discountApplications

discountApplications: DiscountApplicationList

```
list DiscountApplicationList {
    member: DiscountApplication
}
```

[fulfillmentOptionSelection]

@notProperty

fulfillmentOptionSelection: FulfillmentOptionSelection

```
list FulfillmentOptionSelectionList {
    member: FulfillmentOptionSelection
}
```

[buyingApplication]

@property(name: "buyingApplicationInput")

buyingApplication: BuyingApplicationInput

$fulfillmentOptionSelections

$buyingApplication

LineItem Data Type

A representation of an item that a shopper is considering purchasing or intends to purchase contained within carts and orders. This representation minimally includes an identifier of the product itself, the units and quantities that the shopper intends to purchase, and may include additional information as carts and orders evolve over the course of the shopping journey, including item details, prices, promotional discounts, taxes, fulfillment options, destination addresses, and/or the state of the item's fulfillment or associated reversals. LineItem is the object used to capture the needed information for each item the shopper intends to purchase.

Properties

LineItem Property	Type	Category	Required?	Comments
LineItemId	String	Metadata	Required	Transact-generated ID. It must be unique within the order.
ProductId	String	Product Info	Required	Client-provided ID for the product item. Must uniquely identify a product item within the catalog provider.
Product Name	String	Product Info	Required	May be provider-supplied or provided by the client.
Product Description	String	Product Info	Optional	May be provider-supplied or provided by the client.
Product Thumbnail Media	Structure	Product Info	Optional	May be provider-supplied. Includes: • Media URI • Media Type
Unit of Measure	String	Product Info	Required	Provider-supplied by the Catalog. Unit in which a product is sold. Used to accurately describe and fulfill orders and ensure shoppers pay for and receive the correct quantity of products ordered.
Product Taxability / Fees / SNAP-EBT eligibility	String	Product Info	Optional	May be provider-supplied or provided by the client. Fields used by Payments provider to determine if item may be paid for via SNAP-EBT payment method, or used by Tax provider to determine if taxes and fees should be calculated for the item.
Quantity	Structure	Quantity	Required	Provided as input by the client. Consists of: **Unit** — Specifies the unit used to specify quantity, e.g. "Each", "lbs", "oz", "kg", "g", etc. Validated against Catalog provider to ensure unit provided by client is valid for the product ID. **Amount** — Enumerates the quantity. Validated against the Catalog provider to ensure that the amount is of a valid type, i.e., Integer for "Each", Float for others.
SerialNumber	String	Unit Info	Optional	Stores information about the specific unit of a product being sold, either as part of order creation (in the case of physical store purchases), or as part of order fulfillment (for e-commerce purchases). Supports the ability for retailers to require an identifier to be recorded when certain items, e.g., consumer

LineItem Property	Type	Category	Required?	Comments
				electronics such as smartphones, computers, etc., are purchased, with the specific serial number or IMEI or other unique unit identifier added to the order document. This is to ensure that the shopper receives the exact unit that they purchased, and that they are unable to successfully return a different unit than the one which they purchased.
Price	Structure	Price	Required	Provider-supplied from Catalog capability or Pricing capability, depending on workflow configuration and data availability. If both Catalog price and Pricing capability price is available, the Pricing capability's price is used. This is the price used for cart and order subtotal and total calculations and passed to other capability providers such as Promotions and Taxes). Consists of: **Unit** — Specifies the unit used to specify price per unit, e.g. "Each", "lbs", "oz", "kg", "g", etc. Validated against Catalog provider to ensure unit provided by client is valid for the product ID. **Currency** — Specifies the currency of the price, e.g. "USD", "GBP", etc. **Amount** — Enumerates the price per unit in the currency provided.
PriceSource	String	Price	Optional	Transact-provided field that documents the data source for the price, e.g., "Catalog Base Price", "Pricing Provider", or "Client Override". May include additional properties such as "Client Override Reason
RegularPrice	Structure	Price	Optional	May be provider-supplied (from Catalog capability or Pricing capability, depending on workflow configuration) or provided by client. This is a purely informational property for providing additional context to retailers and/or shoppers, but is not used for checkout or by other capabilities. Consists of: **Unit** — Specifies the unit used to specify price per unit, e.g. "Each", "lbs", "oz", "kg", "g", etc. Validated against Catalog provider to ensure unit provided by client is valid for the product ID. **Currency** — Specifies the currency of the price, e.g., "USD", "GBP", etc. **Amount** — Enumerates the price per unit in the currency provided.
Discount	Structure	Promotions	Optional	May be provider-supplied or provided by client. Array of discount objects for various promotions and discounts. Each discount in the array includes: **Promotion ID** — Associates the discount with a promotion.

LineItem Property	Type	Category	Required?	Comments
				Currency — Specifies the currency of the discount applied, e.g., "USD", "GBP", etc. **Amount** — Enumerates the discount applied (should be negative) to the line item in the currency provided.
InventoryReservationId	String	Inventory	Optional	May be provider-supplied or provided by client. Associates the line item with the inventory if and when it is applicable.
Taxes	Structure	Taxes	Optional	May be provider-supplied or provided by client. Array of objects for various taxes and fees. Optional for a line item to be recorded, but required for successful negotiation and finalizing/signing. Each item in the array consists of: **Tax ID** — Associates the tax or fee with a type of tax or fee (e.g., Sales Tax, VAT, Bottle Deposit, Sweetened Beverage Tax, etc.) **Tax Name** – Human-readable field defining the tax or fee. **Currency** — Specifies the currency of the tax or fee, e.g., "USD", "GBP", etc. **Amount** — Enumerates the tax or fee.
FulfillmentRecipientDetails	Structure	Fulfillment Provider-supplied information about available fulfillment options, and client-provided information about chosen fulfillment method and delivery destination.	Optional	**Required for successful negotiation and signing if fulfillment method requires it, e.g., "Shipping", or "Delivery**". Client-provided or provider-supplied information specifying the recipient and destination of line items to be fulfilled via fulfillment methods that ship directly to a recipient, and which may be used by the Tax provider to evaluate tax charges. May consist of a number of fields, including: **Recipient Name** **Shipping Address** **Phone Number**
FulfillmentOptions	Structure	Fulfillment	Optional	Provider-supplied list of possible fulfillment options for a given line item. Each option in the list should consist of: **Fulfillment option ID** — Fulfillment provider ID for a particular fulfillment option **Fulfillment option name** — Human-readable name for the fulfillment option. **Promise** — Specifies the target fulfillment duration of the fulfillment option. **Charges** — Specifies the fees and taxes associated with the fulfillment option.
PackageInfo	structure	Fulfillment	Required	Provider-supplied by the Catalog. Includes the dimension of the product to be shipped, including, length, width, height and weight. Required for Order Fulfillment capability.
ShippingConstraints	String	Fulfillment	Required	Provider-supplied by the Catalog. Attributes that are used to determine fulfillment

LineItem Property	Type	Category	Required?	Comments
				categorizations which may dictate how the item is handled or fulfilled. Includes: • Contains Lithium Battery • Contains Food or Beverage • Hazmat Class • Is Fragile?
LineItemSubtotal	Currency (decmal)	Calculation Info	Required	Transact calculated subtotal of the line item cost. (Unit price x unit amount.
LineItemTotal	Currency (decmal)	Calculation Info	Required	Transact-calculated total of the line item cost. (Unit price x unit amount + Fulfillment costs + Taxes & Fees + Discounts)

instanceId

The instanceId.

Type: String

Length Constraints: Fixed length of 36.

Pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Required: Yes **jobId**

The jobId.

Type: String

Length Constraints: Fixed length of 36.

Pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Required: Yes **s3uri**

The S3 URI from which the CSV is read.

Type: String

Length Constraints: Minimum length of 10.

Pattern: [sS]3://[a-z0-9][a-z0-9.-]{1,61}[a-z0-9]/.+

Required: Yes **status**

The ConfigurationJobStatus.

BillOfMaterialsImportJob

Type: String

Valid Values: NEW | FAILED | IN_PROGRESS | QUEUED | SUCCESS

Required: Yes **message**

When the has reached a terminal state, there will be a message.

Type: String Required:

No

Product Data Type

A representation of the items a shopper is considering for purchase. A cart also contains sales context, e.g., date/time, order capture channel, shopper identity (when identified), and store location. At a minimum, a Transact cart contains a Transact-generated Cart ID, Transact engine metadata, datetimes for the cart's creation and last update, and at least one line item. Multiple order negotiations can be initiated from a single cart.

Transact is designed to support cart creation, management and storage, but if required, retailer developers may choose to create, manage, and store carts outside of Transact in the future and pass cart objects into Transact for use in order orchestration. Cart is the primary object used to capture and contain a retail customer's intended order.

Properties

```
@documentation("All the product details for a given product")
structure DetailedProductInfo {

  @documentation("A unique ID created by a retailer to identify the product")
  @required
  productId: ProductId

  @documentation("Label/name given to a product")
  @required
  title: MediumString

  @documentation("Tax code defined by the tax vendor for the product")
  @required
  productTaxCode: SmallString

  @documentation("Unit of measure by which the product is being sold.  Refer to the Unit spec from JSR-
  363.")
  @required
  unitOfMeasure: UnitOfMeasure

  @documentation("The Type for the unit of measure")
  unitType: UnitType

  @documentation(
    "A unique ID specific to a product catalog if the retailer has multiple catalogs.
    This value should be provided by retail client when they have multiple catalogs and
    need to specify which catalog the product belongs to."
  )
  catalogId: CatalogId

  @documentation("Product shipping information, which will only be needed by ecommerce solutions")
  productShippingInfo: ProductShippingInfo
  @documentation("Brand/manufacturer name associated with the product")
  brand: SmallString

  @documentation("Product description")
  description: XLargeString

  @documentation("Age restriction for the product.")
  ageRestriction: AgeRestriction

  @documentation("Variant of the product defined by factors like color, size etc.")
  variants: ProductVariantList

  @documentation(
    "Price of the product per unit of measure. This field provides the flat price of a product.
    Retail clients that have different pricing strategy for the same product, e.g. loyalty price, holiday price
etc.;
    should rely on the Pricing capability for more accurate product prices."
  )
  unitPrice: Amount

  @documentation("Defines if tax is included or excluded as part of the item price")
  isPriceTaxInclusive: Boolean

  @documentation("Media information like images/videos associated with the product.")
  media: MediaList

  @documentation("The state of the product in catalog.")
  status: ProductStatus

  @documentation("Open content fields for additional product information.")
  attributes: ProductAttributeList

  @documentation("Last updated product date-time")
  lastUpdatedAt: DateTime
}

@length(max:100)
list DetailedProductInfoList {
  member: DetailedProductInfo
}

@documentation("Media information like images/videos associated with the product.")
structure Media {

  @documentation("Media url")
  @required
  url: XLargeString

  @documentation("Mime type of media- image/png, audio/wav etc. Reference -
https://www.iana.org/assignments/media-types/media-types.xhtml")
  mimeType: SmallString

  @documentation("Label of the media")
  label: SmallString

  @documentation("Description of the media")
  description: XLargeString
}

@length(max: 20)
list MediaList {
  member: Media
}

@documentation("Identifiers of a product")
structure ProductIdentifier {
  @documentation("A unique ID created by a retailer to identify the product")
  @required
  productId: ProductId,

  @documentation(
    "A unique ID specific to a product catalog if the retailer has multiple catalogs.
    This value should be provided by retail client when they have multiple catalogs and
    need to specify which catalog the product belongs to."
  )
  catalogId: CatalogId
}

@length(min: 1, max: 100)
list ProductIdentifierList {
  member: ProductIdentifier
}

@documentation("Variant of the product defined by factors like color, size etc.")
structure ProductVariant {

  @documentation("Retailer catalog ID of the product variant")
  catalogId: CatalogId

  @documentation("Type of the variant - color, size etc.")
  type: MediumString

  @documentation("Retailer product ID of the product variant")
  variantProductId: ProductId
}

@length(max: 200)
list ProductVariantList {
    member: ProductVariant
}

structure ProductAttribute with [Attribute] {}

@length(max: 200)
list ProductAttributeList {
    member: ProductAttribute
}

@documentation("The state of the product in catalog")
enum ProductStatus {
    ACTIVE
    ARCHIVED
    DRAFT
}
```

ShopperId

The AWS Transact Service Model unique shopper identifier to associate the shopper placing the order with a cart.

@httpQuery("shopperId")

$shopperId

Type: String

Length Constraints: Fixed length of 36.

@pattern("^[A-Za-z0-9_-]+$")

```
string ShopperId
```

Required: Yes'

fulfillerAddress

Address the line item(s) need to be shipped to.

$fulfillerAddress

fulfillerAddress: Address

```
`   structure Address {}
```

Required: Yes'

lineItems

```
structure LineItem {
    id: String
}
list LineItemList {
    member: LineItem
}
lineItems: LineItemList
```

Required: Yes’

s3uri

The S3 URI from which the API service is invoked.

Type: String

Length Constraints: Minimum length of 10.

Pattern: [sS]3://[a-z0-9][a-z0-9.-]{1,61}[a-z0-9]/.+

Pricing Data Type

A representation of the items a shopper intends to purchase, the terms associated with how the shopper intends to purchase them, and the details of a retailer's offer to sell the requested items to the shopper according to the terms the shopper has requested ("sales contract"). Like a cart, an order also contains sales context, e.g., date/time, order capture channel, shopper identity (when identified), and store location.

Properties

```
@documentation("All the prices for the given product")
structure ProductPriceDetails {
  @documentation("A unique ID created by a retailer to identify the product")
  @required
    productId: ProductId

  @documentation("A unique ID specific to a product catalog if the retailer has multiple catalogs")
    catalogId: CatalogId

  @documentation("List of different prices offered for the retail product ID across the different pricelist that
the retailer has created")
  @required
    prices: OfferPriceList
}

@documentation("Price offered for the retail product ID in a single pricelist that the retailer has created")
structure OfferPrice {
  @documentation("A retailer can have multiple price list that they maintain to have different pricing based
on different factors (Purchase volume based pricing, seasonal pricing (holiday pricing), country/store front
pricing etc.)")
  @required
    priceListId: PriceListId

  @documentation("Name of the pricelist")
    priceListName: MediumString

  @documentation("Starting price of a single item of the product (based on uom)")
  @required
  unitBasePrice: Amount

  @documentation("Sale price is what Transact or the retailer should be using to portray as the final price for
the product to the customer and in the order")
  @required
    unitSalePrice: Amount

  @documentation("Defines if tax is included or excluded as part of the item price")
  @required
```

```
    isPriceTaxInclusive: Boolean
}

@documentation("Line item with product related details")
structure PricingLineItem {
  @documentation("A unique ID created by a retailer to identify the product")
  @required
    productId: ProductId

  @documentation("A unique ID specific to a product catalog if the retailer has multiple catalogs")
    catalogId: CatalogId

  @documentation("A retailer can have multiple price list that they maintain to have different pricing based
on different factors (Purchase volume based pricing, seasonal pricing (holiday pricing), country/store front
pricing etc.)")
  @required
    priceListId: PriceListId

  @documentation("Quantity of the item")
    quantity: Quantity
}

@documentation("Line item with price related details")
structure PricingLineItemDetails {
  @documentation("A unique ID created by a retailer to identify the product")
  @required
    productId: ProductId

  @documentation("A unique ID specific to a product catalog if the retailer has multiple catalogs")
    catalogId: CatalogId

  @documentation("A retailer can have multiple price list that they maintain to have different pricing based
on different factors (Purchase volume based pricing, seasonal pricing (holiday pricing), country/store front
pricing etc.)")
    priceListId: PriceListId    @required


  @documentation("Quantity of the item")
    quantity: Quantity

  @documentation("Starting price of a single item of the product (based on uom)")
  @required
    unitBasePrice: Amount

  @documentation("Sale price is what Transact or the retailer should be using to portray as the final price for
the product to the customer and in the order")
  @required
```

```
    unitSalePrice: Amount

  @documentation("Total price of all items without calculating discounts")
    totalBasePrice: Amount

  @documentation("Total price of all items after all discounts")
    totalSalePrice: Amount

  @documentation("Defines if tax is included or excluded as part of the item price")
  @required
    isPriceTaxInclusive: Boolean
}

@length(max: 300)
list PricingLineItemDetailsList {
  member: PricingLineItemDetails
}

@length(max: 300)
list PricingLineItemList {
  member: PricingLineItem
}

@length(max: 20)
list OfferPriceList {
  member: OfferPrice
}

@length(max: 300)
list ProductPriceDetailsList {
  member: ProductPriceDetails
}

@pattern("^[a-zA-Z0-9_-]+$")
@length(max: 50)
string PriceListId
```

Tax Data Type

A representation of an item that a shopper is considering purchasing or intends to purchase contained within carts and orders. This representation minimally includes an identifier of the product itself, the units and quantities that the shopper intends to purchase, and may include additional information as carts and orders evolve over the course of the shopping journey, including item details, prices, promotional discounts, taxes, fulfillment options, destination addresses, and/or the state of the item’s fulfillment or associated reversals. LineItem is the object used to capture the needed information for each item the shopper intends to purchase.

Properties

```
structure TaxableLineItem {
  @documentation("Unique identifier for the line item")
  @required
     lineItemId: LineItemId

  @documentation("The category of the line item.")
  @required
     lineItemType: LineItemType

  @documentation("Quantity of the specific line item ( applies only to products )")
     quantity: Quantity

  @documentation("Cost per unit excluding discounts ( applies only to products )")
     unitSalePrice: Amount

  @documentation("Aggregate price of the line item before any promotions applied")
  @required
     aggregatePrice: Amount

  @documentation("The tax code to use for this line item")
  @required
     productTaxCode: SmallString

  @documentation("Description of the line item")
     productDescription: XLargeString

  @documentation("Defines if tax is included or excluded as part of the item price")
  @required
     isPriceTaxInclusive: Boolean
}

structure TaxGroup {
  @documentation("Address to which the billing information is sent")
  @required
     billingAddress: Address
  @documentation("Address where the order is shipped from")
  @required
    shipFromAddress: Address

  @documentation("Address where the order is shipped to")
  @required
    shipToAddress: Address

  @documentation("Collection of all taxable elements in the tax group. Each line item can correspond to
PRODUCT, SHIPPING, SERVICE type etc.")
  @required
    lineItems: TaxableLineItemsList
}

structure DetailedTaxLineItem {
  @documentation("Unique identifier for the line item")
  @required
    lineItemId: LineItemId

  @documentation("The effective tax rate applied to the line item")
    appliedTaxRate: BigDecimalString

  @documentation("Taxable amount on which the taxes are calculated")
  @required
    lineItemTaxableAmount: Amount

  @documentation("Total tax payable for the line item")
  @required
    lineItemTaxAmount: Amount

  @documentation("Collection of tax information for the current line item")
  @required
    taxInfos: TaxInfoList
}

structure TaxInfo {
  @documentation("The tax amount for the current tax type")
  @required
    amountForType: Amount

  @documentation("The tax rate for the current tax type")
    rateForType: BigDecimalString

  @documentation("Name of the applied tax")
  @required
    name: MediumString
  @documentation("Tax or Fee classification")
  @required
    classification: MediumString

  @documentation("Subclassification providing detailed tax type information")
  @required
    subClassification: MediumString

  @documentation("The source of the tax, corresponding to a Location type from input. Can be 'ORIGIN' or
'DESTINATION'")
    origin: TaxOrigin

  @documentation("Name of the jurisdiction")
  @required
    jurisdictionName: MediumString

  @documentation("Jurisdiction type, e.g. city, state, special district")
  @required
    jurisdictionType: JurisdictionType

  @documentation("Location information for the Jurisdiction")
  @required
    jurisdictionLocation: Address
}

@length(max: 300)
list DetailedTaxLineItemsList {
  member: DetailedTaxLineItem
}

@length(max: 20)
list TaxInfoList {
  member: TaxInfo
}

@length(max: 300)
list TaxGroupsList {
  member: TaxGroup
}

@length(max: 300)
list TaxableLineItemsList {
  member: TaxableLineItem
}

enum LineItemType {
  PRODUCT
```

```
   SHIPPING
   SERVICE
}
```

`enum TaxOrigin {`

```
   ORIGIN
   DESTINATION
}
```

Common Parameters

The following list contains the parameters that the Cart and Order actions use.

CartId

```
cartId: String
```

Required: Yes

@required

@nestedProperties

Type: CartDocument

constraintViolations: ConstraintViolationList

```
list ConstraintViolationlisst {
    member: ConstraintViolations
}
structure ConstraintViolation {
    category: String
}
```

Contains business violations in case the order could not be signed.

ShopperId

The AWS Transact Service Model unique shopper identifier to associate the shopper placing the order.

@httpQuery("shopperId")

$shopperId

@pattern("^[A-Za-z0-9_-]+$")

```
string ShopperId
```

Required: Yes'

fulfillerAddress

Address the line item needs to be shipped to.

$fulfillerAddress

fulfillerAddress: Address

```
` structure Address {}
```

lineItems

```
structure LineItem {
    id: String
}
list LineItemList {
    member: LineItem
}
lineItems: LineItemList
```

Required: Yes

@input

structure CreateCartLineItemsInput for Cart with [TransactBaseInput] {

@required
@nestedProperties
cart: CartDocument}

```
 constraintViolations: ConstraintViolationList
}
structure CartDocument for Cart {
```

@required

```
}
```

orderId

```
orderId: String
```

Required: Yes

Order

@nestedProperties

order: OrderDocument

structure OrderDocument for Order {

@required

@input

structure CreateOrderInput for Order with [TransactBaseInput] {

@property(name: "lineItemInputs")

Required: Yes

ListOrders

list: `ListOrders {}`

```
member: Orders
```

Orders

```
orders: OrderIdList
```

The list of found Order object(s).

payments

$payments

```
payments: PaymentList
list PaymentList {
   member: Payment
}
```

recipients

$recipients

recipients: RecipientList

```
list RecipientList {
     member: Recipient
}
```

discountApplications

$discountApplications

discountApplications: DiscountApplicationList

```
list DiscountApplicationList {
     member: DiscountApplication
}
```

```
fulfillmentOptionSelection
   @notProperty
   fulfillmentOptionSelection: FulfillmentOptionSelection
    list FulfillmentOptionSelectionList {
        member: FulfillmentOptionSelection
    }
buyingApplication
   @property(name: "buyingApplicationInput")
   buyingApplication: BuyingApplicationInput
   $fulfillmentOptionSelections
   $buyingApplication

@documentation("Input of BatchGetProduct API")
@input
structure BatchGetProductInput with [CapabilityBaseInput] {
   @documentation("List of product identifiers")
   @required
  productIdentifiers: ProductIdentifierList,

}
@documentation("Error of a product failed to get product detail")
structure BatchGetProductError {
   @documentation("Identifiers of a product")
   @required
  productIdentifier: ProductIdentifier,

   @documentation("Localization information to use to return the product information in a particular
   language")
  shopperLocale: Locale

   @documentation("Error code of a product failed to get product detail")
   @required
   code: String,

   @documentation("Error message of a product failed to get product detail")
   @required
   message: String,
}

@length(max: 100)
list BatchGetProductErrors {
   member: BatchGetProductError
}
```

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.
HTTP Status Code: 403

ExpiredTokenException

The security token included in the request is expired
HTTP Status Code: 403

IncompleteSignature

The request signature does not conform to AWS standards.
HTTP Status Code: 403

InternalFailure

The request processing has failed because of an unknown error, exception or failure.
HTTP Status Code: 500

MalformedHttpRequestException

Problems with the request at the HTTP level, e.g. we can't decompress the body according to the decompression algorithm specified by the content-encoding.
HTTP Status Code: 400

NotAuthorized

You do not have permission to perform this action.
HTTP Status Code: 401

OptInRequired

The AWS access key ID needs a subscription for the service.
HTTP Status Code: 403

RequestAbortedException

Convenient exception that can be used when a request is aborted before a reply is sent back (e.g. client closed connection).
HTTP Status Code: 400

RequestEntityTooLargeException

Problems with the request at the HTTP level. The request entity is too large.
HTTP Status Code: 413

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.
HTTP Status Code: 400

RequestTimeoutException

Problems with the request at the HTTP level. Reading the Request timed out.
HTTP Status Code: 408

ServiceUnavailable

The request has failed due to a temporary failure of the server.
HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.
HTTP Status Code: 400

UnrecognizedClientException

The X.509 certificate or AWS access key ID provided does not exist in our records.
HTTP Status Code: 403

UnknownOperationException

The action or operation requested is invalid. Verify that the action is typed correctly.
HTTP Status Code: 404

ValidationError

The input fails to satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Language-Specific AWS SDKs

For more information about using these APIs in one of the language-specific AWS SDKs, refer to the following links:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3