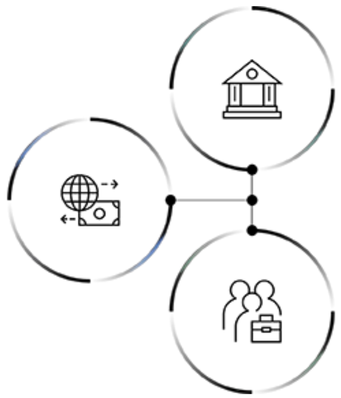
IBM Sterling Order Management

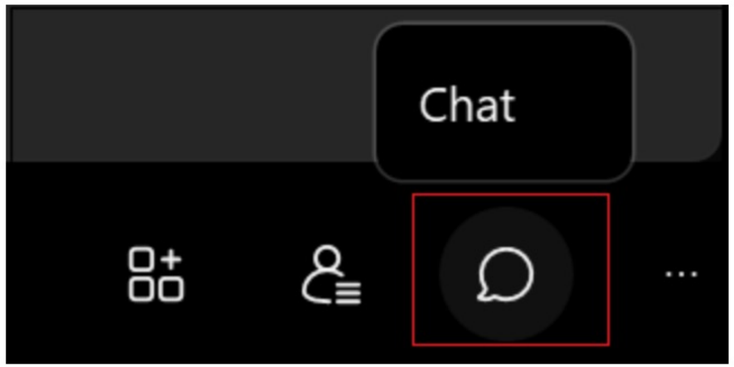
Holiday Readiness 2023



Payments Deep Dive

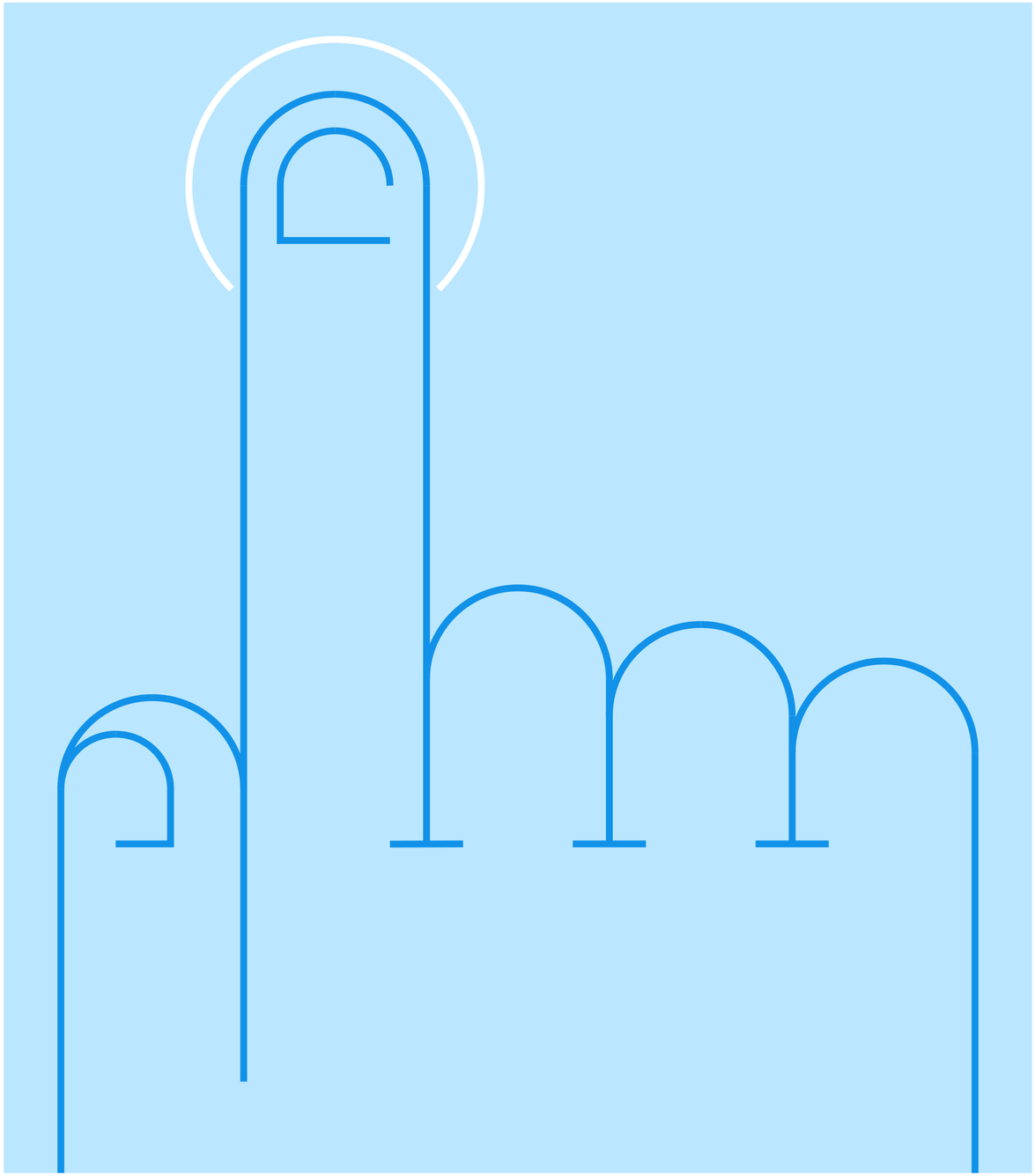
Have a Question(s)?

1. Open the Chat panel from the link in the lower right of the meeting window:



2. In the **To** drop-down list, select the recipient of the message.

3. Enter your message in the chat text box, then press **Enter** on your keyboard.



Your Holiday Readiness Team
… and today's speakers

Chris Burgess
Manager –
Americas & AP Support Experience Team

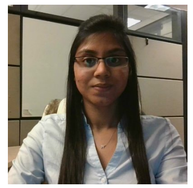
Mike Callaghan
Program Director –
WW Supply Chain Support

Shoeb Bihari
Technical Lead / SRE Advisor –
Order Management Support

Paresh Vinaykya
Executive Technical Account
Manager – Expertise Connect

Jitendra Buge
Technical Support Engineer
Order Management Support

Vijay Gosavi
Development Lead - Payments
Order Management Support



Damini Tacouri
Technical Support Analyst
Order Management Support

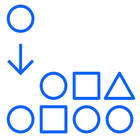
Jyothi Guptha
Customer Engagement Manager –
Order Management Support

IBM OMS Holiday Readiness

Our Mission Statement

Stable Platform

Continuous improvement of platform and monitoring, with focus on performance, stability, reliability

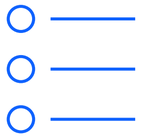


Best Practices

Establish, expand and apply a robust collection of proven self-help best practices focused on peak season success

Proactive Engagement

Early and regular identification, communication, and mitigation of potential risks



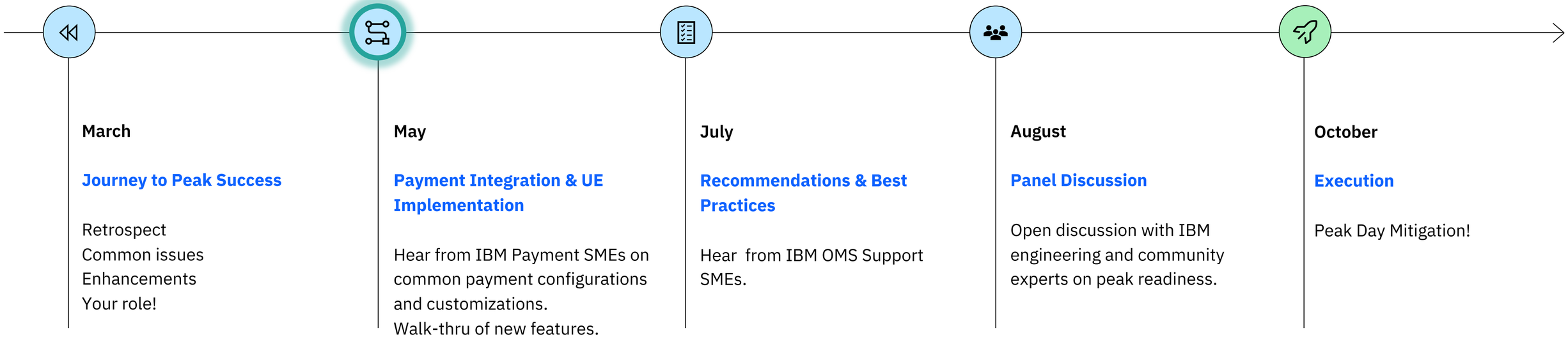
Prescriptive Guidance

Deeper partnership with specific clients in need of direct analysis and prescriptive guidance via our Enhanced Event Readiness offering

The Path Ahead

Journey to Peak Success

The IBM OMS Support team are continuously expanding our technical best practices based on the observations and learnings over our supported launches and peak events!



March

Journey to Peak Success

Retrospect
Common issues
Enhancements
Your role!

May

Payment Integration & UE Implementation

Hear from IBM Payment SMEs on common payment configurations and customizations.
Walk-thru of new features.

July

Recommendations & Best Practices

Hear from IBM OMS Support SMEs.

August

Panel Discussion

Open discussion with IBM engineering and community experts on peak readiness.

October

Execution

Peak Day Mitigation!

Agenda

1. Serviceability & Performance Enhancements
   - Payment Audit & Reasons
2. Dynamic Charge Transaction Request (CTR) Distribution
3. Recommendations & Best Practices
4. Common issues
5. Do’s & Dont’s

What's *New*?

Performance
Transparency
Serviceability

Continuous Improvement

OMS Version: 23.1.2.2 (10.0.2303.2)+

#OMSDemoDays

The IBM Order Management

Continuous improvement into our core platform to promote performance, stability, resiliency, self-service, security

Challenges:

- Infinite Loop
- Multiple open authorizations
- Settlement delay
- Partial cancellation throwing Insufficient funds
- Orders with too many charges transaction records
- Database contention

New Product Features

Enhancement for publishing refund and settlement details

- Payment serviceability enhancement to track payment reason and mapping with additional metadata.
- New event ON_REFUND_OR_SETTLEMENT event

DB managed properties

- Payment reason feature configuration properties maintained in DB as well (i.e., properties will be configurable from SMA).

Enhanced rounding logic

- 592940: `yfs.useNewRoundOffPriceForConventional=Y`
- 593403: Rounding logic is now implemented for sub-total column variables during invoicing in IBM Sterling Call Center.
- 593276: Addressed the 6-digit rounding logic for multi-line orders during invoicing.

Automatic order hold

- Automatic hold is applied via `changeOrder` if looping condition is detected due to payment mismatch.
- `yfs.payment.infiniteLoop.paymentHoldType`
- `yfs.payment.infiniteLoop.allowViewingOfOrder`

On the horizon...

1. Performance improvements in collection **getJobs** query to help with thread contention.
2. Automatic detection and avoidance of thread contention resulting from excessive charge transaction records.

SB

Payment Audit & Reasons

- New enhancement to track payment reason and mapping with additional metadata, providing transparency and serviceability for payment transactions. Read more →
- With this feature enabled, a message providing details of what activities caused it is published
- `ON_REFUND_OR_SETTLEMENT` event has been added to the `PAYMENT_COLLECTION` transaction to publish order and collection details processed by the transaction.
- For typical payment processing, new event will be raised after the second payment collection transaction (i.e., after the second requestCollection API call).

Required configuration

- Enable `yfs.yfs.payment.reason.enablePaymentAudit=Y` property.
- Enable `ON_REFUND_OR_SETTLEMENT` event under `PAYMENT_COLLECTION` transaction.

Read more →

Payment Audits

- Comprehensive payment audit information is logged in payment audit database tables:
  - `YFS_PAYMENT_REASON`
  - `YFS_PAYMENT_REASON_CHARGE`
  - `YFS_PAYMENT_REASON_MAPPING`
  - `YFS_PAYMENT_REASON_ORDLINE`
  - `YFS_PAYMENT_REASON_TAX`

Details

For any changes such as adding a line, changes in price, cancelling, etc. the system will populate the related data along with a reason code to indicate what caused the transaction, here are the sample reason codes:

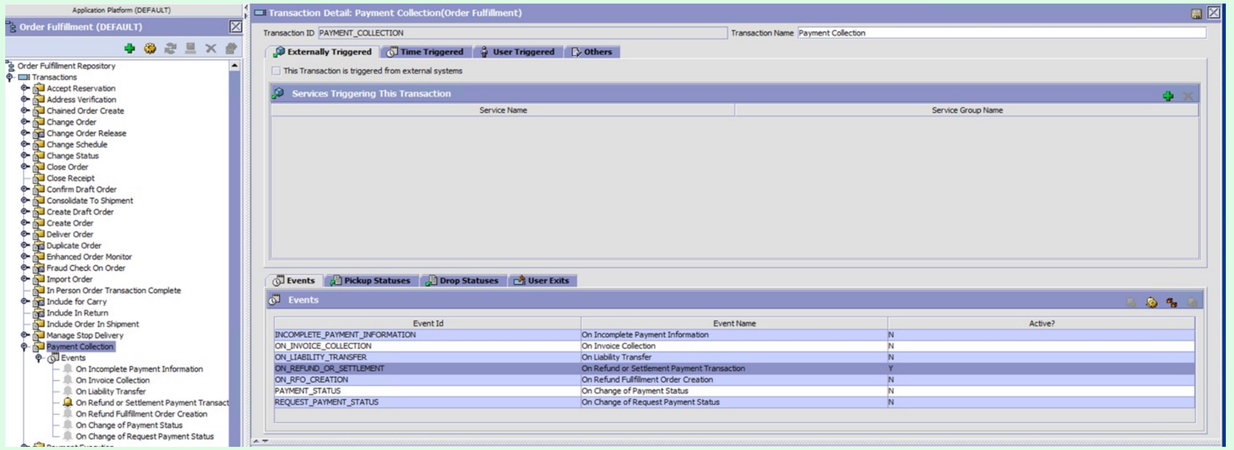
Settlement:

Reason Code	Reason Text
SHIPMENT	Shipment Invoice
ORDER_INVOICE	Order Invoice
ADJUSTMENT	Adjustment Invoice
CHANGE_PRICE	Change Price
ADDITION	Add Order Line
CREATE_ORDER	Create Order Settlement (order has been created and payment agent collects fund)
INSUFFICIENT_CHARGE	Insufficient Charge

Returns:

Reason Code	Reason Text
RETURN	Return Invoice
ADJUSTMENT	Adjustment Invoice
CHANGE_PRICE	Change Price
CANCEL	Cancel Quantity
EXCESS_CHARGE	Excess Charge (order was overcharged and refund is issued)

This feature also allows for customization of what details are included in the published message and whether the message is published.



DT/VG

Scenario 1 - Adding a line to an order

Message published will have the reason code along with details on the new line added.

```
<Order DocumentType="0001" OrderHeaderKey="..." OrderNo="SO1000001" >
```

- One or more payment transaction details.

```
<ChargeTransactionDetailList>
  <ChargeTransactionDetail ChargeType="CHARGE" CreditAmount="52.50" RequestAmount="52.50" Status="CHECKED">
    <PaymentMethod PaymentType="CREDIT_CARD"></PaymentMethod>
  </ChargeTransactionDetail>
</ChargeTransactionDetailList>
```

- One or more reason codes and associated order lines.

```
<ReasonCodeList>
  <ReasonCode Type="SETTLEMENT" Code="ADDITION" ReasonText="Add Order Line">
    <HeaderCharges/></HeaderCharges>
    <HeaderTaxes/></HeaderTaxes>
    <OrderLines>
      <OrderLine OrderLineKey="..." PrimeLineNo="2" SubLineNo="1" ChangedQuantity="" ChangedUnitPrice="" Quanity="1">
        <LinePriceInfo LineTotal="52.50" UnitPrice="50.00"/>
        <LineCharges></LineCharges>
        <LineTaxes>
          <LineTax LineTaxKey="..." ChangedLineTax="" Tax="2.50" TaxName="Tax"></LineTax>
        </LineTaxes>
      </OrderLine>
    </OrderLines>
    </ReasonCode>
</ReasonCodeList>
```

```
</Order>
```

Scenario 2 – Decreasing line quantity

Message published will have the reason code along with details on the line and changed quantity.

```
<Order DocumentType="0001" OrderHeaderKey="..." OrderNo="SO1000001" >
```

- One or more payment transaction details.

```
<ChargeTransactionDetailList>
  <ChargeTransactionDetail ChargeType="CHARGE" CreditAmount= "-105.00 RequestAmount="-105.00" Status="CHECKED">
    <PaymentMethod PaymentType="CREDIT_CARD"></PaymentMethod>
  </ChargeTransactionDetail>
</ChargeTransactionDetailList>
```

- One or more reason codes and associated order lines.

```
<ReasonCodeList>
  <ReasonCode Type="REFUND" Code="CANCEL" ReasonText="Cancel Quantity">
    <HeaderCharges/></HeaderCharges>
    <HeaderTaxes/></HeaderTaxes>
    <OrderLines>
      <OrderLine ChangedQuantity="-1" ChangedUnitPrice="" OrderLineKey="..." PrimeLineNo="1" Quanity="1" SubLineNo="1">
        <LinePriceInfo LineTotal="105.00" UnitPrice="100.00"/>
        <LineCharges></LineCharges>
        <LineTaxes>
         <LineTax ChargeCategory="" ChangedLineTax="-5.00" ChargeName="" Tax="5.00" TaxName="Tax"></LineTax>
        </LineTaxes>
      </OrderLine>
    </OrderLines>
    </ReasonCode>
</ReasonCodeList>
```

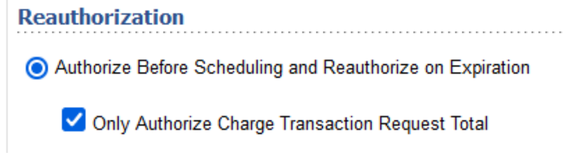
```
</Order>
```

Dynamic Charge Transaction Request (CTR) Distribution

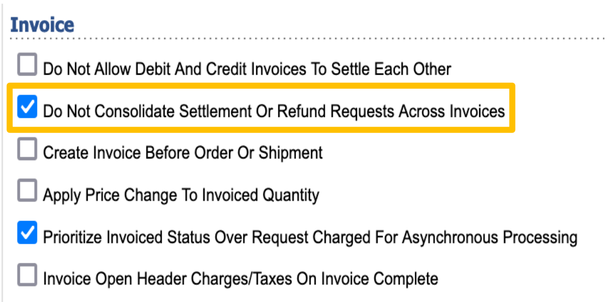
- Designed to get Authorization in phase manner during every release instead of taking the full Authorization on an order at one time in the beginning.
- For each release of a line, authorization should only be created for the corresponding amount of the release (i.e., map one CTR Authorization to one release with one invoice).
- In case of any mismatch with the amount authorized vs amount invoiced, any remaining authorized amount will be considered excess and should ideally be reversed.
- Authorization corresponding to a release is created using `manageChargeTransaction Request` API.
  Read more →

Required configurations

1. Only Authorize Charge Transaction Request Total



2. Do Not Consolidate Settlement Or Refund Requests Across Invoices (if not selected, this is where things can go wrong, and we can see inconsistent and unexpected behavior.



– Disable following flags:

1. Use Same Authorization Multiple Times

*	CHARGE_TRANSACTION_KEY	CHARGE_TYPE	TRANSFER_F...	TRANSFER_TO...	STATUS	CREDIT_AMOUNT	DEBIT_AMOUNT	BOOK_AMOUNT	OPEN_AUTHORIZED_AMOUNT	REQUEST_AMOUNT	DISTRIBUTED_AMOUNT
1	2023042516234580324	CREATE_ORDER	...		CHECKED...	0.00	0.00	52.00	0.00	0.00	0.0
2	2023042517040880761	AUTHORIZATION	...		CHECKED...	0.00	0.00	0.00	32.00	32.00	0.0
3	2023042517114080980	AUTHORIZATION	...		CHECKED...	0.00	0.00	0.00	20.00	20.00	0.0
4	2023042517145680986	SHIPMENT	...		CHECKED...	0.00	32.00	-32.00	0.00	0.00	32.0
5	2023042517155580996	CHARGE	...		CHECKED...	32.00	0.00	0.00	-32.00	32.00	-32.0
6	2023042517201981022	CANCEL	...		CHECKED...	0.00	0.00	-20.00	0.00	0.00	0.0
7	2023042517212381166	AUTHORIZATION	...		CHECKED...	0.00	0.00	0.00	-20.00	-20.00	0.0

Happy Path

1. `createOrder` → Total: $52, No Authorization Created, Line1: $32, Line2: $20
2. `scheduleOrder` → Scheduled
3. `releaseOrder` → Line1: $32
4. `manageChargeTransactionRequest` → Create CTR for corresponding release of $32

```
<ChargeTransactionRequestList OrderHeaderKey="CTR_Order03">
    <ChargeTransactionRequest ChargeTransactionRequestId="1"
     MaxRequestAmount="32.00 " Operation="Manage"/>
</ChargeTransactionRequestList>
```

5. Call Payment APIs to process the charge transaction request Created → Authorized `YFS_CHARGE_TRAN_REQUEST`:

*	CHARGE_TRAN_REQUEST_KEY	ORDER_HEADER_KEY	CHARGE_TRAN_REQUEST_ID	PAYMENT_STATUS
1	2023042516503980641	CTR_Order03	1	AUTHORIZED

6. Ship Line1 (i.e., first line)
7. `releaseOrder` → Line2: $20
8. `manageChargeTransactionRequest` → Create authorization for $20
9. Call Payment APIs to process this authorization.
   Now we have 1 CTR for each release which will later map to one invoice each.
10. `createShipmentInvoice` for release 1 ($32), & call Payment APIs to settle $32 → $32 is `COLLECTED`, $20 is `AUTHORIZED` for release 2.
11. Cancel line 2 (release 2) → `CANCEL` entry created for -$20.
12. Call Payment APIs → `AUTHORIZATION` for $20 is reversed, order is moved to `PAID` status.

DT/VG

Best Practices Payment Integration

These best practices can help solidify y(our) solution, and it can help avoid unforeseen issues in future.

Common issues - 2022

1. Inconsistent authorization reversals with CTR feature enabled.
2. Penny differences causing error infinite loop detected in dynamic distribution best match.
3. Orders having excessive charge transaction records.
4. Contention on YFS_ORDER_HEADER table.

General Best Practices →

Third Party Payment Integrations

Subscribe to a payment solution provider that can capture PAN data in a secure way

Ensure that payment information is captured and submitted to external payment gateway via it's iFrame URL

Sensitive data should never be entered directly into IBM applications before tokenizing them.

Reusing Authorization IDs

Reuse the same authorization by enabling `Use Same Authorization multiple times` option under `financial rules`.

However, when using Dynamic CTR Distribution – get a new authorization for each release. **Do not reuse the same authorization.**

Handling Penny Differences

To avoid penny differences for returns, when a quantity is cancelled, use the return amount calculated by OMS instead of calculating this externally and then passing that amount to OMS.

OMS supports 2-digit decimals.

Consume latest fix pack, contains rounding enhancement

Excessive Charge Transaction Records

Review orders having many charge transaction records.

Having excessive YCT records shows underlying issue.

Place orders having excessive YCT on hold, to prevent further processing.

```
SELECT ORDER_HEADER_KEY, COUNT(*) FROM
OMDB.YFS_CHARGE_TRANSACTION GROUP BY
ORDER_HEADER_KEY HAVING COUNT(*) > 100
ORDER BY ORDER_HEADER_KEY DESC WITH UR;
```

Payment Collection Failure

Ensure the following parameter is set to ensure `PAYMENT_COLLECTION` agent does not fail with `java.lang.IllegalArgumentException: Comparison method violates its general contract!`

Read more →

Authorization Reversal Strategy

We can generate an authorization reversal before the authorization expires using the Reverse Authorization feature.

Reverse authorization immediately in case of any cancellations. This will ensure that the system has authorization corresponding to the correct amount.

For this, enable **Partial Reversal Supported** along with the **Reverse Excess** reversal strategy.

Read more →

Case Study

Problem:

Payment Collection agent's `getJobs` is taking time

Impact

- Delay in payments processing
- Potential impact to order flow, and cascading impact on other components.

Challenge

- Orders were not getting authorized in time due to delayed get jobs of Payment collection agent.
- This was delaying orders being dropped to fulfillment and creation of chained orders were getting affected.
- Cascading impact on other flows of system

Root cause

- Payment Collection `getJobs` query was consuming high CPU and I/O.
- Contention around YFS_ORDER_HEADER table due to large number of charge transaction records
- Oracle DB was doing a full table scan instead of using existing index
- `REORG` was not done in recent time
- `YFS_PERSON_INFO` query was also taking CPU
- Contention around `YFS_INVENTORY_ITEM` and `YFS_RES_POOL_CAPCTY_CONSMPTN` table.

Mitigation

- Place hold on older orders by running `getJobs` query manually
- Put hold on order having large number of charge transaction records
- Make sure JMS session pooling is on and bulk sender properties are set:
- `yfs.agent.bulk.sender.enabled`
- `yfs.agent.bulk.sender.batch.size` → set to 5000
- Increase the number of message to buffer to 50K
- Explicitly call `ProcessOrderPayments` or `requestCollection` APIs to reduce load on agents

Solution

- Use the attribute ConsiderOracleDateTimeAsTimeStamp →
- Regularly perform DB maintenance on YFS_ORDER_HEADER
- JMS Performance properties →
- Validate the index on YFS_PERSON_INFO table →

Recommendation

- Regularly run database maintenance jobs such as REORG and RUNSTAT
- In case of oracle DB, explore the attribute ConsiderOracleDateTimeAsTimeStamp
- Explore the JMS session property and use it as required
- Review the long running SQLs and explore the indexing opportunity
- For item contention, explore the use of hot sku feature →
- JMS Performance properties →
- For YFS_RES_POOL_CAPCTY_CONSMPTN contention, review capacity cache and locking properties →

Case Study

Subject

Infinite Loop Detected in Dynamic Distribution Best Match error

Impact

- Delay in payments processing
- Slowness in the Store

Challenge

- Processing of notifications from external payment gateway was affected.
- Too many errors in system leads to comparatively degraded system performance.
- This also led to contention around `YFS_ORDER_HEADER` table.
- Payment collection agent was holding lock and on hitting `getShipmentList` from webstore, the system would get hang

Root cause

- Could observe it in 4 scenarios for a client
- Change price after order was shipped, invoiced, and paid.
- Rounding issue with split shipment at the time of `createShipmentInvoice`
- Second shipment invoice creation post change price at header level and shipment, invoicing of first line.
- `max_charge_limit` was less than the `total_amount`.

Mitigation

- Updated the authorization expiration date to a future date for orders throwing the infinite loop error
- Put a hold on the problematic orders.
- `Set PAYMENT_HOLD` in agent criteria config field "`HoldTypeOnRollback`" of payment collection agent

Solution

- Refactored `YFSRecalculateHeaderTaxUE` implementation to make sure the header tax amount returned from UE is not less than what was already invoiced to make sure records do not reach to inconsistent state.
- Allow reversing excess authorization
- Make sure that records with the same authorization id should not have different authorization expiration dates.
- Refactored the custom implementation to fix the issue with `max_charge_limit` being less than `total_amount`

Recommendation

- Explore the use of `HoldTypeOnRollback` criteria parameter of collection agent
- Ensure to double check the payment and pricing related UE implementation to ensure it is not leading to inconsistencies.
- Same auth id records should have same auth expiration date

Case Study

Subject

`PaymentCollection` on sales order was not executing automatically

Impact

- Tender posting were delayed, and settlements were not collected, causing direct financial business impact.

Challenge

- Delayed processing of new orders.
- Huge backlog of orders waiting to be processed by the payment collection agent.
- Cascading impact on other flows of system

Root cause

- 10k orders from earlier year were eligible for processing
- Too many different types of errors during agent processing
- Large number of charge transaction records
- `getJobs` call pulling about 5000 records in 1 hr but only 32 orders were getting processed
- Custom issue of authorization expiration dates were not being updated

Mitigation

- Apply hold on old orders from last year
- Apply hold on orders having large charge transaction records
- Increase the number of records to buffer to 25000

Solution

- Addressed the errors from agent processing
- Applied hold on problematic orders
- Custom code was fixed to make sure the authorization expiration date was updated correctly

Recommendation

- Keep a check on old orders with high number of charge transaction records and put them on hold to avoid repeated processing
- Regularly check the agent logs to address the frequent errors
- Temporarily increase the number of records to buffer criteria attribute to clear the backlog

Case Study

Problem

Payment collection agent is slow/hung/ utilizing high CPU

Impact

- Delay in payments processing
- Potential impact to order flow, and cascading impact on other components.

Challenge

- Payment collection agent was not able to process orders in expected pace.
- Database contention having cascading impact on other components such as Order Monitor, Store server, etc.
- Impact to order fulfilment NFRs.

Root cause

- SQL query timeout's while waiting to fetch the Order Header data
- High number of orders with excessive `YFS_CHARGE_TRANSACTION` records

Mitigation

- Apply hold on problematic orders
- Update `AUTHORIZATION_EXPIRATION_DATE` to future date in `YFS_ORDER_HEADER` table

Solution

- Implement query timeout for agent server. Read more →
- Stop payment agent server, apply hold on orders, and start back the payment servers
- In rare scenario, need to run manual SQLs to unblock client by updating the auth expiry date, if manual hold approach not working.

Recommendation

- Keep a check on old orders with high number of charge transaction records and put them on hold to avoid repeated processing
- SQL to identify such orders -
```
SELECT ORDER_HEADER_KEY, COUNT(*) FROM  YFS_CHARGE_TRANSACTION GROUP BY ORDER_HEADER_KEY HAVING COUNT(*) > 500 ORDER BY ORDER_HEADER_KEY
```
- Try to identify pattern which lead to this situation and fix it.

Plan and take necessary action to position y(our) solution for success, it is critical to *TAKE ACTION NOW!*

↓

01
User Exit

- Make sure correct authorization IDs are stamped along with corresponding expiration dates.
- Records having the same authorization IDs should have the same authorization expiration date.
- Handle all the exceptions from the collection UE. Otherwise, charge and authorization transactions will get stuck in the 'invoked' user exit status.
- `RecalculateLineTaxUE` and `RecalculateHeaderTaxUE` output should include all necessary taxes to avoid wiping out previous existing taxes.

02
External Calls

- Periodically review the response time of the external calls to payment system.
- Implement both connect and socket read time to ensure external call does not wait in socket-read indefinitely.
- Long running transaction can lead to DB contention, and resource problem on JMS (MQ Server).
- Unforeseen performance issues and impact to other components.

Do's

- ❑ If Dynamic CTR feature is enabled, use `manageChargeTransaction` API and create separate authorizations for each release/shipment. If single line has multiple releases, then one CTR should be created for each release.
- ❑ Review the javadocs before implementing `processOrderPayments`, and use `RequestCollection, ExecuteCollection, RequestCollection.`
- ❑ Avoid redundant processing of orders by the payment agents. Use the `getJobs` query to verify eligible orders.

Dont's

- ❑ Do not take authorization as part of `createOrder` when Dynamic CTR Distribution is enabled.
- ❑ Do not call `processOrderPayments` as part of long transaction boundary. This API is intended for In-person scenarios e.g., carry lines.

  Note: This API cannot be used with any of the order modification APIs or any APIs that modify orders - either through events, `multiApi` calls or services.

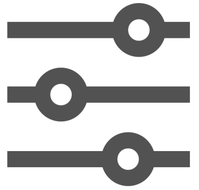
  The `requestCollection()` API will be invoked in a new transaction boundary and with a special condition - each Charge and Authorization request created will have `UserExitStatus` set to `"ONLINE"`. When `requestCollection()` is complete, it will return to `processOrderPayments()` and execute a commit in the new transaction boundary then close it. Thus, even if an error is thrown after this point, the database will not rollback the changes made by `requestCollection()`. Javadoc →
- ❑ Do not `useUnlimitedCharges` on the payment method.

Enhanced Event Readiness Offering

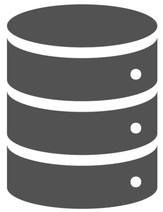
A proactive engagement leveraging a methodical approach to provide targeted, prescriptive guidance toward stability and success on IBM Order Management

Support Backlog Reviews, Prioritization

Best Practice Enablement, Consultation



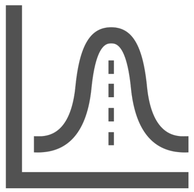
Application Configuration Audit



Database workload review

Application workload review

Production performance review



Peak Projection and Capacity validation

Peak Day Readiness Checklist

SWAT Peak Day Standby

IBM Event Readiness Team

OMS Performance Experts
apply years of proactive preparation and support of worldwide clients for successful go-lives and peak events

- ✓ Identify, mitigate potential risks
- ✓ Align to proven best practices
- ✓ Peak day mitigation techniques

Support Experience Team
prioritize Support workload, augment communication and escalation to help avoid blockers

Expert Labs (optional)
available to perform comprehensive reviews and health checks

***Event Readiness** is modelled as 80-120 hour engagement over 4 months – partnering as you prepare, test, and execute go-live or peak event;*
For November peak, our Engagement must begin no later than September 1, ensuring ample time to proactively review, implement, and validate recommendations

Technical Best Practices

Q&A