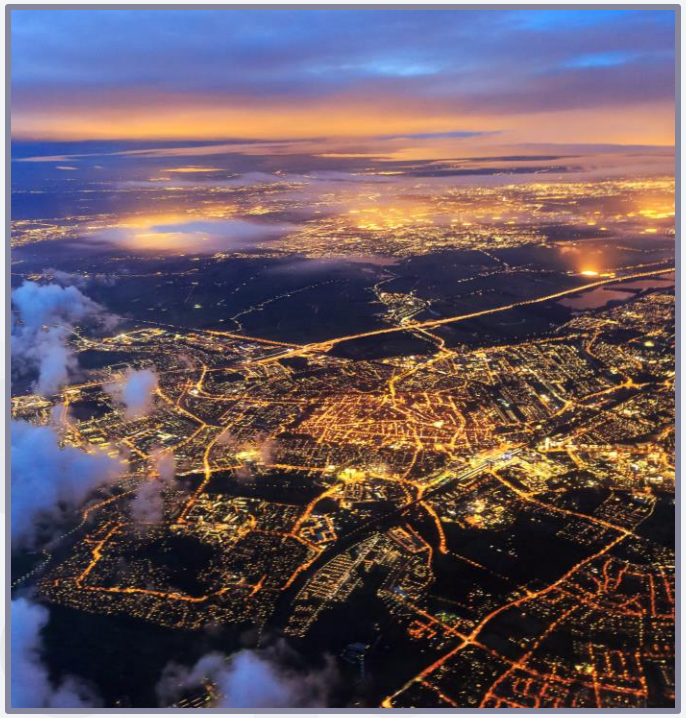
Enabling XML-Based Interoperability in JD Edwards via JDE Orchestration

Most implementation teams assume XML-based integrations require middleware before Orchestrator can consume the data. This session challenges that assumption with a production-proven pattern.

Mo Shujaat · Sandeep Prasad · ERP Suites

Hi 👋🏽 I'm Mo Shujaat!



- VP of Advisory Services @ ERP Suites
- Over 15 years of JDE experience, across a diverse set of companies and industries.
- Distribution consultant by trade, orchestration builder by heart with a burning passion to improve businesses
- Experienced leading multiple JD Edwards implementations, upgrades, and digital transformation projects
- Enjoys building optimization roadmaps with clients and seeing their businesses thrive over the years

ERP Suites in Numbers

20+
years of JDE expertise

300+ Organizations served worldwide

120+ People dedicated to you

500+ Years of JDE experience

We Help Our Customers Realize IT

Cloud Hosting

Public, private and hybrid cloud solutions

Managed Services

ERP system management with first-rate response times

AI and Gen AI

Integrating your business data with emerging technologies

JDE Consulting

Develop, implement, and support JDE applications

EPM Services

EPM consulting, advisory, and managed services

ERP Suites Services

Business Advisory Services

- Project Management
- Technical Strategic Road mapping
- Enterprise Architecture Strategy
- Systems Gap Analysis
- Process Engineering
- Organizational Change Management
- Digital Transformation
- Analytics & Insights Strategy

Functional Consulting Services

- JDE Distribution & Warehousing
- JDE Manufacturing
- JDE Financials
- JDE Human Capital Management
- Managed Services
- UXOne Expertise
- User Defined Objects
- Orchestration Design & Build

Technical and Infrastructure Services

- Technical Refresh
- Technical Upgrades
- Cloud Migrations
- IBM iSeries Administration
- Cloud Administration
- Networking & Server Infrastructure
- Identity Management
- Cybersecurity

The claim

JD Edwards Orchestrator ingests XML natively.

Three design rules, one Groovy step, zero translation tiers — proven on a live QAD Precision (iTRAX) shipment-compliance feed.

0

middleware tiers to license, monitor or explain

3

binding rules that make it work — or fail it silently

1

Groovy step from raw XML to an updated Sales Order

The case: QAD Precision to JD Edwards

What arrives

- **ProcessSPSResponse** XML on every shipment event
- **Split in two:** message details, then the business data
- Compliance result plus shipment references
- Vendor-fixed format — XML is not going away

What JDE needs

- Sales Order status updated in F4211 / F42119
- **Keys live inside** <ShipmentReference>
- Seconds, not an overnight batch
- No drift between shipment reality and order status

`157688SO` **= Sales Order 157688, order type SO.**

The business keys are concatenated inside one element — they have to be parsed, not passed through.

The document, at a glance

```
ProcessSPSResponse  ←  root
 ├─ ApplicationArea
 │    ├─ Sender
 │    │    └─ LogicalId  ERP
 │    ├─ CreationDateTime
 │    └─ ProcessedByServer
 └─ DataArea  ←  the input
      ├─ Process
      ├─ SPSResponse
      │    ├─ OriginalMessage
      │    ├─ TRAXClient  LPI
      │    ├─ Shipper  2800
      │    ├─ ShipmentReference  157688SO
      │    └─ TRAXDespatchNumber
      └─ co:ComplianceResult
           ├─ co:ComplianceStatus  PASS
           └─ co:TestedTime
```

1 Root
ProcessSPSResponse
Orchestration name

2 Wrapper
DataArea
Input, Value Type Object

3 Target
ShipmentReference
157688 + SO, one string

Values from the live payload; client and host details removed.

Two ways to solve it

The usual answer: add a tier

QAD XML

➡

iPaaS / ESB transform

➡

JSON

➡

JDE AIS

- Another licence
- Another runtime to watch
- Weeks per partner
- Two places to debug

What we did: bind it natively

QAD XML

➡

JDE AIS

➡

Orchestration

- No transform tier
- Days per partner
- One log, one team
- One change process

Why almost everyone misses it

1. **JSON-centric documentation**
   Examples, training and blogs all speak JSON, so XML reads as unsupported.

2. **Implicit XML behaviour**
   The binding switches on a header plus a naming convention — documented almost nowhere.

3. **Non-obvious naming dependency**
   Nothing in the AIS UI hints that the orchestration name must be the root element.

4. **Default to middleware thinking**
   ESB habits say 'transform first' rather than 'can the ERP read this?'

```
<ProcessSPSResponse>      ← the name
  <DataArea>              ← the input
    <SPSResponse>
      <ShipmentReference>157688SO</..>
    </SPSResponse>
  </DataArea>
</ProcessSPSResponse>
```

Nothing in the document — or in Studio — signals that the top two tags decide whether the binding works.

What's in a name?

Well — your entire XML interoperability solution.

The three load-bearing rules

1

Orchestration Name = XML Root Element

AIS routes on the root tag: <ProcessSPSResponse> needs an orchestration of the same name. Exact case.

2

Input Name = Encapsulating Element

Value Type: Object. <ShipmentReference> sits inside <DataArea>, so the input is DataArea — the subtree binds to it.

3

HTTP Headers = application/xml

Content-Type and Accept. Accept is the switch that moves the engine off its JSON binding path.

! **Miss any one of the three and nothing complains.** The orchestration runs, returns HTTP 200, and hands the Groovy step null inputs.

Rule 1 — the name is the contract

```
<?xml version='1.0' encoding='UTF-8'?>
<ProcessSPSResponse
     xmlns:co="...compliance"
     lang="en-US" revision="S49">
  <ApplicationArea> ... </ApplicationArea>
  <DataArea> ... </DataArea>
</ProcessSPSResponse>
```

Orchestration name

ProcessSPSResponse

- AIS matches the root element to the name
- Exact match, including case
- Studio never validates or warns
- A mismatch gives null inputs, not an error

The document element is the only routing key AIS has.

Named ProcessSPSResponse

Binds. DataArea arrives populated.

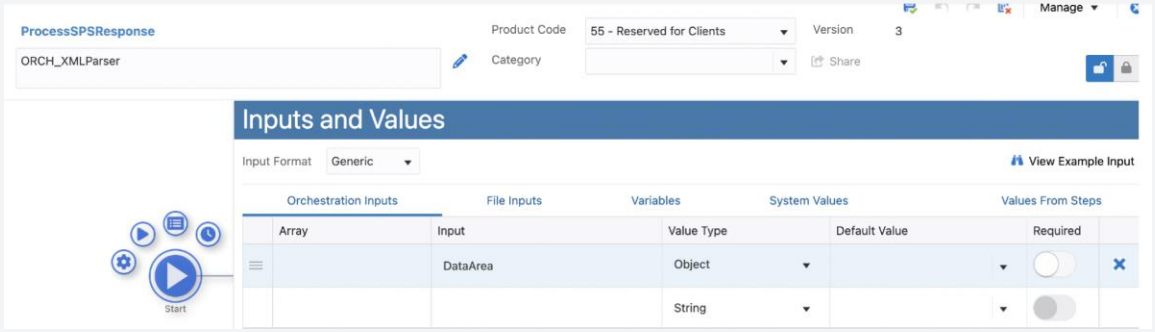
Named anything else

HTTP 200. Null inputs. No error anywhere.

Rule 2 — the input is the element

Name it after the wrapper

- **Input name =** the element that wraps the target data
- **Value Type =** Object — not String, not Number
- The engine binds the entire XML subtree to it
- Rename the input and the inner element changes with it



Orchestrator Studio — Inputs and Values: DataArea declared with Value Type Object.

Groovy then reaches the key as `parsedXml.DataArea.SPSResponse.ShipmentReference`

Rule 3 — the headers throw the switch

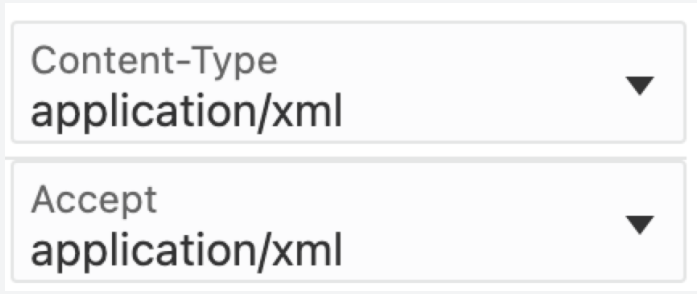
`Content-Type: application/xml`

Tells AIS the body it is receiving is XML. Required on any POST or PUT that carries a payload.

`Accept: application/xml`

Tells AIS to answer in XML — and this is the header that activates the XML binding path behind Rules 1 and 2.

Set Accept to application/json and Rules 1 and 2 stop mattering — the engine goes back to reading JSON property keys.



Both headers set in Orchestrator Studio.

Anatomy of the payload

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessSPSResponse xmlns:co="..." revision="S49">
  <ApplicationArea>
    <Sender><LogicalId>ERP</LogicalId>...</Sender>
    <CreationDateTime>2025-11-17T13:23:20</...>
  </ApplicationArea>
  <DataArea>
    <Process confirm="Always"/>
    <SPSResponse>
      <TRAXClient>LPI</TRAXClient>
      <Shipper>2800</Shipper>
      <ShipmentReference>157688SO</...>
    </SPSResponse>
    <co:ComplianceResult>
      <co:SourceReferenceKey2>157688SO</...>
      <co:ComplianceStatus>PASS</co:ComplianceStatus>
    </co:ComplianceResult>
  </DataArea>
</ProcessSPSResponse>
```

Root element
Fixes the orchestration name.

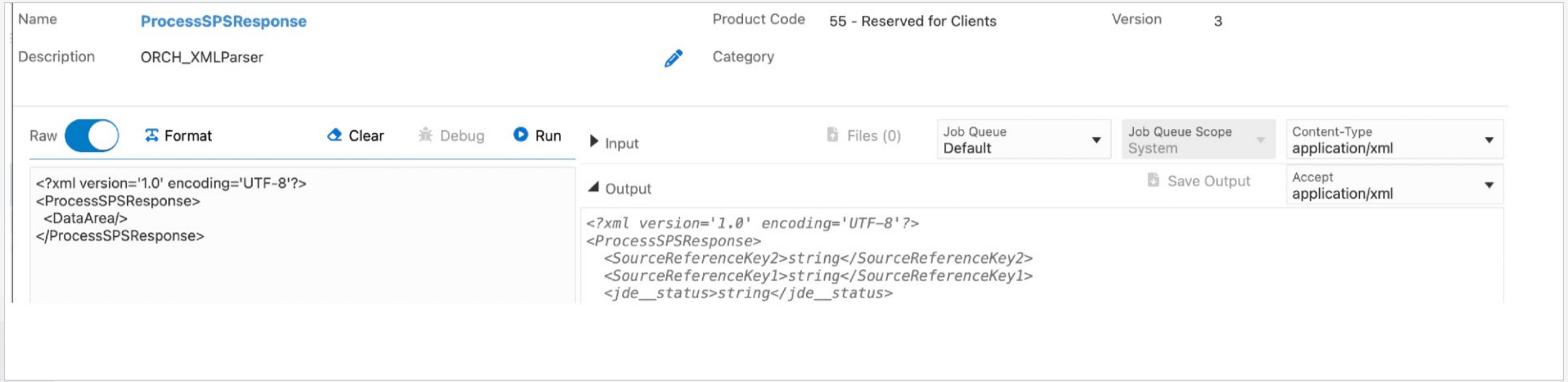
ApplicationArea
Who sent it, when, and with which message ID. No business data.

DataArea
The wrapper, so this becomes the input, typed Object.

ShipmentReference
The business key: 157688 + SO — order number and type in one string.

co: namespace
Prefixed elements need '*:ComplianceResult' or a namespace-aware parser.

The Orchestrator advertises its own contract



Orchestrator Studio, Accept = application/xml: the request and response templates are generated from the orchestration name and its declared inputs.

Root = orchestration name
The Studio writes <ProcessSPSResponse> because that is the orchestration name.

Inner element = declared input
<DataArea/> appears because an input named DataArea exists, typed Object.

Both headers = application/xml
Switch Accept to JSON and the templates come back JSON-shaped.

The mapping, on one slide

XML structure	Orchestration artifact	How Groovy reaches it
`<ProcessSPSResponse> (root)`	Orchestration name = ProcessSPSResponse	–
`<DataArea> (wrapper)`	Input DataArea, Value Type Object	`inputMap.get("DataArea")`
`<SPSResponse><ShipmentReference>`	—	`parsedXml.DataArea.SPSResponse .ShipmentReference.text()`

Take a photo of this one. Every inbound XML integration you build afterwards is this table with different element names.

One Groovy step does the parsing

```
import groovy.xml.XmlSlurper
import com.oracle.e1.common.OrchestrationAttributes

HashMap<String, Object> main(
        OrchestrationAttributes orchAttr, HashMap inputMap) {
  HashMap<String, Object> returnMap = new HashMap<>();
  String xmlStringVal = (String) inputMap.get("xmlInput");
  if (xmlStringVal != null && !xmlStringVal.trim().isEmpty()) {
    try {
      def parsedXml = new XmlSlurper().parseText(xmlStringVal);
      String shipmentReferenceVal =
        parsedXml.DataArea.SPSResponse.ShipmentReference.text();
      orchAttr.writeDebug("Extracted: " + shipmentReferenceVal);
      returnMap.put("ShipmentReference", shipmentReferenceVal);
    } catch (Exception e) {
      orchAttr.writeWarn("Failed to parse XML: " + e.getMessage());
    }
  } else {
    orchAttr.writeWarn("XML input string was null or empty.");
  }
  return returnMap;
}
```

Four things to get right

- **`groovy.xml`**, not groovy.util — XmlSlurper moved package in Groovy 4+, which recent JDE Tools releases ship.
- **JsonSlurper** is unchanged; only the XML classes relocated.
- **Null-check and try/catch** — writeDebug and writeWarn land in the AIS orchestration log.
- **Namespaces**: reach prefixed nodes as '*:ComplianceResult' or configure a namespace-aware parser.

Two ways to take the payload in

Object binding

Recommended

- **Input** DataArea, Value Type Object
- Engine parses; the subtree arrives structured
- Downstream steps navigate without re-parsing
- This is the path the three rules activate

Raw string passthrough

Useful, deliberately

- **String input, commonly** xmlInput
- Groovy builds its own XmlSlurper and parses the text
- Full parsing flexibility for variable schemas
- Good for diagnostic orchestrations

Pick one mode per orchestration and stay with it — mixing them is how the next maintainer loses an afternoon.

End to end, in seven steps

1

QAD POSTs
ProcessSPSResponse XML, Content-Type application/xml

2

AIS routes
Root element picks the orchestration

3

Input binds
DataArea subtree arrives as an Object

4

Groovy parses
XmlSlurper walks to ShipmentReference

5

Keys derived
Order 157688, type SO

6

JDE updates
Form Service writes F4211 / F42119

7

QAD acknowledged
Structured XML response returned

Every step lives inside JDE — one log, one change process.

What it is worth

Dimension	With middleware	Native Orchestrator
Integration cost (licence + ops)	Baseline	20–40% lower — the tier disappears
Partner onboarding	Weeks of bespoke build	Days, template-driven
End-to-end status latency	Hours, batch-style	Seconds, real time
Failure surface	Multiple components	One platform: JDE only

Indicative ranges, not audited metrics — validate against your own baseline before quoting them externally.

Plus: every payload, parse and update is in the AIS log with correlation IDs.

When it breaks, look here

Symptom	Where to look
Compile error: cannot resolve groovy.util.XmlSlurper	Import groovy.xml.XmlSlurper — package moved in Groovy 4+
Input is null although valid XML was posted	Check all three rules: name vs root, input vs wrapper, Accept header
Namespaced elements come back empty	Use '*:ComplianceResult' or a namespace-aware parser
HTTP 200 but the Sales Order never changes	Add writeDebug before the Form Service step and read the AIS log

Security is not optional: token auth or mutual TLS on the AIS endpoint, plus an upstream IP allow-list.

A framework, not a one-off

Naming convention template

Name = root element. Input = wrapper. A checklist, not a memory test.

Input binding pattern

Wrapper exposed as Value Type Object, the same way every time.

Groovy parser template

groovy.xml.XmlSlurper boilerplate with null checks and logging hooks.

HTTP contract checklist

Content-Type and Accept verified on both sides before go-live.

Applying it to the next partner

1. Root element → orchestration name
2. Wrapper element → input, typed Object
3. Content-Type and Accept, both sides
4. Drop in the parser, adjust the GPath
5. Chain Form or Data Service requests

Take these three things home

1. Name = root element
2. Input = wrapper, typed Object
3. Both headers = application/xml

And three things to do next

- **Adopt it as the standard —** Make the pattern the default for every inbound XML feed, instead of a middleware ticket.
- **Publish the templates —** Orchestration skeleton, Groovy boilerplate, test harness — so nobody rediscovers the rules.
- **Migrate by volume —** Start with the highest-volume feed currently routed through a translation tier.

XML stops being a blocker the moment the three rules are non-negotiable.

erp suites

Rapid Orchestrations

Quick Automation Bundle

Choose 3 automations from $15,000

Order2Cash Automation

Get Hold Code Info

Quickly retrieve and display sales order hold code information and reason without navigating multiple inquiry screens in JDE.

Sales Order Hold Code Release

Release sales orders from hold with a single, saving time and avoiding delays in order processing and fulfillment.

Sync Sales Order & Work Order Date

Automatically sync promised ship dates and work order start dates to ensure production and sales commitments are aligned.

Send RMA Shipping Label to Customer

Generate and email return shipping labels to customers automatically based on RMA data in JDE.

1 Click Print/Reprint Invoice

With a single click, print or reprint sales invoices from JDE. Ideal for AR teams looking to reduce time navigating menus.

Save over 10 clicks per document

1 Click Print/Reprint Pick Ticket

One-click solution for printing or reprinting pick tickets, reducing delays and avoiding errors from manual navigation.

1 Click Print/Reprint Packing Slip

Automatically print or reprint packing slips directly from sales orders or shipments, speeding up logistics workflows.

1 Click Print/Reprint Order Ack

Streamline print of order acknowledgments from sales orders with one click, useful in customer service or order desk roles.

1 Click Quote to Order Release

Convert quotes to orders with a single click by triggering the order release process, avoiding multiple screen navigation.

1 Click Blanket Order Release

Release blanket orders with a single click by automating the release process, saving time for purchasing teams.

Quick Automation Bundle

Choose 3 automations from $15,000

Master Data Automation

Copy item from existing item

Automatically create a new item in JDE by copying data from an existing item, including UOMs, category codes, branch plant info, and cost records. Greatly reduces manual entry and ensures data consistency.

Save over 50 clicks!

Create new item via CSV

Bulk-create new item master records in JDE by uploading a CSV file, eliminating repetitive keystrokes and the need to navigate multiple entry screens.

Create new JE via CSV

Upload journal entries in bulk via CSV with orchestrations, speeding up period-end close cycles and reducing data entry errors.

Faster closing!

Email Batch Approval

Enable email-based batch approvals for journal entries, vouchers, or purchase orders; streamlining finance workflows and supporting remote approvals.

New Supplier Workflow

Guide users through a structured supplier onboarding process that collects, validates, and inputs all required vendor data into JDE automatically.

New Customer Workflow

Simplify the customer onboarding process by using orchestrations to capture and validate new customer data, eliminating manual form entry.

Fetch Automated Exchange Rates

Automatically retrieve and update exchange rates from a trusted source (e.g., Xignite or ECB) into JDE, removing the need for daily manual input.

Quick Automation Bundle

Choose 3 automations from $15,000

Procure2Pay Automation

1 Click Print/Reprint PO

Enable one-click printing of purchase orders or reprints, improving procurement response times and reducing process delays.

Save over 10 clicks!

1 Click Quote to Order Release

Trigger quote-to-order release from procurement quotes with one click, reducing entry steps and streamlining buying decisions.

Save over 10 clicks!

1 Click PO Approval, Req Approval

Trigger both PO and requisition approvals with a single click, reducing delays and ensuring compliance with approval workflows.

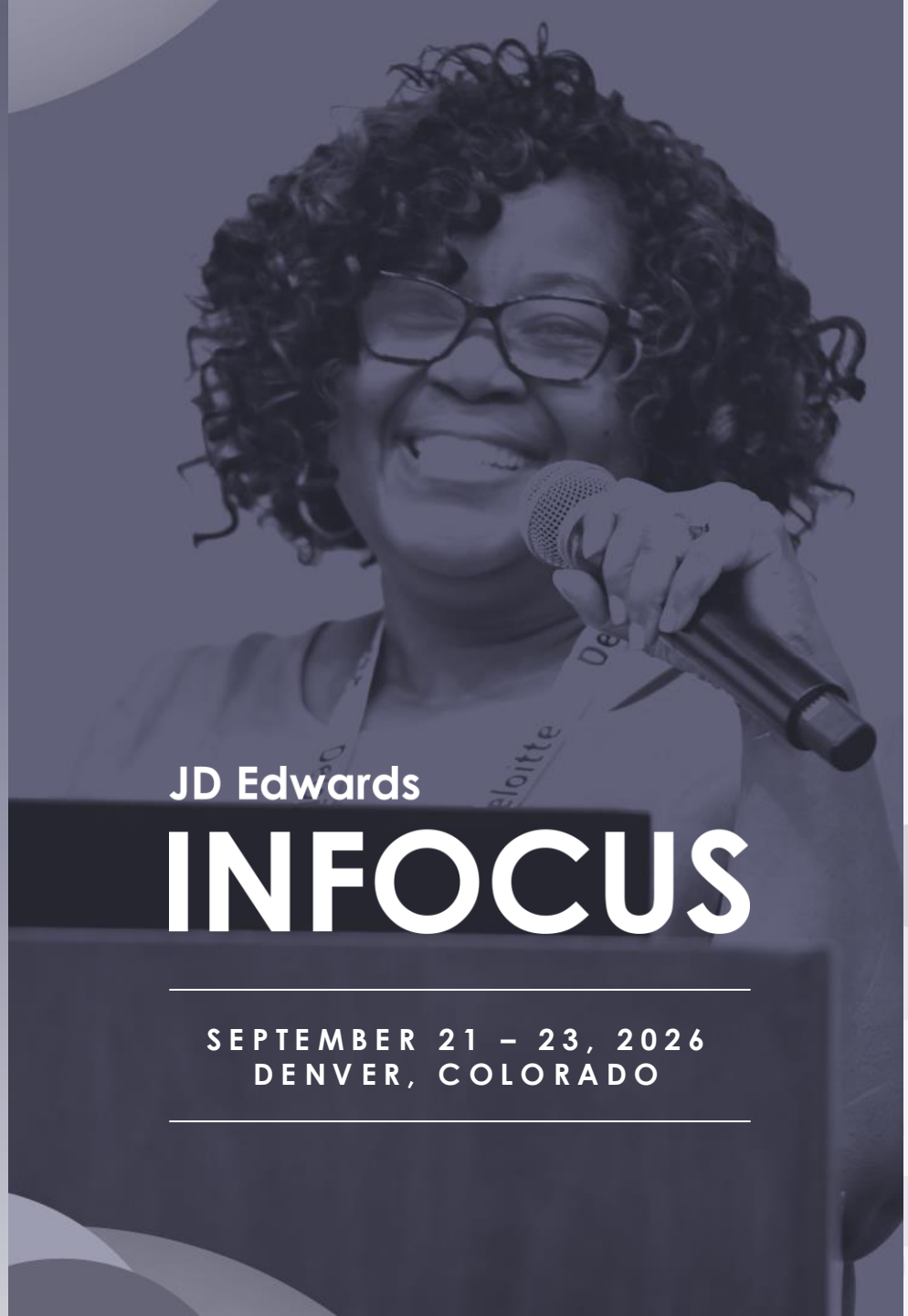
1 Click Blanket Order Release

Enable quick release of blanket purchase orders, reducing delay and simplifying procurement tasks for planned buys.

1 Click Req to PO Conversion

Automatically convert requisitions to purchase orders with a single click, simplifying the buying process and reducing turnaround time.

Save over 15 clicks per release



Questions?

Contact me:

Mo Shujaat
ERP Suites -- mshujaat@erpsuites.com

Sandeep Prasad
ERP Suites – sprasad@erpsuites.com

Session ID:

P-052976- Enabling XML-Based Interoperability in JD Edwards via JDE Orchestration

Quest
JD Edwards
Community

LOVE THIS SESSION?

COMPLETE AT LEAST **3 SESSION SURVEYS** PER DAY AND GET ENTERED INTO THE DAILY DRAWINGS TO WIN A **$500** GIFT CARD!