Industry Solutions
IBM Sterling Order Management

IBM Sterling Payment Processing Overview

Bappaditya Sinha

08/23/2012

Goals of Presentation

- High level understanding of payment processing
- Different ways of achieving payment processing

Overview

Following are some of the features are available in payment processing -

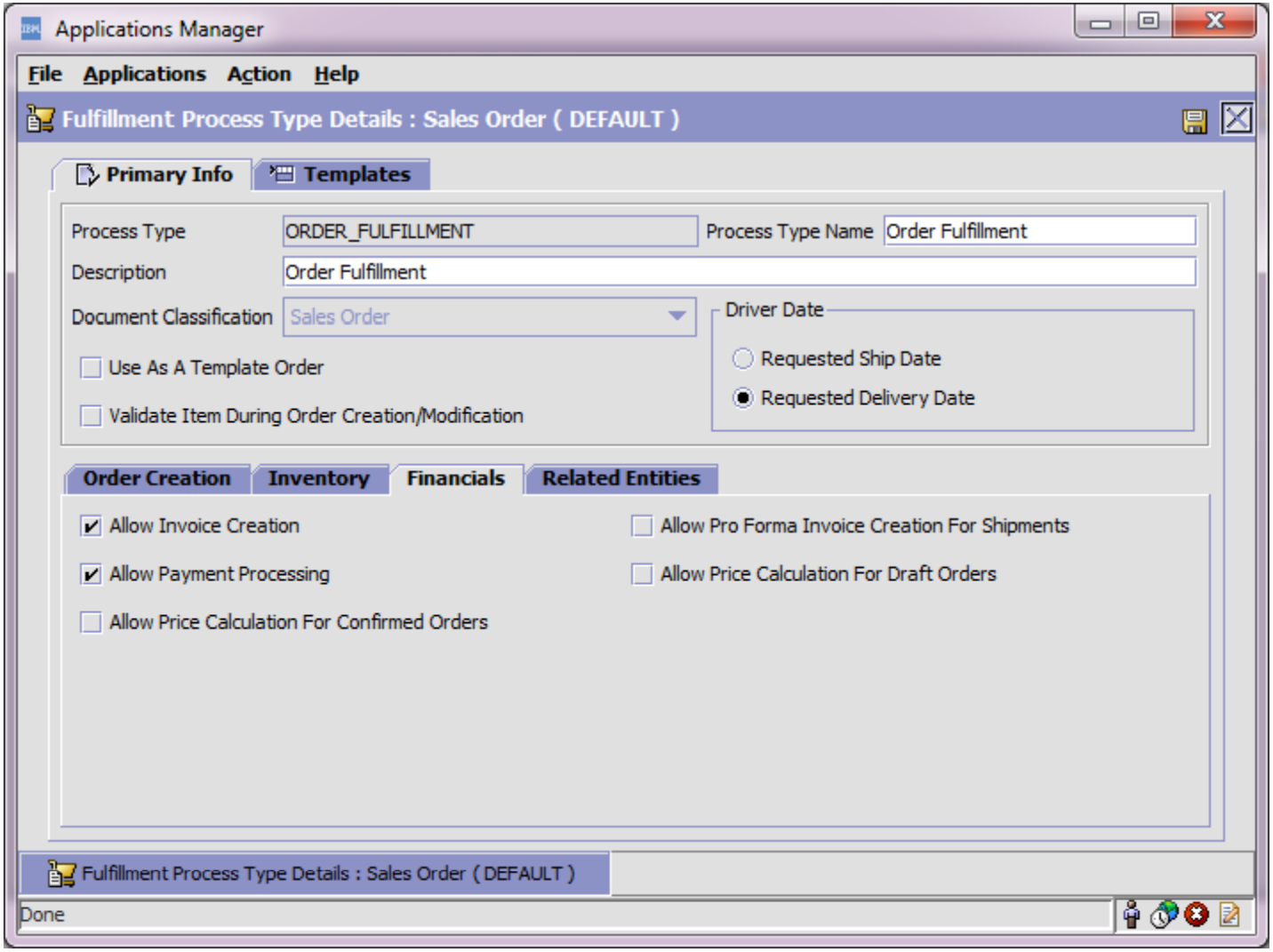
- Payment configuration with multiple levels of control
- Multiple payment methods
- Pre-Authorization and Pre-Charging of multiple payments enabled
- Charge Consolidation
- Handling Credit/Debit memos and manual authorizations/charges
- Pre-Settlement
- Async Processing
- Returns Payment Processing
- Tokenization of Payment Account Numbers (via SSDCS)

Payment Configurations

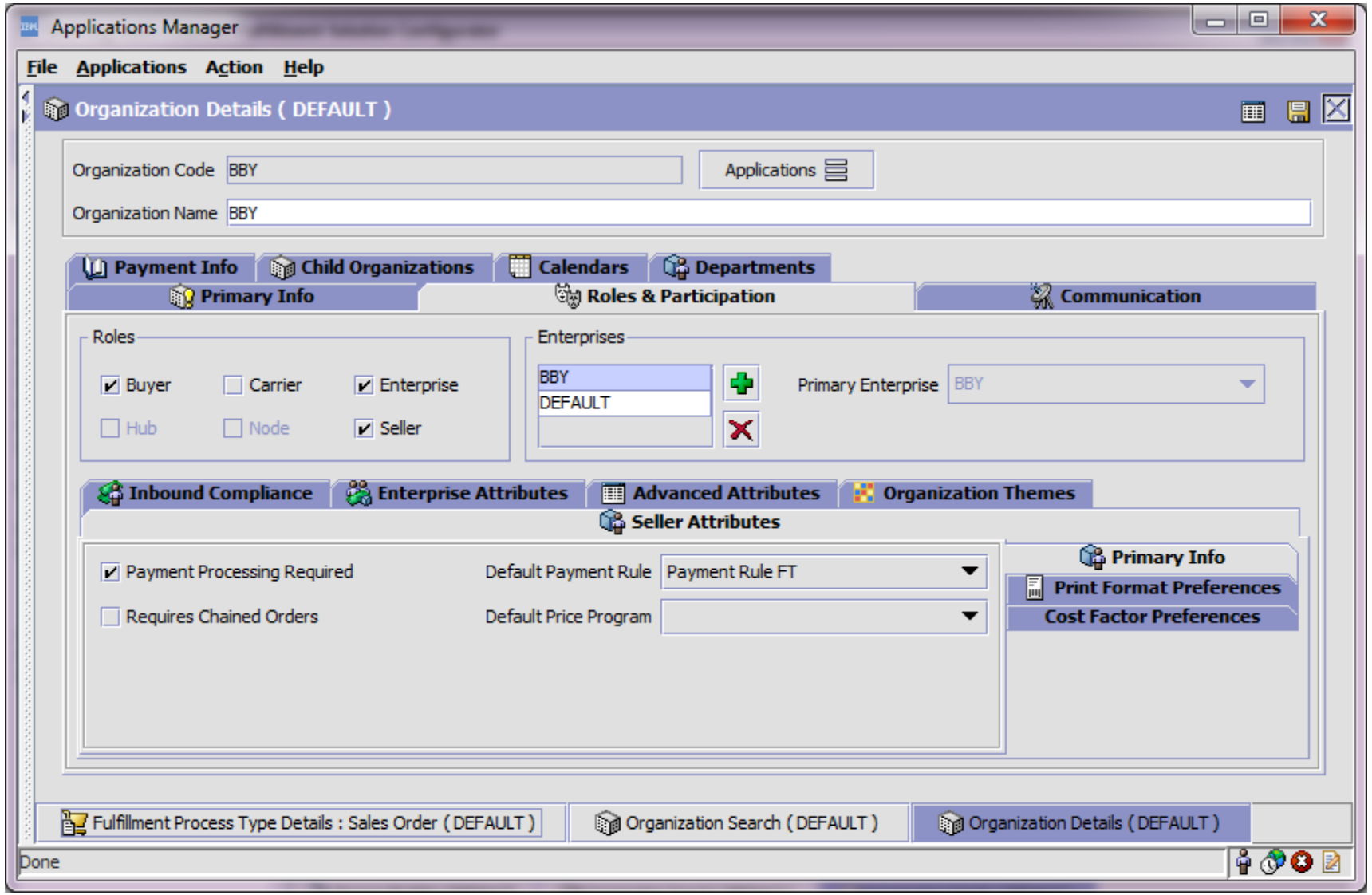
Multiple Levels of Controls:

- **Can be determined at Document type**
  - For Purchase Orders (document type = 0005) set the “Allow Payment Processing” and “Allow Invoicing” to “N”.
  - YFS_DOCUMENT_PARAMS.DOCUMENT_PARAMS (XML)
- **Can be determined at Organization level**
  - Payment Processing Required flag = “Y/N” for Seller Organization
  - Stored in YFS_ORGANIZATION.PAYMENT_PROCESSING_REQD
  - Payment is not processed if this value is N
  - NOT_APPLICABLE (payment status) if this value is N
- **Document type permission setting overrides the seller-level control setting**

Payment Settings At Document Type



Payment Settings At Organization Level

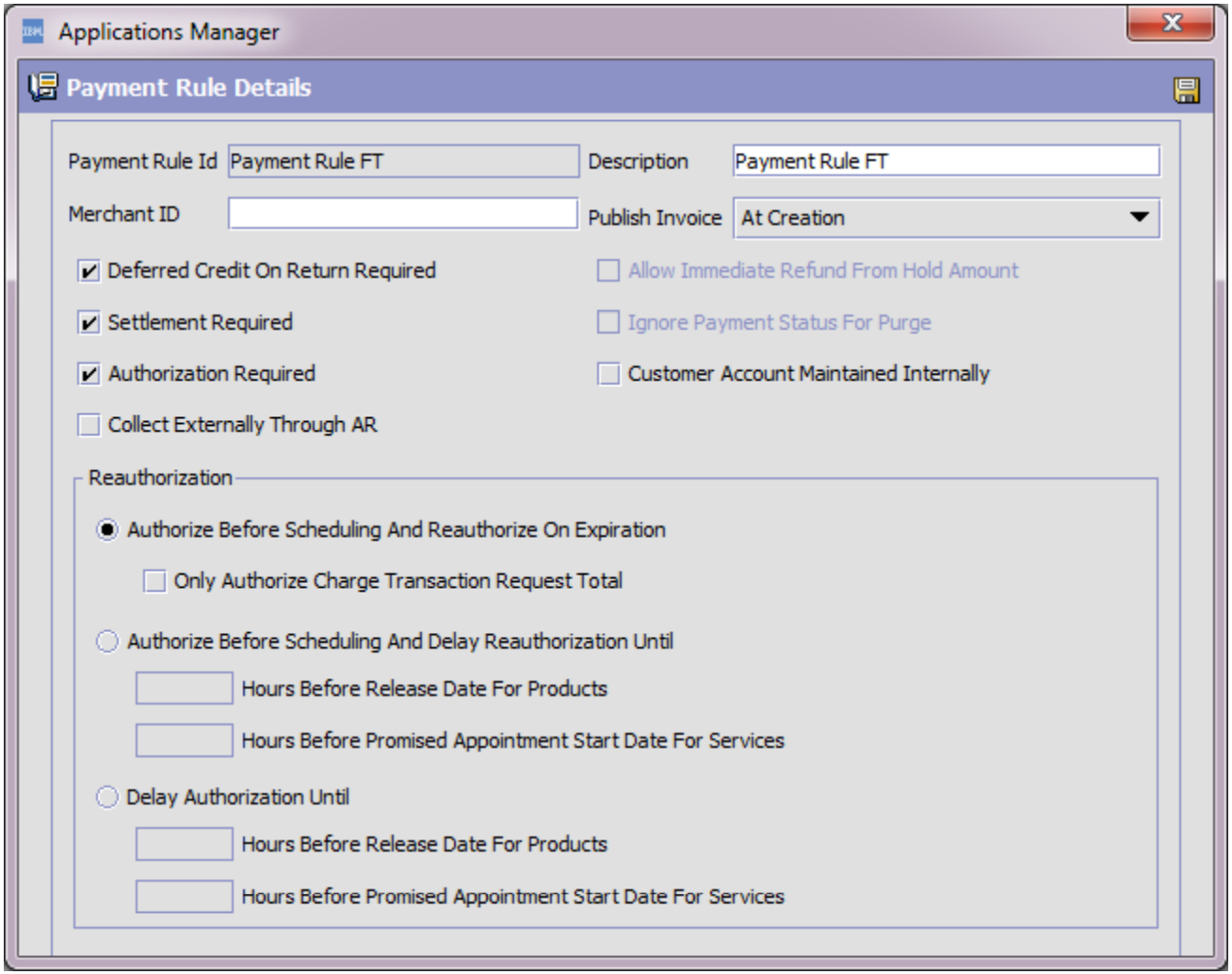


Payment Rule Setting at Seller Organization

One or more payment rules can be configured for the Seller organization. One of these rules should be set as the default payment rule for the Seller organization.

- **FINANCIAL_HANDLING_REQUIRED:** S (Settlement only), A (Authorization Only), B (Both), N (None).
- **ORGANIZATION_CODE:** Seller Org.
- **INTERFACE_TIME:** AT_COLLECT / AT_CREATE.
- **COLLECT_EXTERNALLY_THRU_AR:** Y/N
- **YFS_ORGANIZATION.DEFAULT_PAYMENT_RULE_ID:** The actual YFS_PAYMENT_RULE_ID.PAYMENT_RULE_ID.

Payment Rule Setting in DOM -> Cross App -> Financials



Payment Rule Configuration

- **Publish Invoice** – Either at collection or at creation – This is used by the Send Invoice agent to determine when to send an invoice to the financial system
- **Collect externally through AR** – indicates that the invoices created will be handled through an external accounts receivable system
- **Merchant ID** – a third party name for the system that will handle payments
- **Deferred credit on return required** – a flag to indicate whether or not an event should be raised if a pre-collected order is cancelled.
- **Customer Account maintained Internally** – Indicates whether or not customer accounts are maintained in Sterling or outside. It controls whether or not YFSCollectionCustomerAccountUE is called or not.
- **Authorization Required** – Indicates if payments require authorizations or not
- **Settlement Required** – Indicates if invoices must be paid for the order to move into a paid status.

Payment Rule Configuration - Reauthorization

- There are three authorization methods that determine when an order is initially authorized and when it needs to be re-authorized:
- 1) Authorize before scheduling and reauthorize on expiration
- 2) Authorize before scheduling and delay reauthorization until…
- 3) Delay authorization until…

- These are only applicable if the Authorization required flag is checked

Authorization

- The two rules: Authorize before schedule… will require each order to get an authorization from the payment UserExits (CollectionCreditCard, CollectionSVC, etc…) before the order is allowed to be scheduled.
- If the total amount on the order is greater than the authorized amount on the order, the order's payment status will be "await authorization"
- The PAYMENT_COLLECTION and PAYMENT_EXECUTION agents or the processOrderPayments API must be run on the order to get an authorization
- When the executeCollection API or the PAYMENT_EXECUTION agent is run, it will call the payment UserExits for the correct payment type to get an authorization from a payment gateway.
- The APIs needed to get an authorization is requestCollection, executeCollection, requestCollection OR one call to processOrderPayments (which internally calls request, execute, request)

Reauthorization

- There are 3 different reauthorization methods that can be selected for a payment rule. These determine when an each order needs to get an authorization and when the authorization expires, when to get another authorization.
- Typically, you cannot charge a payment type unless you have an authorization. If an authorization expires, you cannot charge that payment method and pay your order.
- The first two rules require an authorization before your order is allowed to be scheduled. The difference is when to get another authorization:
  - Once an authorization expires, the first rule will get another authorization immediately (when the agents pick the order up)
  - The second rule will create “future authorizations” along with real authorizations that expire a configurable number of hours before the release date – this prevents the payment agents from picking up the order and reauthorizing

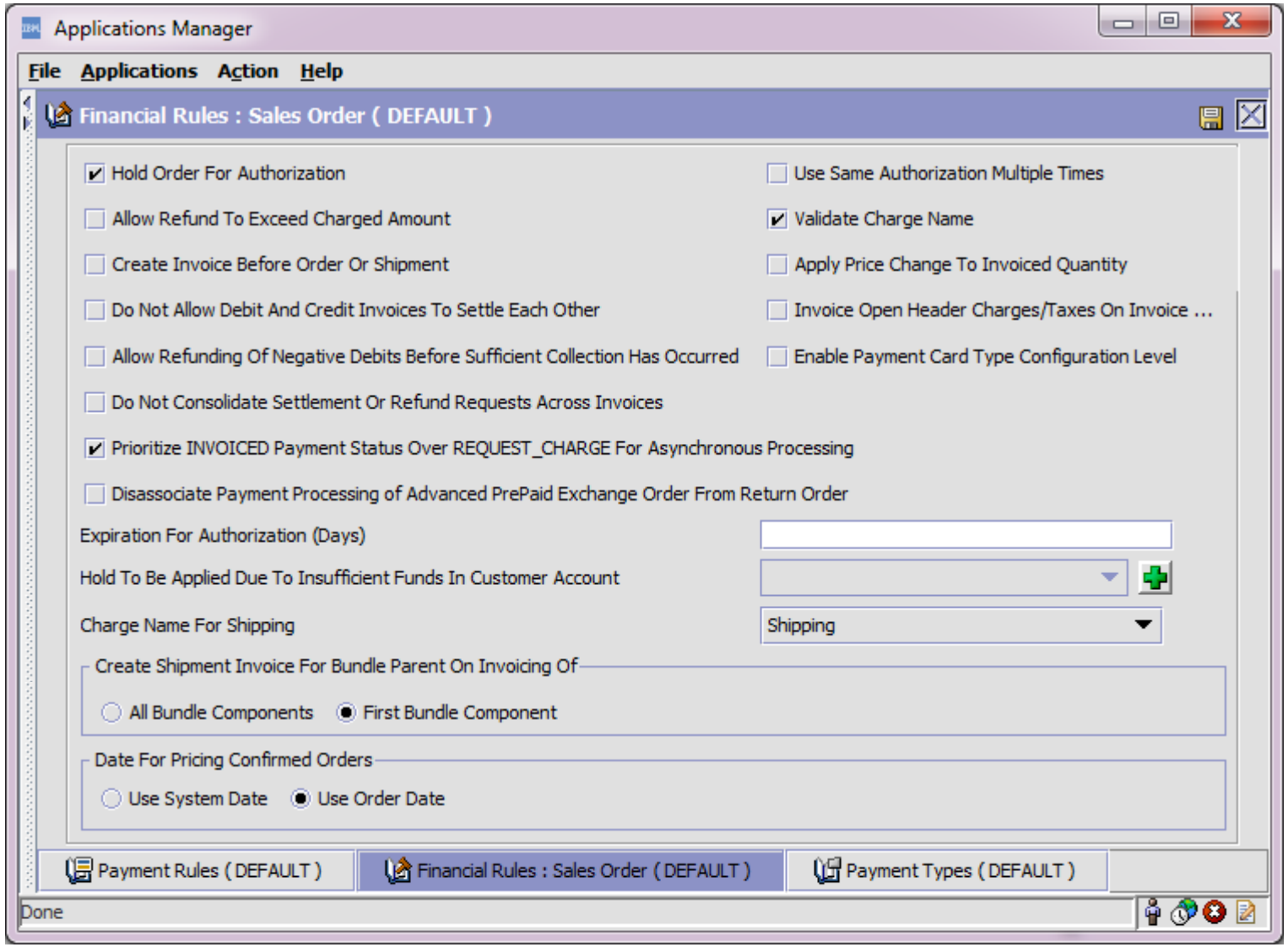
Reauthorization (cont.)

- The third reauthorization rule is similar to the second, but does not get the initial authorization.
- When your order is created, it will create “future authorizations” that expire on highdate. When schedule is run, it will recreate those future authorizations to expire a configurable number of hours before the release date
- Customers may want to limit the number of authorizations for orders placed far in the future. Many unnecessary authorizations will cost the company money for using the payment gateway.

PAYMENT RULESET LEVEL SETTING

- Hold Order For Authorization
- Use Same Authorization Multiple Times
- Expiration For Authorization (days)
- Validate Charge Category and Name
- Allow Refund To Exceed Charged Amount

PAYMENT RULESET LEVEL SETTING

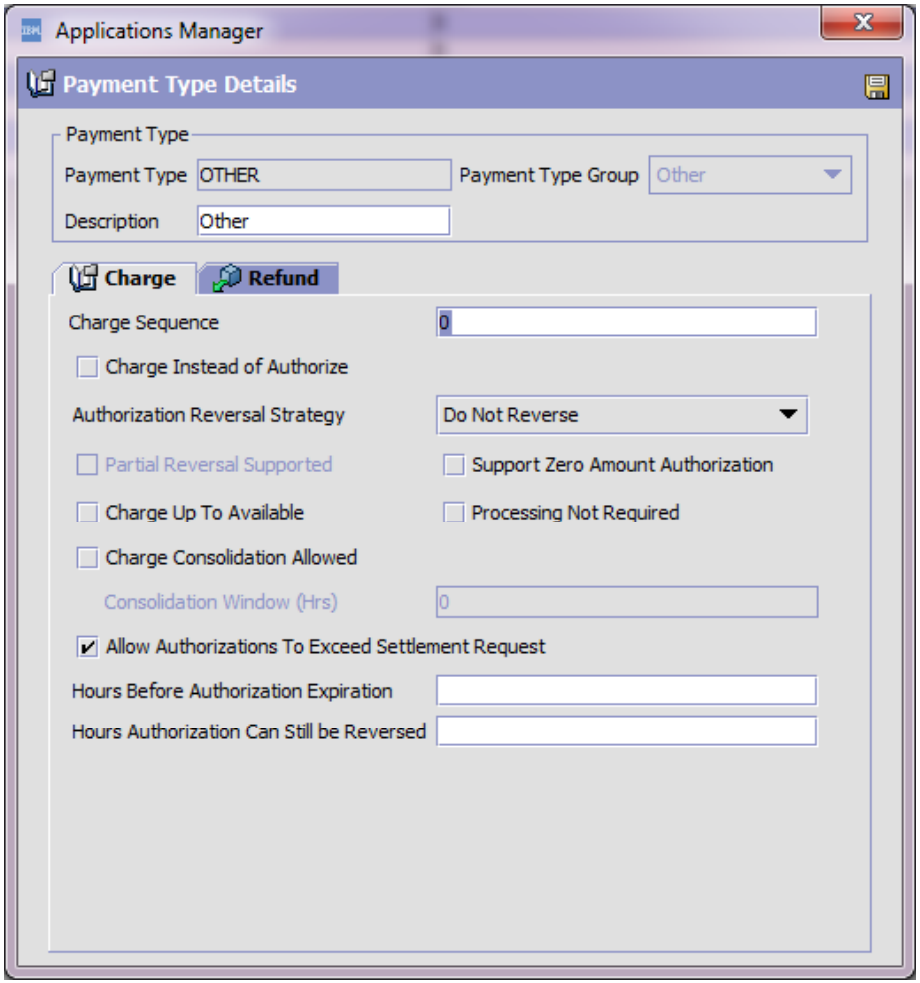


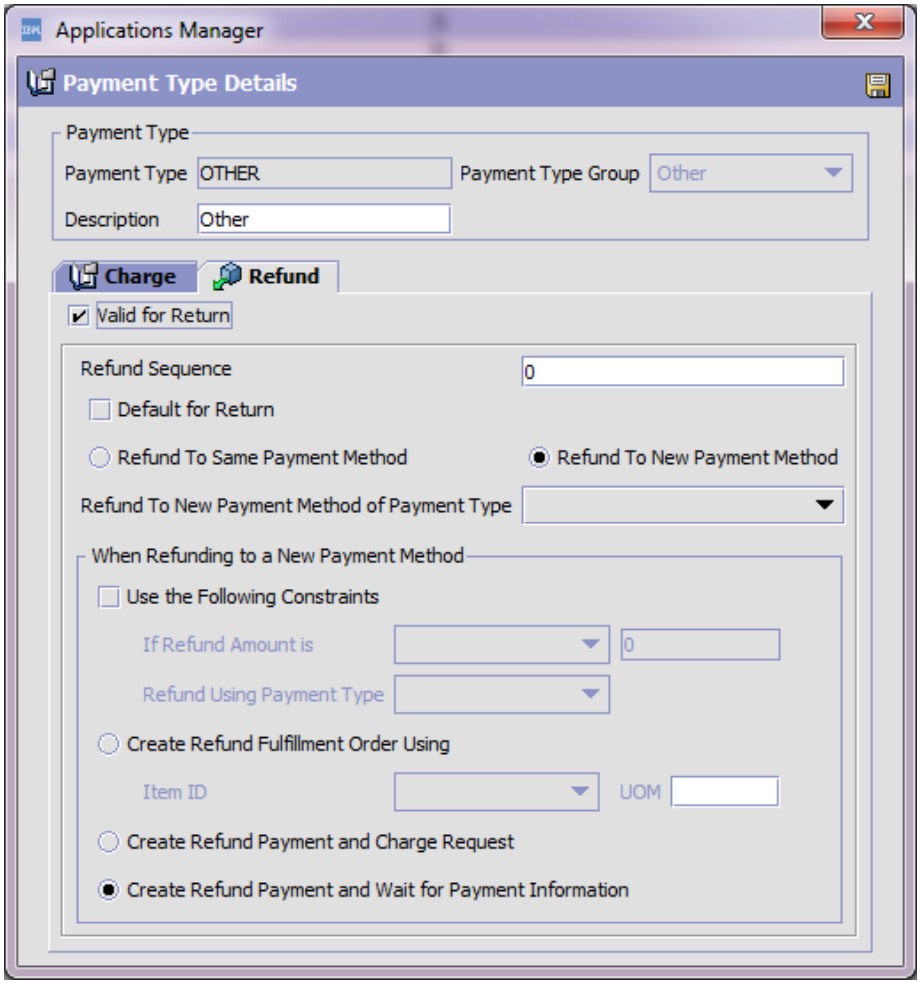
Payment Types

Stored in YFS_PAYMENT_TYPE table

- **CHARGE SEQUENCE:** *Valid values are integer numbers. Determines the sequence in which authorization/charge are placed*
- **REFUND SEQUENCE:** *Valid values are integers. Determines the refund sequencing.*
- **CHARGE CONSOLIDATION ALLOWED:** *Y/N. Determines whether to consolidate the charging for a particular payment type. (Not used for returns).*
- **CONSOLIDATION WINDOW :** *Hours. Used to determine how long to hold a payment without sending it to cybersource or any other payment interface.*
- **VALID FOR RETURN:** *Y/N.*
- **DEFAULT FOR RETURN:** *Y/N.*

Payment Types Configuration





Passing Payments During CreateOrder Time

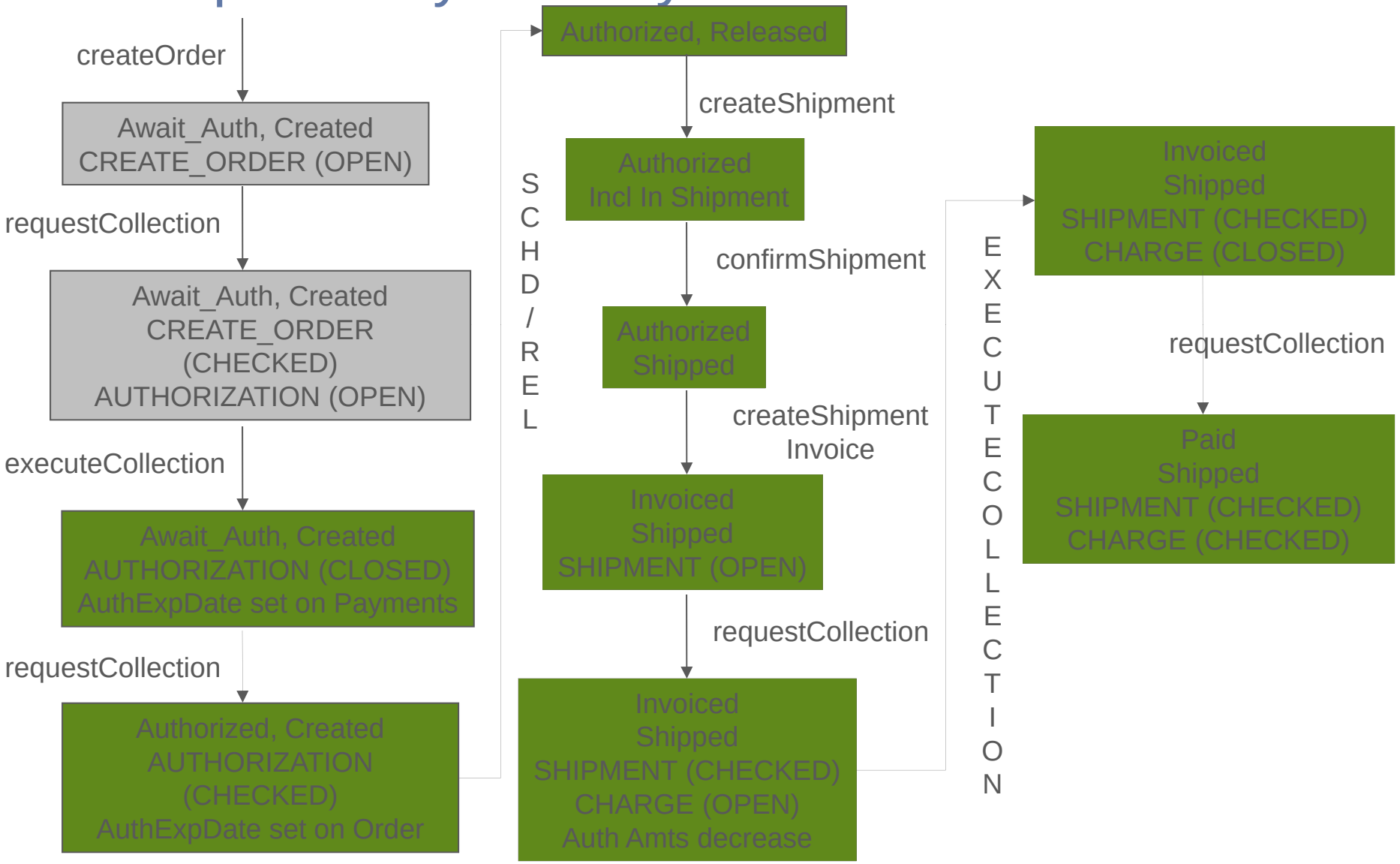
- Create Order:
  - Multiple payment methods can be passed.
- Order moves to Await Pay Info and createOrder raises INCOMPLETE_PAYMENT_INFO Event if:
  - CreditCardNo is not passed for CREDIT_CARD
  - Payment_Reference1 is not passed for other payment types
  - If no Payment Methods are passed.

Payment Method Key Values

Some key parameters that can be passed for payment methods during create order time are:

- **Charge Sequence:** Order of Charge/Auth
- **Max Charge Limit:** Max value that can be charged
- **Unlimited Charges:** Y/N. Overrides Max Charge Limit
- **DisplayCreditCardNo:** eg. Credit Card No XXXXXXXXXXXX9999 if Display CCNo value is 9999
- **Order/PaymentRuleId:** If not passed will default to Seller Org's Default Payment Rule Id.

Complete Payment Cycle of a Sales Order



Pre-Authorized/Pre-Charged Cases

- Order Payment Status = “Await Authorization”.
- Authorized Amount = “/PaymentDetails/@ProcessedAmount” in create Order for Pre-Auth
- Collected Amount = “/PaymentDetails/@ProcessedAmount” in create Order for Pre-Charge
- /PaymentDetails/@ChargeType=‘AUTHORIZATION’ (Pre-Auth)
- /PaymentDetails/@ChargeType=‘CHARGE’ (Pre-Charged)
- Yfs_charge_transaction.status=‘CLOSED’ at createOrder time for AUTHORIZATION & CHARGE type of records.
- Running request collection agent will authorize the order.
- Order payment status = “AUTHORIZED” status.
- YFS_CHARGE_TRANSACTION.STATUS=“CHECKED”
- OrderHeader’s Auth Expiration Date = min(yfs_charge_transaction.Auth_Exp_Date) for that order.
- All UE info logged into YFS_CREDIT_CARD_TRANSACTION

Invoicing/Charging

- CreateOrderInvoice or CreateShipmentInvoice can be used.
- Creates a “SHIPMENT” record with STATUS=‘OPEN’.
- Order Invoice is created. (Works of Status Qty)
- Run requestCollection now.
- Charge Type = “CHARGE” records created.
- The charge requests will be made in the sequence it was authorized.
- The authorized amounts are now reduced.
- Running execute collection will charge the order.
- Running request collection will move the order to PAID status.

Manual Authorizations/Credit/Debit Memo

Manual Authorization/Charging:

- Used for manually entering authorizations or charging that has been done on the order.
- Authorization/Charge records are similar to Pre-Auth/charge records.
- Running requestCollection will authorize or charge the order.

Credit/Debit Memo:

- Used for crediting or debiting money to/from customer.
- ADJUSTMENTS charge type records are created.
- For Credit Memo, Invoice Amount = -$Amount To Be Given
- For Debit Memo, Invoice Amount = $Amount To Be Collected

CREDIT / DEBIT MEMO

Order Collection Details

Help Close

Order

Enterprise	E1	Buyer		Seller	S1
Order #	CreditDebitMemo1	Status	Shipped	Order Date	11/29/2002
Order Type	[X]	Carrier/Service		Requested Delivery Date	07/02/2003

Collection Details

Total Collected (Credits)	$ 1,110.00	Total Invoiced (Debits)	$ 1,100.00
Total Authorized	$ 0.00	Open Order Amount	$ 0.00

Date	Transaction Type	Collected	Invoiced	Open Order	Authorized	Requested Amount	Pre Settled Amount	Status
07/02/2003	CREATE_ORDER	0.00	0.00	1,110.00	0.00	0.00	0.00	CHECKED
07/02/2003	AUTHORIZATION ⊞	0.00	0.00	0.00	610.00	610.00	0.00	CHECKED
07/02/2003	AUTHORIZATION ⊞	0.00	0.00	0.00	500.00	500.00	0.00	CHECKED
07/02/2003	SHIPMENT	0.00	1,110.00	-1,110.00	0.00	0.00	0.00	CHECKED
07/02/2003	CHARGE ⊞	500.00	0.00	0.00	-500.00	500.00	0.00	CHECKED
07/02/2003	CHARGE ⊞	610.00	0.00	0.00	-610.00	610.00	0.00	CHECKED
07/02/2003	ADJUSTMENTS	0.00	-20.00	0.00	0.00	0.00	0.00	CHECKED
07/02/2003	CHARGE ⊞	0.00	0.00	0.00	0.00	-10.00	0.00	OPEN
07/02/2003	ADJUSTMENTS	0.00	10.00	0.00	0.00	0.00	0.00	CHECKED

Some more tips on Auth/Charge

Authorization Only (Payment Rule)

Charge Only (Payment Rule)

Collect Externally Through AR (Payment Rule):

- Invoice shows Collected External Thru AR
- CREATE_ORDER at create Order Time
- SHIPMENT at Shipment Time
- Order moves from Authorized to Paid status

ADDITIONS and ADJUSTMENTS:

- ADDITIONS : new qty or line with more charges
- ADJUSTMENTS : CREDIT/DEBIT
- SHIPMENT_ADJUSTMENTS: When final shipment total is different from original order total.

Some more Auth/Charge Tips

Consolidation of Charges

- yfs_charge_transaction.collection_date = date/time of transaction + yfs_payment_type.min_wait_time_for_consol
- yfs_payment_type.charge_consol_allowed=Y/N
- ExecuteCollection does not pickup until collection_date

Minute Differences Between RequestColl Agent and API

RequestCollection Agent picks up records if:

- payment status != 'PAYMENT_HOLD'
- and != 'NOT_APPLICABLE'
- and (AuthExpDate – Days for Expiration) <= sysdate

API does not consider any of these.

The above query changed in 5.0-SP2 to select all payment statuses except PAYMENT_HOLD, and NOT_APPLICABLE and with other parameters remaining same.

Some more Tips on Auth/Charge

ExecuteCollection

- Works on any charge transaction “AUTHORIZATION” OR “CHARGE” that are open
- Will not pick any charge transaction record that have collection date > date_time_stamp at point of run.
- API can override the above behaviour using “IgnoreCollectionDate” parameter.

Suspend Charges

- yfs_payment.suspend_any_more_charges

PreSettlement

- Mechanism to collect some amount or total amount before order is shipped.
- Use changeOrder() and pass settled qty
- REQUEST_SETTLEMENT is created
- Running through one payment cycle will charge for that settled quantity
- Rest of the charge will happen during createShipmentInvoice time.
- Invoice for settled quantity

Asynchronous Processing

UserExit is coded to do asynchronous processing

- asynchRequestProcess=true
- Running executeCollection moves Payment Status to “Requested_Auth” or “Requested_charge”
- recordExternalCharges() records the auth or collected amounts.
- RequestCollection moves orders to “AUTHORIZED” or “PAID” status.

RecordCollection / Send Invoice

- RecordCollection:
  - PaymentExecution is not run external collection
  - Use recordCollection to log the charge or auths made in cybersource or other external systems in a batch mode
- Publish Invoice
  - At Collect: publishes invoice only after collection
  - At Create: publishes invoice after invoice creation

RETURNS PAYMENT PROCESSING

Returns Against Sales Order

- Payment methods on the Sales Order
- Multiple payments can be refunded
- Payments can be suspended for refund

Refund Logic

- Ascending order of configurator's payment type setting (&) Descending order of Sales Order's Charge Sequence
- VALID_FOR_RETURN payments only
- **First parse:** Try refund whatever is possible.
- **Second parse:** PAYMENT->EXCEED_CHARGE_AMOUNT_FOR_REFUND = Y.
- **Third parse:** Payment_type -> "DEFAULT_FOR_RETURN"
- **None of the above succeeds:** Raise Incomplete Payment Information Event.
- Max Charge Limit / Unlimited Charges is not read.

RETURNS PAYMENT PROCESSING

- Can be invoiced pre-receipt (OR) post-receipt
- Disadvantages of Pre-Receipt
  - Over-received quantity might not be refunded
- Line Price Info is derived from parent Order.
- Header Charges/Taxes are not derived.
- Line Charges/Taxes are not derived also.
- For BlindReturns these can be passed at CreateOrder Time.
- CREATE_ORDER_INVOICE.0003 transaction is used. This is a derived transaction.
- No Authorization Required.

Some more tips on return

- Invoice is created for the return
- For Blind Returns, only one payment method is processed
- Return Payment Status = Authorized.
- Moves to Paid after running requestCollection for returns
- For Returns Against Sales Order PAYMENT_COLLECTION.0001, PAYMENT_EXECUTION.0001 are used.
- For Blind Returns, PAYMENT_COLLECTION.0003 and PAYMENT_EXECUTION.0003 are used.

IBM Sterling Sensitive Data Capture Server (SSDCS) Overview

- Introduction
- Installing SSDCS
- Configuring SSDCS
- Deploying SSDCS

Introduction

- SSDCS
  - The IBM Sterling Sensitive Data Capture Server (SSDCS) is an application that integrates with the Sterling Selling and Fulfillment Suite to ensure that credit card numbers and stored value card numbers are secure by tokenizing them. SSDCS enables the Sterling Selling and Fulfillment Suite to achieve Payment Application Data Security Standard (PA-DSS) compliance.
- Payment Application Data Security Standard (PA-DSS)
  - Standard for payment applications, which are systems that capture information pertaining to cards, such as credit cards. It mandates that sensitive information, such as a credit card number, should not be stored along with application data.
- Payment Card Industry Data Security Standard (PCI-DSS)
  - Standard for the secure implementation of payment applications by customers.

Introduction (Contd..)

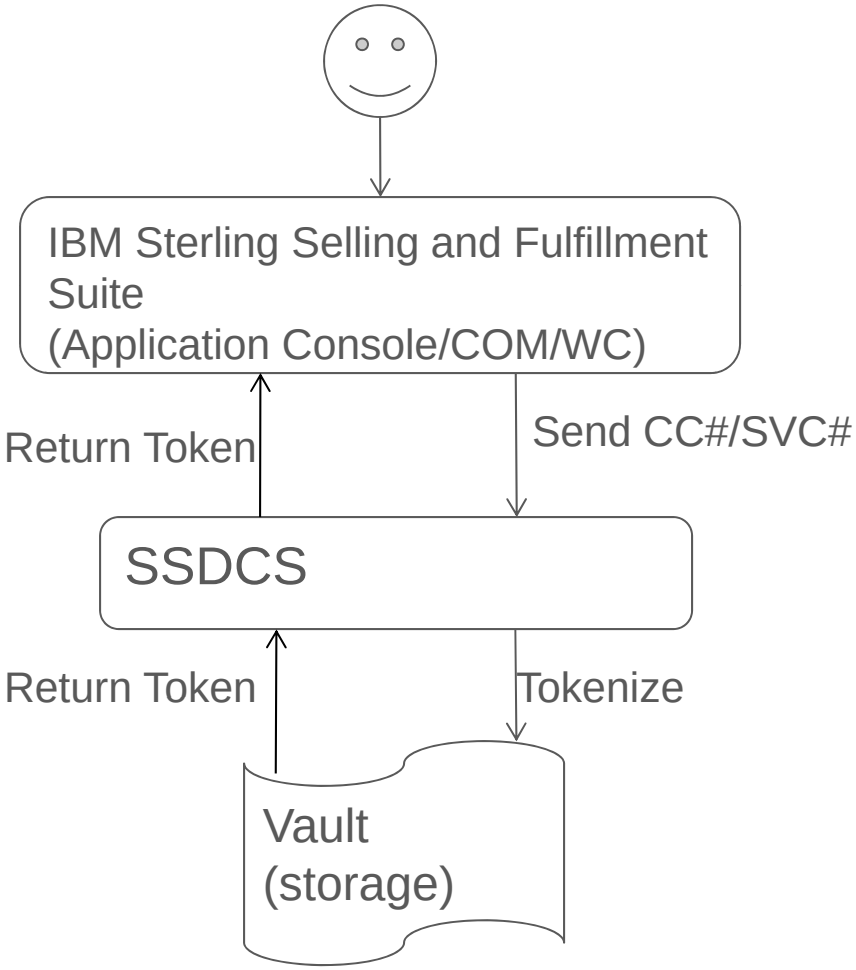
- Purpose
  - Tokenize CC#/SVC#
  - Foundation is out of scope for PA-DSS

- Tokenization
  - Process of storing CC#/SVC# in a vault system that associates a token to a securely stored CC#/SVC#.
  - The only way in which a token can be returned to its original value is by contacting the vault system.

Introduction (Contd..)

- SSDCS is:
  - A brand new application shipped with Sterling Selling and Fulfillment Suite 9.0
  - Used for validating CC# (Luhn Algorithm)
  - Used for handling CC#/SVC# tokenization
- SSDCS is not or does not:
  - An application that consumes platform. Therefore, exiting stories around user exits, property override through customer_overrides.properties, or any framework level components does not exist for SSDCS.
  - Provide a configuration user interface. Configuration is handled through properties only.
  - Provide any APIs, only Servlets.
  - Require any database component.

Introduction (Contd.)



Installing SSDCS

- SSDCS is packaged as a zip file with foundation, located under <INSTALL_DIR>/repository/external/ssdcs.zip.
- To Install:
  1) Determine the location in which SSDCS files will reside. Define a variable <SSDCS_DIR> for this location.
  2) Unzip ssdcs.zip to the <SSDCS_DIR>
     - bin – contains scripts
     - documentation – contains Javadocs
     - jar & jar/extn – contains product JAR, third-party JARs, and customer extension JARs
     - log – Directory for log files. Set SSDCS_LOG_DIR to <SSDCS_DIR>/log (using absolute path).
     - properties – contains property files
     - resources – contains resource files; for example, web.xml
     - WebContent – contains jsps
     - external_deployment – location of the compiled WAR file

Configuring SSDCS - Properties

- Set foundation properties
  - yfs.ssdcs.url=https://<ip>:<port>/ssdcs
  - yfs.ssdcs.servlet=/tokenize
  - yfs.ssdcs.jsp=/jsp/ssdcs_tokenize_pan.jsp
  - yfs.ssdcs.tokenize.svc=Y|N
  - yfs.ssdcs.tokenize.cc =Y|N
  - sc.accesss.token.expire.in.seconds (optional)
  - sc.access.token.max.allowed.expire.in.seconds (optional)
- Set SSDCS properties
  - ssdcs.smcfs.url=https://<ip>:<port>/smcfs/accessTokenServlet
  - ssdcs.ue.PanValidator (optional) - The Primary Account Number (PAN) validation implementation.
  - ssdcs.ue.Tokenize - The tokenization implementation.

Configuring SSDCS – User Exists

- PanValidator
  - PanValidator is provided to validate that the credit card or stored value card number are valid primary account numbers.
  - If not implemented, the Luhn algorithm will be run to validate the pan.
  - Input
    - <PanValidator Pan="" SterlingPaymentType=""/>
  - Ouput
    - <PanValidator PanValid="true|false|<blank>" CardType=""/>
- Tokenize
  - Tokenize is provided to contact a 3rd party vault vendor to tokenize sensitive data.
  - Input:
    - <Tokenize Pan="" DisplayNumber="" SterlingPaymentType=""/>
  - Output:
    - <Tokenize Token="" DisplayNumber=""/>

Configuring SSDCS – ESAPI Properties

- ESAPI.properties
  - Contains validation patterns that have Validator.ssdcs. as a prefix.
- Hidden Form Fields on the JSP being validated:
  - ssdcsAuthenticationToken
  - ssdcsCssURl
  - ssdcsRedirectUrl
  - ssdcsDataType
  - ssdcsDataTypeDetail
  - ssdcsDataToTokenize
  - ssdcsToken
  - ssdcsDisplayValue
  - ssdcsResultCode
  - ssdcsFailReason
  - ssdcsResultDescription
  - ssdcsTabIndex
  - ssdcsAdditionalResultData
  - ssdcsAutoSubmit
  - ssdcsDbg

Configuring SSDCS - Logging

- system.log - Used for logging business logic issues, such as debugging and timing information.
- access.log - Used exclusively for logging security issues. The access logger records the activity of any malicious requests that are detected, such as unauthorized users attempting to log in.
- ESAPI.log - Used for debugging the ESAPI setup. Both ESAPI internal classes and SSDCS extensions and implementations are logged here.

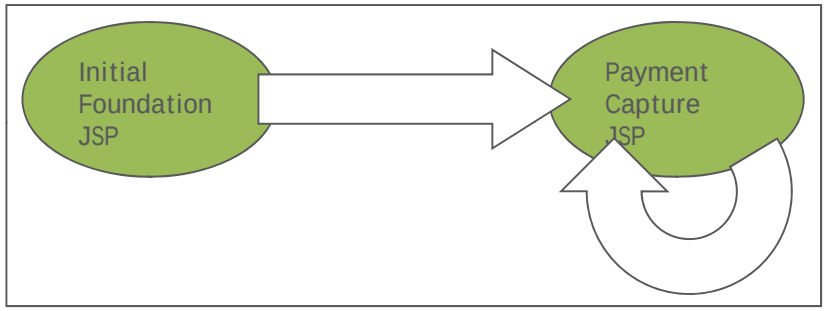
  - Note: Sensitive information, such as CC#/SVC#, is not logged.

Deploying SSDCS

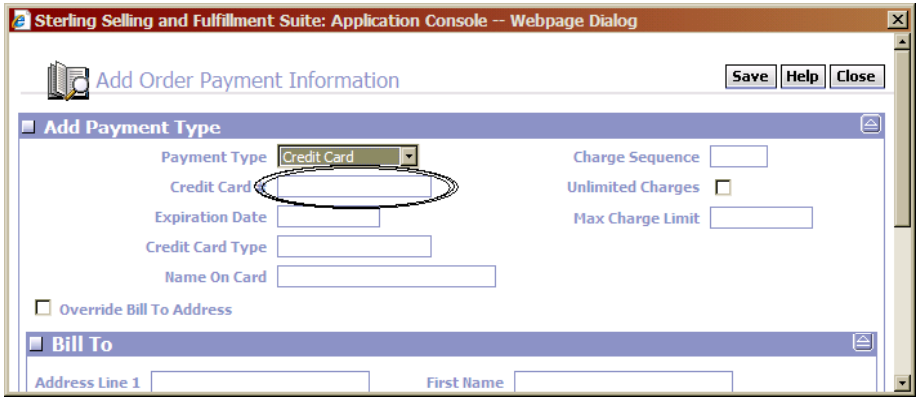
- Open command window and navigate <SSDCS_DIR>/bin
- Run the following command after ensuring that ant is in your classpath. This will build ssdcs.war and it will be located under the external_deployment directory
  - Ant –f build_deployment.xml –Dappserver_type=<weblogic|websphere|jboss>
- Deploy ssdcs.war

JSPs

- The JSPs for SSDCS are embedded within an IFRAME that captures CC# and SVC#.



- For example:



JSPs

When a CC# or SVC# is entered:

1. SSDCS validates whether the data entered in the form fields is valid data. For example, the ssdcsAuthenticationToken field must consist of only alphanumeric characters.
2. SSDCS validates whether the session token passed to it from Selling and Fulfillment Foundation is a valid session. If the session is not valid, Selling and Fulfillment Foundation returns an error.
3. SSDCS validates the credit card number or stored value card number that is entered. For credit card numbers, this involves running the Luhn algorithm. If the number is not valid, SSDCS returns an error.
4. After the CC# or SVC# is validated, SSDCS calls the Tokenize UE to tokenize the PAN and contacts the vault to store it.
5. The UE returns a token as well as a display value. The display value is what is displayed in the JSP text field instead of the CC# or SVC# that was entered.

Conclusion

- SSDCS – New Application in IBM Sterling Selling and Fulfilment Suite 9.0.
  - No Sterling application can store PAN. Each application must implement SSDCS to handle CC#/SVC# tokenization going forward.
  - PA-DSS certification
  - Keep foundation out of scope
  - Tokenize CC# and SVC#

- Install/Configure/Deploy SSDCS

Additional Resources

- Related Documentation
- IBM Sterling PDFs
  - Product concepts guide, Chapter 12
  - Distributed Order Management Config guide, Financial sections
  - SSDCS Configuration Guide
  - PA-DSS implementation Guide
- IBM Sterling Javadocs
  - Apis covered in this document.

Any Questions