# Part III
# Payment System

## Overview

### Introduction

Part III presents the description and processing of the payment system portion of the SET protocol, including all messages related to authorization, capture, and management of the payment system.

### Organization

Part III includes the following chapters:

| Chapter | Title | Contents | Page |
|---|---|---|---|
| 1 | Common Data and Flows | Presents data structures used throughout the protocol, and describes the message flows embodied in the protocol. | 368 |
| 2 | Cardholder/Merchant Messages | Describes the messages exchanged between the Cardholder and Merchant applications in the course of the protocol. | 404 |
| 3 | Merchant/Payment Gateway Messages | Describes the messages exchanged between the Merchant and Payment Gateway applications in the course of the protocol. | 464 |

# Chapter 1
# Common Data and Flows

## Overview

### Introduction

Chapter 1 presents the data structures common to payment messages, and presents the message flow model for the payment system.

### Organization

Chapter 1 includes the following sections:

| Section | Title | Contents | Page |
|---|---|---|---|
| 1 | Data Structures | Presents data structures common to multiple payment messages. | 369 |
| 2 | General Flow | Presents a summary of a typical payment flow, plus a summary of all messages which may be present in payment system flows. | 400 |

### Notation

The notation used in the data structure tables (such as Table 1 on page 370) is presented in Part I on page 93.

### Date format

Any SET application that receives a date that contains fractional seconds shall retain the fractional seconds to use in subsequent response messages. That is, copy the date exactly as sent.

# Section 1
# Data Structures

**Definition**

SET messages include several data structures that bear data items which recur from message to message, representing control structures, recurring application data, etc.

The following tables define logically-related groups of fields that appear in more than one message. These definitions are presented here for ease of reference.

# TransIDs

**Purpose**

**TransIDs** provides all the data necessary to uniquely identify the transaction of which the message is a part. In particular, **TransIDs** enables an entity to relate each message to the transaction of which it is a part as well as to the request/response pair ~~(since the request/response pairs can occur only once in each transaction)~~.

**TransIDs data**

| **TransIDs** | **{LID-C, [LID-M], XID, PReqDate, [PaySysID], Language }** |
|---|---|
| **LID-C** | *Local ID; convenience label generated by and for Cardholder system* |
| **LID-M** | *Local ID; convenience label generated by and for Merchant system.* |
| **XID** | *Globally unique ID.* |
| **PReqDate** | *Date of purchase request; generated by Merchant in* **PInitRes** *or by Cardholder in* **PReq**. |
| **PaySysID** | *Used by some payment card brands to label transaction from time of authorization onward* |
| **Language** | *Cardholder's natural language* |

**Table 1: TransIDs Data**

**LID-C, LID-M, and PaySysID**

**LID-C**, **LID-M**, and **PaySysID** are identifiers which are assigned, respectively, by the Cardholder, Merchant, and payment system infrastructure to tag transactions in a manner convenient for each of them; however, other parties may not assume characteristics of these labels. **LID-M** may often be used to hold the Merchant's order number associated with the transaction.

**Generating XID**

**XID** is a transaction ID usually generated by the Merchant system, unless there is no **PInitRes**, in which case it is generated by the Cardholder system. It is a randomly-generated 20-byte variable that is globally unique (statistically). Merchant and Cardholder systems shall use appropriate random number generators to ensure the global uniqueness of **XID**.

**PReqDate and Language**

**PReqDate** provides the date of the transaction start and **Language** provides the language the Cardholder requests for the transaction. They are included here for convenience so that they travel with each message.

# PI (Payment Instructions)

**Purpose**

**PI** (Payment Instructions) is the most central and sensitive data structure in SET. It is used to pass the data required to authorize a payment card payment from the Cardholder to the Payment Gateway, which will use the data to initiate a payment card transaction through the traditional payment card financial network. The data is encrypted by the Cardholder and sent via the Merchant, such that the data is hidden from the Merchant unless the Acquirer passes the data back to the Merchant.

**Variations**

There are three versions of the **PI**.

| | |
|---|---|
| **PIUnsigned** | Created by a Cardholder with no signature certificate. Used in a **PReqUnsigned** message.<br>Data integrity is provided through the addition of a hash of the **PI** data which is protected in the OAEP block. No source authentication is provided by this mechanism. |
| **PIDualSigned** | Created by a Cardholder that possesses a signature certificate. Used in a **PReqDualSigned** message.<br>The Cardholder signature authenticates the source as well as providing data integrity. |
| **AuthToken** | Created by the Payment Gateway. The Merchant extracts the **PI** for later incorporation into **AuthReq**.<br>This version is used to support split shipments and installment or recurring payments, and is passed back from the Payment Gateway after initial authorization to be used to request subsequent authorizations. |

**Table 2: PI Variations**

**Two parts**

The Payment Instructions consist of two parts:

| | |
|---|---|
| **PANData** | contains the payment card data and is provided stronger cryptographic treatment. |
| **PIData** | contains all other payment data, transaction data, and cryptographic support variables. |

**Table 3: PI Parts**

**Purchase amount**

If **InstallRecurData** exists (that is, if the PI is for installment or recurring payments, as discussed on page 375), **PurchAmt** reflects the anticipated total purchase amount agreed upon between the Merchant and the Cardholder, rather than the amount of any one payment.

*Continued on next page*

# PI (Payment Instructions), continued

### PI data

| | |
|---|---|
| **PI** | **< PIUnsigned, PIDualSigned, AuthToken >**<br>*Cardholder creates* **PIUnsigned** *or* **PIDualSigned**.<br>*Payment Gateway creates* **AuthToken** *to support split shipments or installment/recurring payments.*<br>*Merchant shall retain the* **PI** *for later incorporation into* **AuthReq**. |
| **PIUnsigned** | **EXH(P, PI-OILink, PANToken)**<br>*See page 382 for* **PANToken**. |
| **PIDualSigned** | **{PISignature, EXL(P, PI-OILink, PANData)}**<br>*See page 381 for* **PANData**. |
| **AuthToken** | *See page 378.* |
| **PI-OILink** | **L(PIHead, OIData)**<br>*See page 373 for* **PIHead**. *See page 436 for* **OIData**. |
| **PISignature** | **SO(C, PI-TBS)** |
| **PI-TBS** | **{HPIData, HOIData}** |
| **HPIData** | **DD(PIData)** |
| **HOIData** | **DD(OIData)**<br>*See page 436 for* **OIData**. |
| **PIData** | **{PIHead, PANData}**<br>*See page 373 for* **PIHead**.<br>*See page 381 for* **PANData**. |

**Table 4: PI Data**

*Continued on next page*

# PI (Payment Instructions), continued

### PIHead data

| | |
|---|---|
| **PIHead** | **{TransIDs, Inputs, MerchantID, [InstallRecurData], TransStain, SWIdent, [AcqBackKeyData], [PIExtensions]}** |
| **TransIDs** | *See page 370.* |
| **Inputs** | **{HOD, PurchAmt}** |
| **MerchantID** | *Copied from Merchant signature certificate* |
| **InstallRecurData** | *See page 377.* |
| **TransStain** | **HMAC(XID, CardSecret)** |
| **SWIdent** | *String identifying the software (vendor and version) initiating the request. It is specified in the* **PI** *so the Payment Gateway knows the software of the Cardholder.* |
| **AcqBackKeyData** | **{AcqBackAlg, AcqBackKey}** |
| **PIExtensions** | *The data in an extension to the payment instructions ~~must~~ shall be financial and should be important for the processing of an authorization by the Payment Gateway, the financial network, or the Issuer.* |

**Table 5: PIHead Data**

*Continued on next page*

# PI (Payment Instructions), continued

**PIHead data** (continued)

| | |
|---|---|
| **HOD** | *The same value as placed in* **OIData**. *See* "OIData" *on page 436* |
| **PurchAmt** | *The amount of the transaction as specified by the Cardholder* |
| **XID** | *Copied from* **TransIDs**; *see page 370* |
| **CardSecret** | *See* "PANData0" *in Part II on page 271.* |
| **AcqBackAlg** | *Selected from Encryption IDs in Payment Gateway certificate.* |
| **AcqBackKey** | *Key for* **AcqCardMsg** *of an appropriate length for* **AcqBackAlg** |

**Table 5: PIHead Data,** continued

**PI extension guidelines**

SET Payment Gateway certificates include a private certificate extension, **SETExtensions**. This certificate extension lists the object identifiers of the message extensions that the Payment Gateway can process in payment instructions. Cardholder software ~~can~~ shall use this data to ensure that no unrecognized critical extension is put into the payment instructions (in **PIExtensions** or in **InstallRecurData.SIRExtensions**). See Part II, page 336, for further detail.

# InstallRecurData

**Purpose**

**InstallRecurData** allows the Cardholder to authorize installment or recurring payments. This component of the **PI** (Payment Instructions) is copied into the authorization token **(AuthToken)** described on page 378.

**Recurring-Frequency**

**RecurringFrequency** indicates the minimum number of days between authorizations. A frequency of monthly is indicated by a value of 28. The earliest possible date for each authorization is based on the actual date of the prior authorization. For example, if **recurringFrequency** is 28, the following authorization dates are acceptable:

| typical authorization dates | earliest possible authorization dates |
|---|---|
| 1/31/99 | 1/31/99 |
| 2/28/99 | 2/28/99 |
| 3/31/99 | 3/28/99 |
| 4/30/99 | 4/25/99 |
| 5/31/99 | 5/23/99 |
| 6/30/99 | 6/20/99 |

Later authorizations are acceptable (until **recurringExpiry**).

**Recurring-Expiry**

It is the responsibility of the Cardholder and Payment Gateway software to ensure that **recurringExpiry** is not later than the card expiration date.

Note: The card needs to be valid only at the time of authorization. It is not a problem if it expires between authorization and capture.

*Continued on next page*

# InstallRecurData, continued

### Create InstallRecurData

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***installTotalTrans*** — an integer representing the maximum number of permitted authorizations for installment payments (optional)<br>***recurringFrequency*** — an integer representing the minimum number of days between authorizations (optional)<br>***recurringExpiry*** — the final date after which no further authorizations are permitted (optional)<br>Note: Either ***installTotalTrans*** will be provided or both ***recurringFrequency*** and ***recurringExpiry*** will be provided; no other combination is valid. |
| 2 | If ***installTotalTrans*** is not provided, continue with Step 4.<br>Otherwise, construct *InstallRecurInd*:<br>*installTotalTrans* — ***installTotalTrans*** |
| 3 | Append the result of Step 2 to the tag [0] and continue with Step 7. |
| 4 | Construct *Recurring*:<br>*recurringFrequency* — ***recurringFrequency***<br>*recurringExpiry* — ***recurringExpiry*** |
| 5 | Construct *InstallRecurInd*:<br>*recurring* — the result of Step 4 |
| 6 | Append the result of Step 5 to the tag [1]. |
| 7 | Construct *InstallRecurData*:<br>*installRecurInd* — the result of Step 3 or Step 6<br>*irExtensions* — any message extension(s) required to support additional business functions (optional) |
| 8 | Return the result of Step 7. |

*Continued on next page*

# InstallRecurData, continued

### InstallRecurData

| | |
|---|---|
| **InstallRecurData** | **{InstallRecurInd, [IRExtensions]}** |
| **InstallRecurInd** | **< InstallTotalTrans, Recurring >** |
| **IRExtensions** | *The data in an extension to installment or recurring data ~~must~~ shall be financial and should relate to the processing of subsequent authorizations by the Merchant and the Payment Gateway.*<br>*Note: The installment/recurring data is not transmitted to the Issuer.* |
| **InstallTotalTrans** | *Cardholder specifies a maximum number of permitted authorizations for installment payments.* |
| **Recurring** | **{RecurringFrequency, RecurringExpiry}** |
| **RecurringFrequency** | *The minimum number of days between authorizations (a frequency of monthly is indicated by a value of 28).* |
| **RecurringExpiry** | *A final date after which no further authorizations are permitted.* |

**Table 6: InstallRecurData**

# AuthToken

**Purpose**

**AuthToken** represents data required by the Payment Gateway for subsequent authorizations of a transaction. It is provided by the Payment Gateway when authorization for part of an order is made. As long as the transaction is incomplete, an **AuthToken** is returned. Once the transaction is completed, no more **AuthToken**s are required or provided. The Payment Gateway updates the **AuthToken** as necessary, and only the Payment Gateway can read the data it contains.

**AuthToken data**

| | |
|---|---|
| **AuthToken** | **EncX(P1, P2, AuthTokenData, PANToken)** |
| **AuthTokenData** | **{TransIDs, PurchAmt, MerchantID, [AcqBackKeyData], [InstallRecurData], [RecurringCount], PrevAuthDateTime, TotalAuthAmount, AuthTokenOpaque}** |
| **PANToken** | *Fields copied from Cardholder-produced* **PIHead**. *See page 373.* |
| **TransIDs** | |
| **PurchAmt** | |
| **MerchantID** | |
| **AcqBackKeyData** | |
| **InstallRecurData** | *See page 377.* |
| **RecurringCount** | *Number of recurring authorizations performed so far.* |
| **PrevAuthDateTime** | *Date and time of Merchant's last authorization in a sequence of recurring authorizations.* |
| **TotalAuthAmount** | *The total amount authorized so far by all authorizations for this* **XID**. |
| **AuthTokenOpaque** | *Opaque data defined by the generating Payment Gateway.* |

**Table 7: AuthToken Data**

# AcqCardMsg

**Purpose**

This field provides a mechanism for an Acquirer to send a message back to the Cardholder without exposing it to the Merchant. It may be sent after the Payment Gateway has received the **AuthReq** message from the Merchant.

**AcqCardMsg data**

| | |
|---|---|
| **AcqCardMsg** | **EncK(AcqBackKeyData, P, AcqCardCodeMsg)**<br>**AcqBackKeyData** *is supplied by the Cardholder in the* **PI**. *The encrypted message is destined to the Cardholder.* |
| **AcqBackKeyData** | *Copied from* **PIHead.AcqBackKeyData***; see page 373.* |
| **AcqCardCodeMsg** | **{AcqCardCode, AcqCardMsgData}** |
| **AcqCardCode** | *Enumerated code. See page 379.* |
| **AcqCardMsgData** | **{[AcqCardText], [AcqCardURL], [AcqCardPhone]}** |
| **AcqCardText** | *Textual message to be displayed to cardholder.* |
| **AcqCardURL** | *URL referencing HTML message to be displayed to cardholder.* |
| **AcqCardPhone** | *Phone number to be presented to the cardholder.* |

**Table 8: AcqCardMsg Data**

**Notes**

**AcqCardMsg** is tunneled from the Acquirer to the Cardholder through the Merchant. The Cardholder sends the symmetric key needed to decrypt it to the Merchant in the **PI**; the Merchant passes the key to the Payment Gateway. The Merchant receives **AcqCardMsg** in **AuthRes** and shall copy it to **PRes** and **InqRes**.

This optional field is available only if supported by the profile of a payment card brand via the Payment Gateway's encryption certificate **(Cert-PE)**.

**AcqCardCode**

The following values are defined for **AcqCardCode.**

| | |
|---|---|
| messageOfDay | *A message the Acquirer wishes to display to all users.* |
| accountInfo | *Information about the account to be passed back to the user.* |
| callCustomerService | *Prompts the application to display a message requesting that the user call Customer Service.* |

**Table 9: Enumerated Values for AcqCardCode**

# CapToken

**Purpose**

If authorization without capture is requested, the Payment Gateway may generates a capture token and returns it as part of **AuthRes**. **CapToken** represents data required by the Payment Gateway for capture of the authorized transaction.

Note: **CapToken** is optional at the Acquirer's discretion. It is one way to save data for capture processing, but the data may instead be saved on the Merchant system, the Payment Gateway system or any other Acquirer designated system.

For example, the Payment Gateway will require the account number (PAN) to process the capture. To have the PAN available, the Payment Gateway may:

- store the PAN in the transaction record (and retain the transaction record until the authorization expires, or until so much time has passed since the capture that the Merchant can no longer perform a credit), or
- populate **PANToken** as part of **CapToken** in **AuthRes**, or
- if *MerAuthFlag* in the *MerchantData* private extension to the Merchant certificate is set to TRUE, return **PANToken** (as described on page 382) in **AuthRes**, and extract the PAN from the **PANToken** when it is returned in **CapReq**.

**CapToken data**

| | |
|---|---|
| **CapToken** | **< Enc(P1, P2, CapTokenData), EncX(P1, P2, CapTokenData, PANToken ), {} >**<br>**P1** *and* **P2** *denote Payment Gateways:*<br>• **P1** is the sender.<br>• **P2** is the receiver.<br>*In this version of SET,* **P1** *and* **P2** *are always the same Payment Gateway.* |
| **CapTokenData** | **{AuthRRPID, AuthAmt, TokenOpaque}** |
| **PANToken** | *See page 382.* |
| **AuthRRPID** | *The* **RRPID** *that appeared in the corresponding* **AuthReq** *or* **AuthRevReq**. |
| **AuthAmt** | *Actual amount authorized, which may differ from Cardholder's* **PurchAmt**. |
| **TokenOpaque** | *Opaque data defined by the generating Payment Gateway.* |

**Table 10: CapToken Data**

# PANData

**Purpose**

**PANData** contains data that identifies the specific payment card account. The structure is broken out so that it can conveniently be separated and encrypted under appropriately strong encryption for sensitive data.

**PANData**

| | |
|---|---|
| **PANData** | **{PAN, CardExpiry, PANSecret, EXNonce}**<br>*Always in the extra (OAEP) slot of an encapsulation operator.* |
| **PAN** | *Primary Account Number; typically, the account number on the card.* |
| **CardExpiry** | *Expiration date on the card.* |
| **PANSecret** | *Secret value shared among Cardholder, Payment Gateway, and Cardholder CA; prevents guessing attacks on* **PAN** *in the Cardholder certificate.* |
| **EXNonce** | *A fresh nonce to foil dictionary attacks on* **PANData**. |

**Table 11: PANData**

# PANToken

**Purpose**

**PANToken**, like **PANData**, contains data that identifies the specific payment card account. **PANToken** is used when **PANSecret** is not needed to provide blinding of the data.

- **PANToken** can always be included in **CapToken**, as **CapToken** can be read only by the Payment Gateway that created it.
- **PANToken** can appear in **AuthRes** only if *MerAuthFlag* in the *MerchantData* private extension to the Merchant certificate is set to TRUE. If that criteria is met, sending **PANToken** to the Merchant is at the discretion of the Acquirer/Payment Gateway. For further discussion, see "CapToken" on page 380.

**PANToken data**

| | |
|---|---|
| **PANToken** | **{PAN, CardExpiry, EXNonce}**<br>*Always in the extra (OAEP) slot of an encapsulation operator.* |
| **PAN** | *Primary Account Number; typically, the account number on the card.* |
| **CardExpiry** | *Expiration date on the card.* |
| **EXNonce** | *A fresh nonce to foil dictionary attacks on* **PANToken**. |

**Table 12: PANToken Data**

# SaleDetail

**Purpose**

**SaleDetail** collects data associated with the sale represented by the payment card transaction. It is generated as part of the settlement process between the Merchant and the Payment Gateway. **SaleDetail** carries data from the Merchant necessary for the Payment Gateway to produce a clearing request message (for payment) that can be processed by the Acquirer or financial network for transmission to the Issuer.

**SaleDetail data**

| | |
|---|---|
| **SaleDetail** | **{[BatchID], [BatchSequenceNum], [PayRecurInd], [MerOrderNum], [AuthCharInd], [MarketSpecSaleData], [CommercialCardData], [OrderSummary], [CustomerReferenceNumber], [CustomerServicePhone], OKtoPrintPhoneInd, [SaleExtensions]}**<br>*Note: This field may appear in an* **AuthReq** *with* **CaptureNow** *set to* **TRUE** *or in the capture-related messages; when appearing in* **AuthReq**, *the fields noted as originating from* **AuthResPayload** *are not present.* |
| **BatchID** | *Identification of the settlement batch for Merchant-Acquirer accounting.* |
| **BatchSequenceNum** | *The sequence number of this item within the batch.* |
| **PayRecurInd** | *Enumerated transaction type. See page 384.* |
| **MerOrderNum** | *Merchant order number.* |
| **AuthCharInd** | *Copied from* **AuthResPayload***; see page 539.* |
| **MarketSpecSaleData** | **{[MarketSpecDataID], [MarketSpecCapData]}** |
| **CommercialCardData** | *Description of items for this capture; see page 385. Typically, this information is only included for commercial card products under special arrangement between the merchant and the customer.* |
| **OrderSummary** | *A summary description of the order.* |
| **CustomerReferenceNumber** | *A reference number assigned to the order by the Cardholder.* |
| **CustomerServicePhone** | *The merchant's customer service telephone number* |
| **OKtoPrintPhoneInd** | *A Boolean value indicating if the Issuer may print the customer service telephone number on the cardholder's statement.* |

**Table 13: SaleDetail Data**

*Continued on next page*

# **SaleDetail,** continued

**SaleDetail data** (continued)

| | |
|---|---|
| **SaleExtensions** | *The data in an extension to the sale detail ~~must~~ shall be financial and should be important for the processing of a capture request by the Payment Gateway, the financial network, or the Issuer.* |
| **MarketSpecDataID** | *Copied from* **AuthResPayload***; see page 539.* |
| **MarketSpecCapData** | **< MarketAutoCap, MarketHotelCap, MarketTransportCap >**<br>*Market-specific capture data.* |
| **MarketAutoCap** | *Automobile rental charge description. See page 387.* |
| **MarketHotelCap** | *Hotel charge description. See page 390.* |
| **MarketTransportCap** | *Passenger transport data. See page 392.* |

**Table 13: SaleDetail Data,** continued

**PayRecurInd** The following values are defined for **PayRecurInd**.

| | |
|---|---|
| unknown | *The type of transaction is unknown.* |
| singleTransaction | *The transaction consists of a single authorization and capture.* |
| recurringTransaction | *The transaction consists of multiple authorizations and captures that are repeated on a regular basis.* |
| installmentPayment | *The transaction consists of multiple authorizations and captures that are performed a fixed number of times.* |
| otherMailOrder | *Any other mail order transaction.* |

**Table 14: Enumerated Values for PayRecurInd**

*Continued on next page*

## **SaleDetail,** continued

### CommercialCardData

| | |
|---|---|
| **CommercialCardData** | **{[ChargeInfo], [MerchantLocation], [ShipFrom], [ShipTo], [ItemSeq]}** |
| **ChargeInfo** | **{[TotalFreightShippingAmount], [TotalDutyTariffAmount], [DutyTariffReference], [TotalNationalTaxAmount], [TotalLocalTaxAmount], [TotalOtherTaxAmount], [TotalTaxAmount], [MerchantTaxID], [MerchantDutyTariffRef], [CustomerDutyTariffRef], [SummaryCommodityCode], [MerchantType]}** |
| **MerchantLocation** | **Location***; see page 394* |
| **ShipFrom** | **Location***; see page 394* |
| **ShipTo** | **Location***; see page 394* |
| **ItemSeq** | **{Item +}**<br>*1 to 999 item level detail records* |
| **TotalFreightShippingAmount** | *The total amount added to the order for shipping and handling.* |
| **TotalDutyTariffAmount** | *The total amount of duties or tariff for the order.* |
| **DutyTariffReference** | *The reference number assigned to the duties or tariff for the order.* |
| **TotalNationalTaxAmount** | *The total amount of national tax (sales or VAT) applied to the order.* |
| **TotalLocalTaxAmount** | *The total amount of local tax applied to the order.* |
| **TotalOtherTaxAmount** | *The total amount of other taxes applied to the order.* |
| **TotalTaxAmount** | *The total amount of taxes applied to the order.* |
| **MerchantTaxID** | *The tax identification number of the Merchant.* |
| **MerchantDutyTariffRef** | *The duty or tariff reference number assigned to the merchant.* |
| **CustomerDutyTariffRef** | *The duty or tariff reference number assigned to the cardholder.* |

**Table 15: CommercialCardData**

*Continued on next page*

## **SaleDetail,** continued

**CommercialCardData** (continued)

| | |
|---|---|
| **SummaryCommodityCode** | *The commodity code that applies to the entire order.* |
| **MerchantType** | *The type of merchant.* |
| **Item** | **{Quantity, [UnitOfMeasureCode], Descriptor, [CommodityCode], [ProductCode], [UnitCost], [NetCost], DiscountInd, [DiscountAmount], [NationalTaxAmount], [NationalTaxRate], [NationalTaxType], [LocalTaxAmount], [OtherTaxAmount], ItemTotalCost}** |
| **Quantity** | *The quantity for the line item.* |
| **UnitOfMeasureCode** | *The unit of measure for the line item.* |
| **Descriptor** | *A description of the line item.* |
| **CommodityCode** | *The commodity code for the line item.* |
| **ProductCode** | *The product code for the line item.* |
| **UnitCost** | *The unit cost of the line item.* |
| **NetCost** | *The net cost per unit of the line item.* |
| **DiscountInd** | *Indicates if a discount was applied.* |
| **DiscountAmount** | *The amount of discount applied to the line item.* |
| **NationalTaxAmount** | *The amount of national tax (sales or VAT) applied to the line item.* |
| **NationalTaxRate** | *The national tax (sales or VAT) rate applied to the line item.* |
| **NationalTaxType** | *The type of national tax applied to the line item.* |
| **LocalTaxAmount** | *The amount of local tax applied to the line item.* |
| **OtherTaxAmount** | *The amount of other taxes applied to the line item.* |
| **ItemTotalCost** | *The total cost of the line item.* |

**Table 15: CommercialCardData,** continued

*Continued on next page*

## SaleDetail, continued

### MarketAutoCap data

| | |
|---|---|
| **MarketAutoCap** | **{[RenterName], [RentalLocation], RentalDateTime, [AutoNoShow], [RentalAgreementNumber], [ReferenceNumber], [InsuranceType], [AutoRateInfo], [ReturnLocation], ReturnDateTime, AutoCharges}** |
| **RenterName** | *The name of the person renting the vehicle.* |
| **RentalLocation** | **Location***; see page 394.* |
| **RentalDateTime** | *The date (and optionally time) the vehicle was rented.* |
| **AutoNoShow** | *Enumerated code indicating that the customer failed to show up to rent the vehicle as scheduled. See page 389.* |
| **RentalAgreementNumber** | *The rental agreement number.* |
| **ReferenceNumber** | *The rental reference number.* |
| **InsuranceType** | *The type of insurance selected by the renter.* |
| **AutoRateInfo** | **{AutoApplicableRate, [LateReturnHourlyRate], [DistanceRate], [FreeDistance], [VehicleClassCode], [CorporateID]}** |
| **ReturnLocation** | **Location***; see page 394.* |
| **ReturnDateTime** | *The date (and optionally time) the vehicle was returned.* |
| **AutoCharges** | **{RegularDistanceCharges, [LateReturnCharges], [TotalDistance], [ExtraDistanceCharges], [InsuranceCharges], [FuelCharges], [AutoTowingCharges], [OneWayDropOffCharges], [TelephoneCharges], [ViolationsCharges], [DeliveryCharges], [ParkingCharges], [OtherCharges], [TotalTaxAmount], [AuditAdjustment]}** |
| **AutoApplicableRate** | **<DailyRentalRate, WeeklyRentalRate>** |
| **LateReturnHourlyRate** | *The hourly charge for late returns.* |

**Table 16: MarketAutoCap Data**

*Continued on next page*

## **SaleDetail,** continued

**MarketAutoCap data** (continued)

| | |
|---|---|
| **DistanceRate** | *The rate charged per mile in excess of any free distance allowance.* |
| **FreeDistance** | *The distance the vehicle can travel per day without incurring an additional charge.* |
| **VehicleClassCode** | *The class of vehicle rented.* |
| **CorporateID** | *The corporate identification number that applies to the rental rate.* |
| **RegularDistanceCharges** | *The amount of charges for the rental (excluding extras classified below).* |
| **LateReturnCharges** | *The amount of charges for returning the vehicle after the date and time due back.* |
| **TotalDistance** | *The total distance the vehicle was driven.* |
| **ExtraDistanceCharges** | *The amount of the charges resulting from exceeding the free distance allowance.* |
| **InsuranceCharges** | *The amount of charges resulting from insurance.* |
| **FuelCharges** | *The amount of refueling charges.* |
| **AutoTowingCharges** | *The amount of charges resulting from towing.* |
| **OneWayDropOffCharges** | *The amount of the drop-off charges resulting from a one-way rental.* |
| **TelephoneCharges** | *The amount of charges resulting from the use of the rental vehicle telephone.* |
| **ViolationsCharges** | *The amount of charges resulting from violations assessed during the rental period.* |
| **DeliveryCharges** | *The amount of charges resulting from the delivery of the rental vehicle.* |
| **ParkingCharges** | *The amount of charges resulting from parking the rental vehicle.* |
| **OtherCharges** | *The amount of other charges not classified elsewhere.* |
| **TotalTaxAmount** | *The total amount of taxes applied to the rental.* |
| **AuditAdjustment** | *The amount the transaction was adjusted as a result of auditing by the rental company.* |
| **DailyRentalRate** | *The daily rental rate.* |
| **WeeklyRentalRate** | *The weekly rental rate.* |

**Table 16: MarketAutoCap Data,** continued

*Continued on next page*

## **SaleDetail,** continued

**AutoNoShow**

The following values are defined for **AutoNoShow**.

| | |
|---|---|
| normalVehicle | *The rental vehicle was not a special type* |
| specialVehicle | *The rental vehicle was a special type* |

**Table 17: Enumerated Values for AutoNoShow**

*Continued on next page*

## **SaleDetail,** continued

**MarketHotelCap data**

| | |
|---|---|
| **MarketHotelCap** | **{ArrivalDate, [HotelNoShow], DepartureDate, [DurationOfStay], [FolioNumber], [PropertyPhone], [CustomerServicePhone], [ProgramCode], [HotelRateInfo], HotelCharges}** |
| **ArrivalDate** | *The date the cardholder checked in (or was scheduled to check in) to the hotel.* |
| **HotelNoShow** | *Enumerated code indicating that the customer failed to check in to the hotel as scheduled. See page 391.* |
| **DepartureDate** | *The date the cardholder checked out of the hotel.* |
| **DurationOfStay** | *The number of days the cardholder stayed in the hotel.* |
| **FolioNumber** | *The folio number.* |
| **PropertyPhone** | *The telephone number of the hotel.* |
| **CustomerServicePhone** | *The customer service telephone number (of the hotel or the hotel chain).* |
| **ProgramCode** | *A code indicating the type of special program that applies to the stay.* |
| **HotelRateInfo** | **{DailyRoomRate, [DailyTaxRate]}** |
| **HotelCharges** | **{RoomCharges, [RoomTax], [PrepaidExpenses], [FoodBeverageCharges], [RoomServiceCharges], [MiniBarCharges], [LaundryCharges], [TelephoneCharges], [BusinessCenterCharges], [ParkingCharges], [MovieCharges], [HealthClubCharges], [GiftShopPurchases], [FolioCashAdvances], [OtherCharges], [TotalTaxAmount], [AuditAdjustment]}** |
| **DailyRoomRate** | *The daily room rate. This value includes applicable taxes unless the* **DailyTaxRate** *is specified.* |
| **DailyTaxRate** | *The amount of taxes applied to the daily room rate.* |

**Table 18: MarketHotelCap Data**

*Continued on next page*

## SaleDetail, continued

**MarketHotelCap data** (continued)

| | |
|---|---|
| **RoomCharges** | *The total amount charged for the room (excluding extras classified below).* |
| **RoomTax** | *The amount of tax applied to the* **RoomCharges**. |
| **PrepaidExpenses** | *The total amount of pre-paid expenses.* |
| **FoodBeverageCharges** | *The total amount of food and beverage charges.* |
| **RoomServiceCharges** | *The total amount of room service charges.* |
| **MiniBarCharges** | *The total amount of mini bar charges.* |
| **LaundryCharges** | *The total amount of laundry charges.* |
| **TelephoneCharges** | *The total amount of telephone charges.* |
| **BusinessCenterCharges** | *The total amount of business center charges.* |
| **ParkingCharges** | *The total amount of parking charges.* |
| **MovieCharges** | *The total amount of in-room movie charges.* |
| **HealthClubCharges** | *The total amount of health club charges.* |
| **GiftShopPurchases** | *The total amount of gift shop purchase charges.* |
| **FolioCashAdvances** | *The total amount of cash advances applied to the room.* |
| **OtherCharges** | *The total amount of other charges (not classified above).* |
| **TotalTaxAmount** | *The total amount of taxes applied to the bill.* |
| **Audit-Adjustment** | *The amount the transaction was adjusted as a result of auditing by the hotel.* |

**Table 18: MarketHotelCap Data,** continued

**HotelNoShow** The following values are defined for **HotelNoShow**.

| | |
|---|---|
| guaranteedLateArrival | *Indicates the reservation was made with guaranteed late arrival* |

**Table 19: Enumerated Values for HotelNoShow**

*Continued on next page*

# **SaleDetail,** continued

### MarketTransportCap data

| | |
|---|---|
| **MarketTransportCap** | **{PassengerName, DepartureDate, OrigCityAirport, [TripLegSeq], [TicketNumber], [TravelAgencyCode], [TravelAgencyName], [Restrictions]}** |
| **PassengerName** | *The name of the passenger to whom the tickets were issued.* |
| **DepartureDate** | *The departure date.* |
| **OrigCityAirport** | *The city of origin for the trip.* |
| **TripLegSeq** | **{TripLeg +}**<br>*1 to 16* **TripLeg** *records.* |
| **TicketNumber** | *The ticket number.* |
| **TravelAgencyCode** | *The travel agency code.* |
| **TravelAgencyName** | *The travel agency name.* |
| **Restrictions** | *Enumerated code indicating restrictions on refunds or changes.* *See page 393.* |
| **TripLeg** | **{DateOfTravel, CarrierCode, ServiceClass, StopOverCode, DestCityAirport, [FareBasisCode], [DepartureTax]}** |
| **DateOfTravel** | *The date of travel for this trip leg.* |
| **CarrierCode** | *The carrier code for this trip leg.* |
| **ServiceClass** | *The class of service for this trip leg.* |
| **StopOverCode** | *Enumerated code indicating whether stopovers are permitted for this trip leg.* *See page 393.* |
| **DestCityAirport** | *The destination city for this trip leg.* |
| **FareBasisCode** | *The fare basis code for this trip leg.* |
| **DepartureTax** | *The departure tax for this trip leg.* |

**Table 20: MarketTransportCap Data**

*Continued on next page*

## **SaleDetail,** continued

**Restrictions**

The following values are defined for **Restrictions**.

| | |
|---|---|
| unspecifiedRestrictions | *Unspecified restrictions* |

**Table 21: Enumerated Values for Restrictions**

**StopOverCode**

The following values are defined for **StopOverCode**.

| | |
|---|---|
| noStopOverPermitted | *No stop over permitted on this trip* |
| stopOverPermitted | *Stop over was allowed on this trip* |

**Table 22: Enumerated Values for StopOverCode**

# Location

**Location data**

**Location** is used repeatedly in **SaleDetail**, as well as in **AuthReqPayload**.

| **Location** | **{CountryCode, [City], [StateProvince], [PostalCode], [LocationID]}** |
|---|---|
| **CountryCode** | *The ISO 3166 country code for the location.* |
| **City** | *The city name of the location.* |
| **StateProvince** | *The name or abbreviation of the state or province.* |
| **PostalCode** | *The postal code of the location.* |
| **LocationID** | *An identifier that the merchant uses to specify one of its locations.* |

**Table 23: Location Data**

# RRTags

**Purpose**

**RRTags** carries message identification data; in particular, **RRPID** serves as the statistically unique identifier for a message pair.

**RRTags data**

| | |
|---|---|
| **RRTags** | **{RRPID, MerTermIDs, Date}** |
| **RRPID** | *Fresh request/response pair ID.* |
| **MerTermIDs** | **{MerchantID, [TerminalID], [AgentNum], [ChainNum], [StoreNum]}** |
| **Date** | *Current date for aging logs.* |
| **MerchantID** | *Cardholder inserts this data in* **PIHead**. *It is copied from* **MerID** *in the Merchant signature certificate.* |
| **TerminalID** | *Merchant inserts this data in* **AuthReq**. |
| **AgentNum** | *Merchant inserts this data in* **AuthReq**. |
| **ChainNum** | *Merchant inserts this data in* **AuthReq**. |
| **StoreNum** | *Merchant inserts this data in* **AuthReq**. |

**Table 24: RRTags Data**

# BatchStatus

**Purpose**

To send the status of a batch from a Payment Gateway to a Merchant or vice versa.

**BatchStatus data**

Note: The terms "debit" and "credit" reflect the impact of the transactions on the Merchant's account.

| | |
|---|---|
| **BatchStatus** | **{OpenDateTime, [ClosedWhen], BatchDetails, [BatchExtensions]}** |
| **OpenDateTime** | *The date and time the batch was opened.* |
| **ClosedWhen** | **{CloseStatus, CloseDateTime}** |
| **BatchDetails** | **{BatchTotals, [BrandBatchDetailsSeq]}** |
| **BatchExtensions** | *The data in an extension to the batch administration message ~~must~~ shall be financial and should be important for the processing of the batch administration request.* |
| **CloseStatus** | *Enumerated code indicating status of batch close. See page 397.* |
| **CloseDateTime** | *The date and time the batch was closed.* |
| **BatchTotals** | **{TransactionCountCredit, TransactionTotalAmtCredit, TransactionCountDebit, TransactionTotalAmtDebit, [BatchTotalExtensions]}** |
| **BrandBatchDetailsSeq** | **{BrandBatchDetails +}** |
| **TransactionCountCredit** | *The number of transactions that resulted in a credit to the merchant's account.* |
| **TransactionTotalAmtCredit** | *The total amount credited to the merchant's account.* |
| **TransactionCountDebit** | *The number of transactions that resulted in a debit to the merchant's account.* |
| **TransactionTotalAmtDebit** | *The total amount debited from the merchant's account.* |

**Table 25: BatchStatus Data**

*Continued on next page*

# **BatchStatus,** continued

**BatchStatus data** (continued)

| | |
|---|---|
| **BatchTotalExtensions** | *The data in an extension to the batch administration message ~~must~~ shall be financial and should be important for the processing of the batch administration request.*<br>*Note: Information regarding the processing of the request itself should appear in an extension to* **BatchAdminResData***; information regarding the status of a batch should appear in an extension to* **BatchStatus***; information regarding detail for an item within the capture batch should appear in an extension to* **TransactionDetail**. |
| **BrandBatchDetails** | **{BrandID, BatchTotals}** |
| **BrandID** | *Payment card brand (without product type).* |

**Table 25: BatchStatus Data,** continued

**CloseStatus** The following values are defined for **CloseStatus**.

| | |
|---|---|
| closedByMerchant | *The batch was closed by the Merchant* |
| closedByAcquirer | *The batch was closed by the Acquirer* |

**Table 26: Enumerated Values for CloseStatus**

# TransactionDetail

**Purpose**

To send details of the transactions in a batch from a Payment Gateway to a Merchant or vice versa.

**Transaction-Detail data**

There is one **TransactionDetail** record for each **CapReq** or **CredReq** item that has not been subsequently reversed, and one for each **CapRevReq** or **CredRevReq** item which reverses a message that has already been processed. That is, if the batch that contains a given **CapReq** or **CredReq** item is still open when the Merchant decides to reverse that message, the item is deleted from the batch (the **TransactionDetail** is discarded). However, if the batch is closed, the reversal is placed in a new batch and a **TransactionDetail** record is created for it.

| | |
|---|---|
| **TransactionDetail** | **{TransIDs, AuthRRPID, BrandID, BatchSequenceNum, [ReimbursementID], TransactionAmt, TransactionAmtType, [TransactionStatus], [TransExtensions]}** |
| **TransIDs** | *The transaction identifiers from the authorization/capture processing of the item. See page 370.* |
| **AuthRRPID** | *The* **RRPID** *that appeared in the corresponding* **AuthReq** *or* **AuthRevReq**. |
| **BrandID** | *Payment card brand (without product type).* |
| **BatchSequenceNum** | *The sequence number of this item within the batch.* |
| **ReimbursementID** | *Enumerated code indicating the type of reimbursement for the item. See page 399.* |
| **TransactionAmt** | *The amount for item of the type indicated by* **TransactionAmtType**. *The amount is always specified as a positive value.* |
| **TransactionAmtType** | *Enumerated code indicating the type of amount (credit or debit)* |
| **TransactionStatus** | *Enumerated code indicating the result of passing the transaction to the ~~next upstream~~ first non-SET system. See page 399.* |
| **TransExtensions** | *The data in an extension to the batch administration message ~~must~~ shall be financial and should be important for the processing of the batch administration request.*<br>*Note: Information regarding the processing of the request itself should appear in an extension to* **BatchAdminResData***; information regarding the status of a batch should appear in an extension to* **BatchStatus***; information regarding detail for an item within the capture batch should appear in an extension to* **TransactionDetail**. |

**Table 27: TransactionDetail Data**

*Continued on next page*

# TransactionDetail, continued

**ReimbursementID** The following values are defined for **ReimbursementID**.

| | |
|---|---|
| unspecified | *Unknown or does not appear elsewhere in this list.* |
| standard | *Standard interchange rate.* |
| keyEntered | *Interchange rate for key-entered transactions.* |
| electronic | *Interchange rate for electronic transactions.* |
| additionalData | *Interchange rate for transactions that include additional clearing data.* |
| enhancedData | *Interchange rate for transactions that include data enhancements (such as additional authorization-related data).* |
| marketSpecific | *Interchange rate for transactions within a specific market segment (such as Passenger Transport).* |

**Table 28: Enumerated Values for ReimbursementID**

**TransactionStatus** The following values are defined for **TransactionStatus**.

| | |
|---|---|
| success | *The transaction was successfully passed to the first non-SET processing system.* |
| unspecifiedFailure | *The transaction failed to pass to the first non-SET processing system.* |

**Table 29: Enumerated Values for TransactionStatus**

# Section 2
# General Flow

## Payment Flow

**Request / response message pairs**

The main flow for SET payment processing involves paired request/response messages between Cardholder and Merchant, and between Merchant and Payment Gateway. Each pair of messages supports a step in the payment process. There is a basic set of required pairs, and additional optional pairs.

**Purchase**

The **PReq/PRes** message provide the basic purchase process between the Cardholder and the Merchant. The **PRes** message may be returned immediately as in Figure 1 on page 401, or at any time later in the protocol. The data returned will depend on the stage of the protocol at which the **PRes** is returned.

**Authorization**

Authorization is performed by means of the **AuthReq**/**AuthRes** messages exchanged between the Merchant and the Payment Gateway. Authorization provides the Merchant approval by the Issuer to continue processing.

**Capture messages**

Capture may be accomplished with the **CapReq/CapRes** message pair exchanged between the Merchant and the Payment Gateway. This activity completes the purchase for the Payment Gateway, and results in the actual charge against the cardholder's account.

*Continued on next page*

# Payment Flow, continued

**Protocol summary**

Figure 1 below shows a typical example of a payment protocol flow. Optional messages are written in italics.

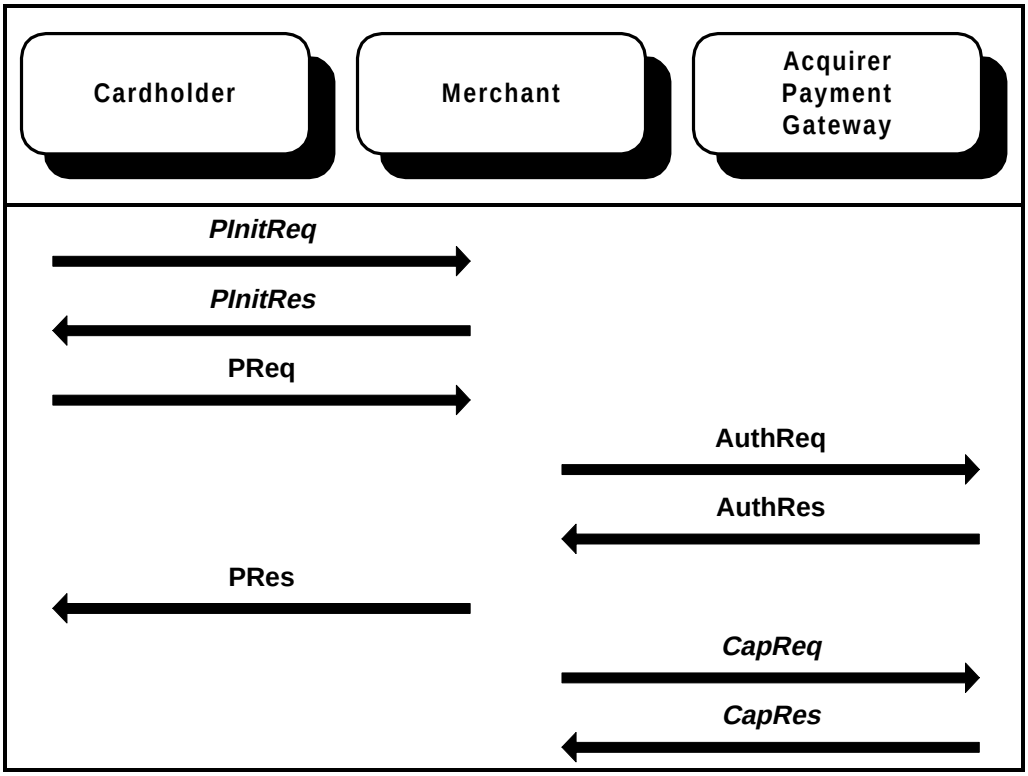


**Figure 1: Payment Protocol Flow**

*Continued on next page*

# Payment Flow, continued

### Payment flow options

Figure 2 (which continues on the next page) shows a more elaborate example of the messages which may occur in processing a transaction. Optional messages are shown in italics. Each message is described in the following sections: Cardholder/Merchant messages on page 404 and Merchant/Payment Gateway messages beginning on page 464.

In addition to the messages shown, certain messages may be reversed:

| this message: | may be reversed | by sending this message: | to which the response is: |
|---|---|---|---|
| **AuthReq** | partially or completely | **AuthRevReq** | **AuthRevRes** |
| **CapReq** | completely | **CapRevReq** | **CapRevRes** |
| **CredReq** | completely | **CredRevReq** | **CredRevRes** |

Note: Other orderings of the messages are also allowed.

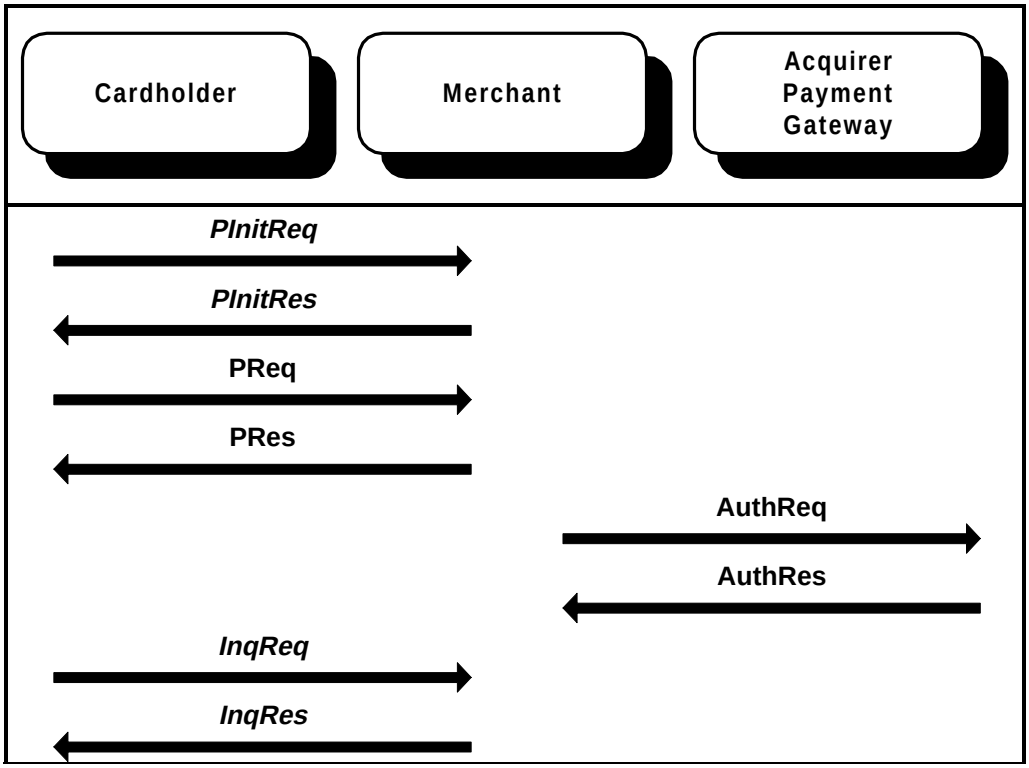


**Figure 2: Payment Flow Options**

*Continued on next page*

# **Payment Flow,** continued

**Payment flow options** (continued)

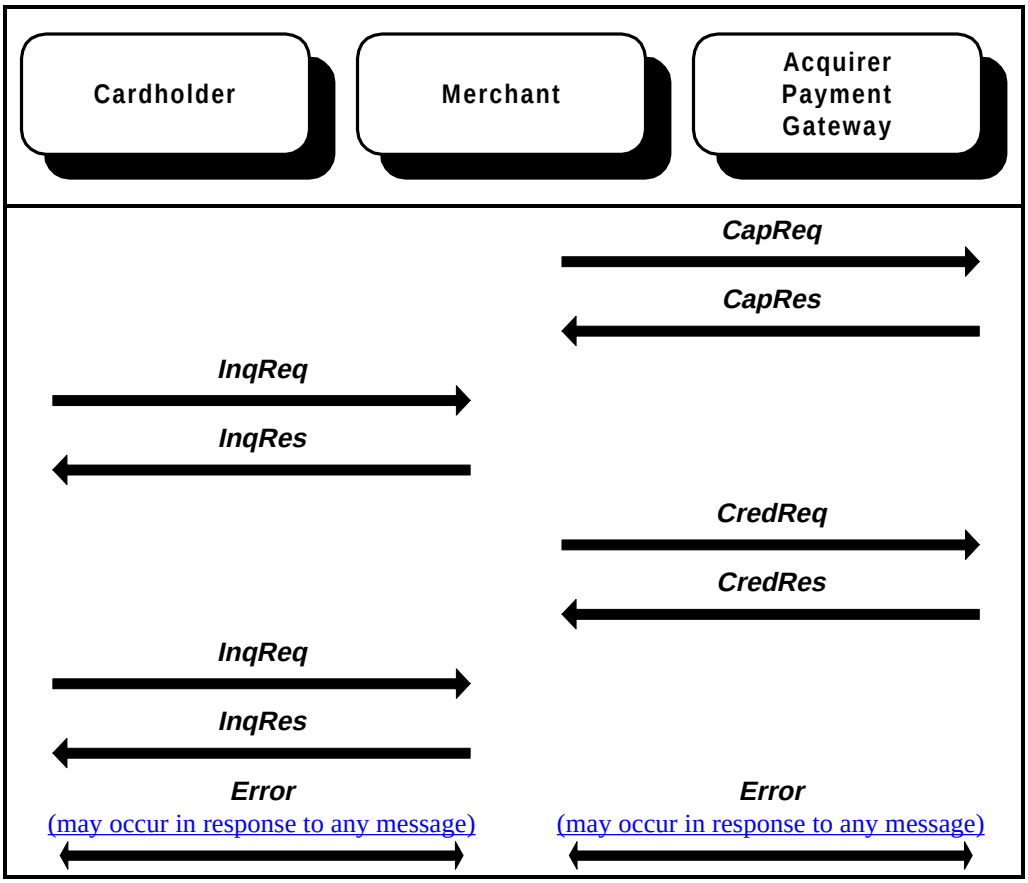


**Figure 2: Payment Flow Options (continued)**

# Chapter 2
# Cardholder/Merchant Messages

## Overview

### Introduction

Chapter 2 describes messages exchanged between the Cardholder and Merchant.

### Organization

The following sections are included:

| Section | Title | Contents | Page |
|---|---|---|---|
| 1 | Payment Initialization Request/Response Processing | Presents the **PInitReq** and **PInitRes** messages, which support initialization of the protocol, including selection of the payment card and exchange of certificates. | 405 |
| 2 | Purchase Request/Response Processing | Presents the **PReq** and **PRes** messages, which encompass the purchase transaction between the Cardholder and Merchant. | 422 |
| 3 | Inquiry Request/Response Processing | Presents the **InqReq** and **InqRes** messages, enabling the Cardholder to query the Merchant regarding the status of the transaction. | 456 |

# Section 1
# Payment Initialization Request/Response Processing

## Overview

### Introduction

The payment initialization processing consists of two messages, a request from a Cardholder to Merchant and a response from the Merchant to the Cardholder.

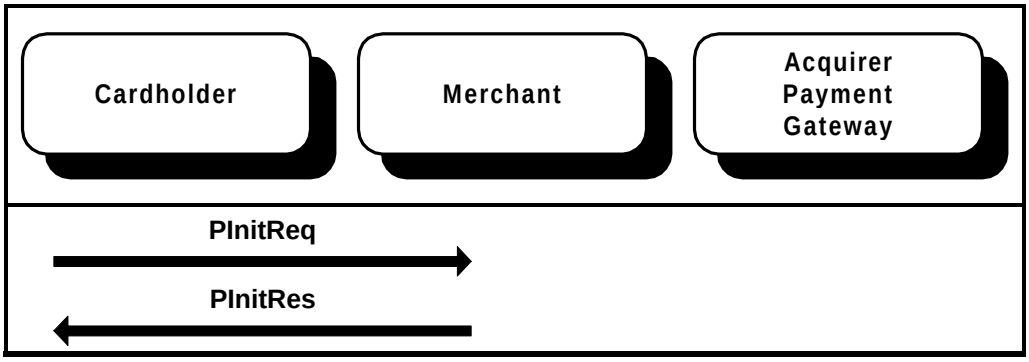


**Figure 3: PInitReq/PInitRes Message Pair**

### Purpose

The purpose of this message pair is to obtain certificates and CRLs for the Cardholder. In the absence of this message pair, this data must be obtained through some other means (such as a CD-ROM). These messages are usually preceded by a shopping phase and a SET Initiation Process.

The request message, **PInitReq**:

- identifies the Cardholder's preferred language,
- provides enough data about the cardholder's selection of a payment card to enable the Merchant software to select an appropriate Payment Gateway certificate,
- provides a local Cardholder-defined identifier for the transaction,
- sends a challenge variable to ensure freshness of the response message, and
- may include~~s~~ Thumbprints of relevant certificates and CRLs already held by the Cardholder (so that the Merchant need not re-send those certificates and CRLs).

The response message, **PInitRes**:

- contains needed certificates and CRLs (in the signature), as well as the **BrandCRLIdentifier**;
- establishes a Merchant date and an **XID**; and
- returns the Cardholder's challenge and Thumbprints, adding a challenge generated by the Merchant.

*Continued on next page*

## **Overview,** continued

**Variations**

These messages may be omitted in non-interactive environments, with the data in these messages provided by off-line mechanisms (such as CD-ROM) and the challenges omitted; there is therefore less guarantee of message freshness.

# Cardholder Prepares for PInitReq

**Transition from shopping**

At the conclusion of shopping, the cardholder will indicate readiness to begin payment. As part of the transition, an initiation process will take place:

- For off-line transactions, such as shopping from a CD-ROM, the transition will be defined by the application.
- For on-line transactions, such as shopping via the World Wide Web, the most common transition uses the process described in the *SET External Interface Guide.* (See "Related documentation" in the Preface.)

**Order record**

For the purposes of this documentation, a logical record is defined containing data from the shopping phase that applies to the payment phase. The actual implementation of collecting and passing this data is at the discretion of the application developer.

| **OrderRecord** | **{ od, purchAmt, [avsData], [marketData], [installRecurData], brandIDs, [lid-M], [ext], [extOIDs] }** |
|---|---|
| **od** | *The order description, which contains text that is displayable to the user. If supported by both the merchant and the cardholder applications, other formats (such as HTML) may be used to include formatting information. A method of using other formats is described in the SET External Interface Guide. (See "Related documentation" in the Preface.)* |
| **purchAmt** | *The amount of the transaction.* |
| **avsData** | *Cardholder billing address. See page 507.* |
| **marketData** | *Market-specific authorization data. See* **MarketSpecAuthData** *on page 507.* |
| **installRecurData** | *Data about installment or recurring payments. See page* 375. |
| **brandIDs** | *A list of brand identifiers indicating the type of payment cards accepted by the merchant* |
| **lid-M** | *A unique local identifier assigned by the merchant (optional)* |
| **ext** | *Any message extension(s) required to support additional business functions, and associated with* **od** |
| **extOIDs** | *The object identifiers that identify* **ext** |

**Table 30: OrderRecord Data**

*Continued on next page*

# Cardholder Prepares for PInitReq, continued

**Prepare for payment initialization**

The Cardholder application requires certain data to begin processing. The following processing sequence provide one method to obtain that data.

| Step | Action |
|---|---|
| 1 | Receive as input (from the initiation process or an application-defined interface):<br>***order*** — an instance of *OrderRecord* (see page 407) |
| 2 | If ***order.*od** is the same as for a recently completed **PReq**, display a message to the user warning that this appears to be a duplicate order and asking for confirmation to continue. If the user does not confirm, abort processing. |
| 3 | Display ***order.*od** to the user and provide a mechanism for the user to accept the description. If the user does not accept the description, abort processing. |
| 4 | Allow the user to select a payment card. The selection should be limited to those cards whose brand identifier appears in ***order.*brandIDs**.<br>Note: If the user enters an account number from the keyboard, the application must also obtain the expiration date; the application shall store the account number and expiration date in secure data storage. |
| 5 | If ***order.*installRecurData.recurring** is not present, continue with Step 6.<br>Otherwise, validate the following contents of ***order.*installRecurData**:<br>*recurringExpiry* — less than the expiration date of the payment card selected in Step 4<br>If errors occur during validation, advise the user that the selected payment card will expire before the final recurring payment can be authorized and give the use the option to select another card. If the user decides to select another card, continue with Step 4. |
| 6 | Allow the user to select a language to be used for the transaction.<br>Note: The choice may be determined automatically based on the user's profile. |
| 7 | Invoke "Create **PInitReq**" on page 409 with the following input:<br>***order*** — ***order***<br>***pan*** — the result of Step 4<br>***brandID*** — the **BrandID** corresponding to the result of Step 4<br>***language*** — the result of Step 6 |

# Cardholder Generates PInitReq

### Create PInitReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***order*** — an instance of *OrderRecord* (see page 407)<br>***pan*** — an instance of *PAN*<br>***brandID*** — an instance of *BrandID*<br>***language*** — an instance of *Language* |
| 2 | Recommended: Invoke "Create set of Thumbprints for request" on page 118 with the following input:<br>***brand*** — ***brandID*** without *Product* |
| 3 | Construct *PInitReq*:<br>*rrpid* — a statistically unique **RRPID**<br>*language* — ***language***<br>*lid-C* — a unique local identifier ~~(may be assigned sequentially or randomly, but should not be repeated frequently)~~<br>*lid-M* — ***order*.lid-M** (if present)<br>*chall-C* — a fresh random challenge<br>*brandID* — ***brandID***<br>*bin* — the first six digits of ***pan***<br>*thumbs* — the result of Step 2<br>*piRqExtensions* — any message extension(s) required to support additional business functions (optional) |

*Continued on next page*

# Cardholder Generates PInitReq, continued

**Create PInitReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>Store in the message database:
<table>
<tr><td>PInitReq</td><td>the result of Step 3</td></tr>
</table></td></tr>
<tr><td>5</td><td>Store in the transaction database:
<table>
<tr><td>brandID</td><td>brandID</td></tr>
<tr><td>chall-C</td><td>PInitReq.chall-C</td></tr>
<tr><td>language</td><td>language</td></tr>
<tr><td>lid-C</td><td>PInitReq.lid-C</td></tr>
<tr><td>lid-M</td><td>order.lid-M (if present)</td></tr>
<tr><td>order</td><td>order</td></tr>
<tr><td>panRef</td><td>a reference (such as a database retrieval key) to pan and its related data in secure data storage</td></tr>
</table></td></tr>
<tr><td>6</td><td>Invoke “Send Message” on page 109 with the following input:
<table>
<tr><td>recip</td><td>the Merchant</td></tr>
<tr><td>msg</td><td>the result of Step 3</td></tr>
<tr><td>ext</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
<tr><td>rrpid</td><td>PInitReq.rrpid</td></tr>
<tr><td>lid-C</td><td>PInitReq.lid-C</td></tr>
<tr><td>lid-M</td><td>order.lid-M (if present)</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Cardholder Generates PInitReq, continued

### PInitReq data

| | |
|---|---|
| **PInitReq** | **{RRPID, Language, LID-C, [LID-M], Chall-C, BrandID, BIN, [Thumbs], [PIRqExtensions]}** |
| **RRPID** | *Request/response pair ID.* |
| **Language** | *Cardholder's natural language.* |
| **LID-C** | *Local ID; convenience label generated by and for the Cardholder system.* |
| **LID-M** | *Copied from SET initiation messages (if present) described in the External Interface Guide.* |
| **Chall-C** | *Cardholder's challenge to Merchant's signature freshness.* |
| **BrandID** | *Cardholder's chosen payment card brand.* |
| **BIN** | *Bank Identification Number from the cardholder's account number (first six digits).* |
| **Thumbs** | *Lists of Certificate, CRL, and BrandCRLIdentifier Thumbprints in Cardholder's cache.* |
| **PIRqExtensions** | *Note: The purchase initialization request is not encrypted, so this extension shall not contain confidential information.* |

**Table 31: PInitReq Data**

# Merchant Processes PInitReq

**Process PInitReq**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *PInitReq*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Validate the following contents of ***msg***:<br>*rrpid* — ***hdr*.rrpid**<br>*lid-C* — ***hdr*.messageIDs.lid-C**<br>*lid-M* — ***hdr*.messageIDs.lid-M** (if present)<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *wrapperMsgMismatch* |
| 3 | Retrieve the order record (see page 407) from the shopping and initiation phases:<br>• If ***msg*.lid-M** is present, retrieve the order record based on ***msg*.lid-M**. If the record is found, designate it as ***order***; otherwise invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *unknownLID*<br>• If ***msg*.lid-M** is not present, retrieve the order record based on criteria outside the scope of SET. If the record is found, designate it as ***order***; otherwise invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *missingData* |

*Continued on next page*

# Merchant Processes PInitReq, continued

**Process PInitReq** (continued)

| Step | Action |
|---|---|
| 4 | From the trusted cache, retrieve the certificate:<br>• whose *keyUsage* includes *keyEncipherment*,<br>• whose *subject.organizationName* matches ***msg*.brandID** (as indicated by the result of "Compare **BrandID**s" on page 119), and<br>• which identifies the Payment Gateway to receive the transaction. (See first note below.)<br>If found, designate the certificate as ***cert-PE*** and its Thumbprint as ***pe-Thumb***.<br>Otherwise, stop processing and display a message to the operator indicating that corrective action must be taken to obtain a current copy of the Payment Gateway certificate.<br>Notes:<br>• A Merchant may have multiple Acquirers for a single brand and/or multiple BINs with an Acquirer. It is the Merchant's responsibility to establish the criteria to select the appropriate Payment Gateway certificate. Typically this will be a combination of the **BrandID**, Merchant BIN (which may be selected based on Cardholder BIN), and promotional card name (which, if required, must be carried in a message extension).<br>• Under normal circumstances the certificate is retrieved every 24 hours using **PCertReq** and will be available in the trusted cache. |
| 5 | Retrieve the **BrandCRLIdentifier** for the brand identified by ***msg*.brandID** (without *Product*) and designate it as ***bci***; retrieve its Thumbprint and designate it as ***bciThumb***.<br>If ***req*.mThumbs** is present and includes ***bciThumb***, set ***bci*** to NULL. |
| 6 | Store in the message database:<br>*PInitReq* \| ***msg*** |

*Continued on next page*

# Merchant Processes PInitReq, continued

**Process PInitReq** (continued)

| Step | Action |
|---|---|
| 7 | Construct *TransIDs*:<br>*lid-C* — ***msg*.lid-C**<br>*lid-M* — ***msg*.lid-M** (if present) or a unique local identifier (optional)<br>*xID* — a unique transaction identifier<br>*language* — ***msg*.language** |
| 8 | Store in the transaction database:<br>*bin* — ***msg*.bin**<br>*brand* — ***msg*.brandID** without *Product*<br>*brandID* — ***msg*.brandID**<br>*order* — ***order***<br>*pBIN* — ***cert-PE*.subject.commonName.BIN**<br>*pInit* — TRUE<br>*pInitThumbs* — ***msg*.thumbs**<br>*transIDs* — the result of Step 7<br>Designate the resulting transaction record as ***trans***. |
| 9 | Invoke "Create **PInitRes**" on page 415 with the following input:<br>***req*** — ***msg***<br>***trans*** — ***trans***<br>***cert-PE*** — ***cert-PE***<br>***peThumb*** — ***peThumb***<br>***bci*** — ***bci*** |

# Merchant Generates PInitRes

### Create PInitRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***req*** — an instance of *PInitReq*<br>***trans*** — the transaction record<br>***cert-PE*** — an instance of *Certificate*<br>***peThumb*** — an instance of *CertThumb*<br>***bci*** — an instance of *BrandCRLIdentifier* |
| 2 | Copy ***trans*.transIDs** to an instance of *TransIDs* and update the following components:<br>*pReqDate* — the current date and time |
| 3 | Construct *PInitResData*:<br>*transIDs* — the result of Step 2<br>*rrpid* — ***req*.rrpid**<br>*chall-C* — ***req*.chall-C**<br>*chall-M* — a fresh random challenge<br>*brandCRLIdentifier* — ***bci***<br>*peThumb* — ***peThumb***<br>*thumbs* — ***req*.thumbs**<br>*piRsExtensions* — any message extension(s) required to support additional business functions (optional) |

*Continued on next page*

# Merchant Generates PInitRes, continued

**Create PInitRes** (continued)

| Step | Action |
|---|---|
| 4 | Invoke "Compose *SignedData (S)*" on page 150 with the following input:<br>***s*** — the Merchant's signature certificate<br>***t*** — the result of Step 3<br>***type*** — *id-set-content-PInitResData*<br>***certs*** — ***cert-PE*** |
| 5 | Store in the message database:<br>*PInitResData* — the result of Step 3 |
| 6 | Store in the transaction database:<br>*pReqDate* — *TransIDs.pReqDate* |
| 7 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — the Cardholder<br>***msg*** — the result of Step 4<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — ***req*.rrpid**<br>***lid-C*** — ***trans*.transIDs.lid-C**<br>***lid-M*** — ***trans*.transIDs.lid-M** (if present)<br>***xID*** — ***trans*.transIDs.xID** |

*Continued on next page*

# Merchant Generates PInitRes, continued

### PInitRes data

| | |
|---|---|
| **PInitRes** | **S(M, PInitResData)** |
| **PInitResData** | **{TransIDs, RRPID, Chall-C, Chall-M, [BrandCRLIdentifier], PEThumb, [Thumbs], [PIRsExtensions]}** |
| **TransIDs** | *See page 370.* |
| **RRPID** | *Request/response pair ID.* |
| **Chall-C** | *Copied from* **PInitReq**. |
| **Chall-M** | *Merchant's challenge to Cardholder's signature freshness.* |
| **BrandCRLIdentifier** | *List of current CRLs for all CAs under a Brand CA. See page 348 in Part II.* |
| **PEThumb** | *Thumbprint of Payment Gateway key-exchange certificate.* |
| **Thumbs** | *Copied from* **PInitReq**. |
| **PIRsExtensions** | *Note: The purchase initialization response is not encrypted, so this extension shall not contain confidential information.* |

**Table 32: PInitRes Data**

# Cardholder Processes PInitRes

**Process PInitRes**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *SignedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Invoke "Verify *SignedData (S)*" on page 153 with the following input:<br>***d*** = ***msg***<br>***type*** = *id-set-content-PInitResData*<br>Designate the value of ***t*** returned as ***res***. |
| 3 | Validate the following contents of ***res***:<br>*rrpid* = ***hdr.*rrpid**<br>*transIDs.lid-C* = ***hdr.*messageIDs.lid-C**<br>*transIDs.lid-M* = ***hdr.*messageIDs.lid-M** (if present)<br>*transIDs.xID* = ***hdr.*messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 4 | From the message database, retrieve the instance of *PInitReq* whose **LID-C** matches ***res.*transIDs.lid-C**.<br>• If found, designate it as ***req***.<br>• Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownLID* |
| 5 | Retrieve the transaction record based on ***res.*transIDs.lid-C** and designate it as ***trans***. If not found, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownLID* |

*Continued on next page*

# Cardholder Processes PInitRes, continued

**Process PInitRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>6</td><td>Validate the following contents of res:
<table>
<tr><td>transIDs.lid-M</td><td>req.lid-M (if present)</td></tr>
<tr><td>rrpid</td><td>req.rrpid</td></tr>
<tr><td>chall-C</td><td>req.chall-C</td></tr>
<tr><td>thumbs</td><td>req.thumbs</td></tr>
</table>
~~If req.lid-M was not included and res.transIDs.lid-M is present, invoke "Create Error Message" on page with the following input: errorCode unknownLID~~ If errors occur during validation, invoke "Create Error Message" on page 135 with the following input based on the field that failed:
<table>
<tr><td rowspan="4">errorCode</td><td>lid-M</td><td>unknownLID</td></tr>
<tr><td>rrpid</td><td>unknownRRPID</td></tr>
<tr><td>chall-C</td><td>challengeMismatch</td></tr>
<tr><td>thumbs</td><td>thumbsMismatch</td></tr>
</table>
</td></tr>
<tr><td>7</td><td>From the trusted cache, retrieve the certificate whose:
<ul>
<li>keyUsage includes digitalSignature,</li>
<li>issuer matches msg.signerInfos[1].issuerAndSerialNumber.issuer</li>
<li>serialNumber matches msg.signerInfos[1].issuerAndSerialNumber.serialNumber.</li>
</ul>
Designate it as cert-MS.</td></tr>
<tr><td>8</td><td>If the user's configuration indicates that the merchant identity should be confirmed:
<ul>
<li>Select an instance of cert-MS.merchantData.merNameSeq where the language field matches trans.language (or if there is no matching language, the first entry).</li>
<li>Display the corresponding name field to the user.</li>
<li>If the user does not accept the displayed identity, stop processing PInitRes.</li>
</ul></td></tr>
<tr><td>9</td><td>From the trusted cache, retrieve the certificate:
<ul>
<li>whose keyUsage includes keyEncipherment and</li>
<li>whose Thumbprint matches res.peThumb.</li>
</ul>
If found, designate it as cert-PE and continue with Step 12.</td></tr>
</table>

*Continued on next page*

# Cardholder Processes PInitRes, continued

**Process PInitRes** (continued)

| Step | Action |
|---|---|
| 10 | From the untrusted cache, retrieve the certificate:<br>• whose *keyUsage* includes *keyEncipherment* and<br>• whose Thumbprint matches ***res*.peThumb**.<br>If found, designate it as ***cert-PE*** and continue with Step 11.<br>Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *missingCertificateCRLorBCI* |
| 11 | Invoke "Verify certificate" on page 129 with the following input:<br>***cert*** = ***cert-PE*** |
| 12 | Compare the following values:<br>***cert-MS*.subject.organizationalUnitName** = ***cert-PE*.subject.organizationalUnitName**<br>***cert-MS*.MerchantData.merAcquirerBIN** = ***cert-PE*.subject.commonName.BIN**<br>If the values do not match, inform the user and stop processing **PInitRes**. |
| 13 | Invoke "Compare **BrandID**s" on page 119 with the following input:<br>***hier*** = FALSE<br>***brand1*** = ***trans*.brandID**<br>***brand2*** = ***cert-MS*.subject.organizationName**<br>If errors occur during validation, inform the user and stop processing **PInitRes**. |
| 14 | Invoke "Compare **BrandID**s" on page 119 with the following input:<br>***hier*** = FALSE<br>***brand1*** = ***trans*.brandID**<br>***brand2*** = ***cert-PE*.subject.organizationName**<br>If errors occur during validation, inform the user and stop processing **PInitRes**. |
| 15 | If the cardholder has a certificate for the account identified by ***trans*.panRef**, continue with Step 16.<br>Otherwise, if ***cert-PE*.cardCertRequired** is TRUE, inform the user that the transaction cannot proceed without a cardholder certificate and stop processing. |

*Continued on next page*

# Cardholder Processes PInitRes, continued

**Process PInitRes** (continued)

| Step | Action |
|---|---|
| 16 | Store in the message database:<br>*PInitResData* — ***res*** |
| 17 | Store in the transaction database:<br>*chall-M* — ***res*.chall-M**<br>*lid-M* — ***res*.transIDs.lid-M** (if present)<br>*pReqDate* — ***res*.transIDs.pReqDate**<br>*xID* — ***res*.transIDs.xID**<br>Designate the resulting transaction record as ***trans***. |
| 18 | Delete from the message database the instance of *PInitReq* whose **rrpid** corresponds to ***res*.rrpid**. |
| 19 | Invoke "Create **PReq**" on page 426 with the following input:<br>***trans*** — ***trans***<br>***merID*** — ***cert-MS*.merchantData.merID**<br>***cert-PE*** — ***cert-PE***<br>***initRes*** — ***res*** |

# Section 2
# Purchase Request/Response Processing

## Overview

### Introduction

The purchase request/response processing consists of two messages, a request from a Cardholder to a Merchant and a response from the Merchant to the Cardholder.

These messages are at the heart of the payment protocol: This message pair embodies the payment from the Cardholder's point of view.

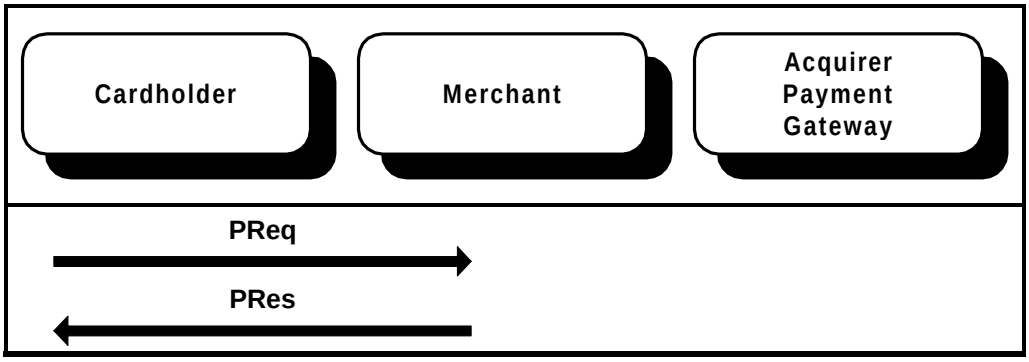


**Figure 4: PReq/PRes Message Pair**

### Structure of PReq

**PReq** is the most complex message in the protocol. It consists of two parts:

- Order Instructions (**OI**) for the Merchant, and
- Payment Instructions (**PI**) tunneled through the Merchant to the Payment Gateway.

These two items are, conceptually, separately signed. The separate signatures are combined in a provably secure optimization: a dual signature.

The Merchant is assumed to get the Order Description (**OD**) and **PurchAmt** out of band. The salted hash of **OD** and **PurchAmt**, that is, **HODInput**, is included in the **PI**. The Payment Gateway verifies that the hash tunneled through the Merchant by the Cardholder is the same as the hash provided by the Merchant in **AuthReq**.

*Continued on next page*

# Overview, continued

**PInitReq optional**

**PReq** may or may not be preceded by a **PInitReq**/**PInitRes** message pair.

**PReq variations**

Some Cardholders will not have certificates. Messages created by such Cardholders are not signed; instead the **PIHead** is linked to **OIData**. Integrity of such messages is provided ~~guaranteed~~ by:

- OAEP used with the **PI**;
- **H(PIHead)** in OAEP block (along with **PANData**);
- **H(OIData)** with **PIHead**; and
- comparison by Payment Gateway of **H(OIData)** as supplied by the Merchant with **H(OIData)** with **PIHead**.

If a Cardholder certificate is available, a dual signature is used to provide integrity and authentication for the two parts of **PReq**.

**PRes variations**

**PRes** may be returned before authorization and capture. The Merchant-Payment Gateway processing performed affects the contents of the message.

**PurchAmt**

Messages sent to the Cardholder are not encrypted. In order to avoid sending amounts in the clear, amounts in later messages are transmitted as a percentage of **PurchAmt**, the purchase amount included in the **PReq**.

For example, in **PRes** the amount authorized is conveyed by means of **AuthRatio**:

- If the full purchase amount is authorized, **AuthRatio** is 1.
- If an item is back-ordered and a lesser amount is therefore authorized, **AuthRatio** is less than 1.
- Under certain circumstances that vary according to brand policy, **AuthRatio** might be more than 1.

In each case, the Cardholder application multiplies the stored **PurchAmt** by **AuthRatio** to determine the amount authorized, so that it can be displayed to the Cardholder.

Capture and credit amounts are conveyed similarly, by means of **CapRatio** and **CreditRatio**.

# Cardholder Prepares for PReq

**Prepare for payment**

If the **PInitReq**/**PInitRes** message pair has not been processed, the Cardholder application requires certain data to begin processing the **PReq**. The following processing sequence provide one method to obtain that data.

| Step | Action |
|---|---|
| 1 | Receive as input (from an application-defined interface):<br>***order*** — an instance of *OrderRecord* (see page 407)<br>***merID*** — an instance of *MerchantID*<br>***cert-PE*** — the Payment Gateway's encryption certificate<br>Note: ***cert-PE*** must be validated prior to invocation of these processing sequence. If it is not available or cannot be validated, the **PInitReq**/**PInitRes** message pair must be exchanged. |
| 2 | If ***order.*od** is the same as for a recently completed **PReq**, display a message to the user warning that this appears to be a duplicate order and asking for confirmation to continue. If the user does not confirm, abort processing. |
| 3 | Display ***order.*od** to the user and provide a mechanism for the user to accept the description. If the user does not accept the description, abort processing. |
| 4 | Allow the user to select a payment card. The selection should be limited to those cards whose brand identifier appears in ***order.*brandIDs**.<br>Note: If the user enters an account number from the keyboard, the application must also obtain the expiration date; the application shall store the account number and expiration date in secure data storage. |
| 5 | If ***order.*installRecurData.recurring** is not present, continue with Step 6.<br>Otherwise, validate the following contents of ***order.*installRecurData**:<br>*recurringExpiry* — less than the expiration date of the payment card selected in Step 4<br>If errors occur during validation:<br>• Advise the user that the selected payment card will expire before the final recurring payment can be authorized.<br>• Prompt the user to select another payment card.<br>• Continue with Step 4. |
| 6 | If the user has a certificate for the payment card selected in Step 4, continue with Step 7.<br>Otherwise, if ***cert-PE*.cardCertRequired** is TRUE, inform the user that the transaction cannot proceed without a cardholder certificate and stop processing. |

*Continued on next page*

# Cardholder Prepares for PReq, continued

**Prepare for payment** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7</td><td>Allow the user to select a language to be used for the transaction.<br>Note: The choice may be determined automatically based on the user's profile.</td></tr>
<tr><td>8</td><td>Store in the transaction database:
<table>
<tr><td>brandID</td><td>the BrandID corresponding to the result of Step 4</td></tr>
<tr><td>chall-C</td><td><s>chall-C</s> <s>from</s> <s>PInitRes</s> a fresh random challenge</td></tr>
<tr><td>language</td><td>the result of Step 7</td></tr>
<tr><td>lid-C</td><td>a unique local identifier</td></tr>
<tr><td>order</td><td>order</td></tr>
<tr><td>panRef</td><td>a reference (such as a database retrieval key) to the result of Step 4 and its related data in secure data storage</td></tr>
<tr><td>pReqDate</td><td>the current date and time</td></tr>
<tr><td>xID</td><td>a unique transaction identifier</td></tr>
</table>
Designate the resulting transaction record as trans.<br>Note: lid-M will not appear in the transaction database if the PInitReq/PInitRes message pair has not been processed.</td></tr>
<tr><td>9</td><td>Invoke "Create PReq" with the following input:
<table>
<tr><td>trans</td><td>trans</td></tr>
<tr><td>merID</td><td>merID</td></tr>
<tr><td>cert-PE</td><td>cert-PE</td></tr>
</table></td></tr>
</table>

# Cardholder Generates PReq

### Create PReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***merID*** — an instance of *MerchantID*<br>***cert-PE*** — an instance of *Certificate*<br>***initRes*** — an instance of *PInitResData* (optional) |
| 2 | Construct *TransIDs*:<br>*lid-C* — ***trans*.lid-C**<br>*lid-M* — ***trans*.lid-M** (if present)<br>*xID* — ***trans*.xID**<br>*pReqDate* — ***trans*.pReqDate**<br>*language* — ***trans*.language** |
| 3 | Construct *HODInput*:<br>*od* — ***trans*.order.od**<br>*purchAmt* — ***trans*.order.purchAmt**<br>*odSalt* — a fresh salt<br>*installRecurData* — ***trans*.order.installRecurData** (if present)<br>*odExtensions* — ***trans*.order.ext** (if present) |
| 4 | Invoke "Compose *DetachedDigest*" on page 143 with the following input:<br>***t*** — the result of Step 3<br>***type*** — *id-set-content-HODInput* |
| 5 | Construct *Inputs*:<br>*hod* — the result of Step 4<br>*purchAmt* — ***trans*.order.purchAmt** |
| 6 | Invoke "Keyed-Hash" on page 142 with the following input:<br>***t*** — ***trans*.xID**<br>***k*** — **CardSecret** (if present) from the record in secure data storage identified by ***trans*.panRef**; otherwise, zero |

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReq** (continued)

| Step | Action |
|---|---|
| 7 | Select a supported algorithm from ***cert-PE*.tunneling**. If found, construct *AcqBackKeyData*:<br>*acqBackAlg* — the object identifier for the selected algorithm<br>*acqBackKey* — a fresh key appropriate to the selected algorithm |
| 8 | Construct *PIHead*:<br>*transIDs* — the result of Step 2<br>*inputs* — the result of Step 5<br>*merchantID* — ***merID***<br>*installRecurData* — ***trans*.order.installRecurData** (if present)<br>*transStain* — the result of Step 6<br>*swIdent* — vendor software identification (the same value used in "Send Message"; see page 109)<br>*acqBackKeyData* — the result of Step 7<br>*piExtensions* — any message extension(s) required to support additional business functions (optional) |
| 9 | Construct *OIData*:<br>*transIDs* — the result of Step 2<br>*rrpid* — a fresh statistically unique **RRPID**<br>*chall-C* — ***trans*.chall-C**<br>*hod* — the result of Step 4<br>*odSalt* — *HODInput.odSalt* (from Step 3)<br>*chall-M* — ***initRes*.chall-M** if ***initRes*** is present; otherwise omit<br>*brandID* — ***trans*.brandID** ~~(omit if **PInitReq/PInitRes** messages were exchanged)~~<br>*bin* — the first six digits of the PAN from the record in secure data storage identified by ***trans*.panRef**<br>*odExtOIDs* — ***trans*.order.extOIDs**<br>*oiExtensions* — any message extension(s) required to support additional business functions (optional) |

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReq** (continued)

| Step | Action |
|---|---|
| 10 | Invoke "Compose Linkage" on page 146 with the following input:<br>***t1*** — the result of Step 8<br>***t2*** — the result of Step 9<br>***type*** — *id-set-content-OIData* |
| 11 | Store in the message database:<br>*PIHead* — the result of Step 8<br>*OIData* — the result of Step 9 |
| 12 | Store in the transaction database:<br>*acqBackKeyData* — the result of Step 7<br>*pReqRRPID* — *oiData.rrpid*<br>Designate the resulting transaction record as ***trans***. |
| 13 | If the Cardholder has a certificate for the selected payment card, invoke "Create **PReqDualSigned**" on page 429 with the following input:<br>***trans*** — ***trans***<br>***cert-PE*** — ***cert-PE***<br>***piHead*** — the result of Step 8<br>***oiData*** — the result of Step 9<br>***pi-oiLink*** — the result of Step 10<br>Otherwise, invoke "Create **PReqUnsigned**" on page 432 with the same input. |

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReqDualSigned**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***cert-PE*** — an instance of *Certificate*<br>***piHead*** — an instance of *PIHead*<br>***oiData*** — an instance of *OIData*<br>***pi-oiLink*** — a linkage |
| 2 | Construct *PANData* from the record in secure data storage identified by ***trans.*****panRef**:<br>*pan* — PAN<br>*cardExpiry* — expiration date<br>*panSecret* — **PANSecret** |
| 3 | Invoke "Compose *EXL*" on page 176 with the following input:<br>***r*** — ***cert-PE***<br>***t*** — ***pi-oiLink***<br>***p*** — the result of Step 2<br>***type-t*** — *id-set-content-PIDualSignedTBE*<br>***type-p*** — *id-set-content-PANData* |
| 4 | Construct *PIData*:<br>*piHead* — ***piHead***<br>*panData* — ***p*** (updated in Step 3) |
| 5 | Invoke "Compose *DetachedDigest*" on page 143 with the following input:<br>***t*** — the result of Step 4<br>***type*** — *id-set-content-PIData* |

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReqDualSigned** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>6</td><td>Invoke "Compose DetachedDigest" on page 143 with the following input:
<table>
<tr><td>t</td><td>oiData</td></tr>
<tr><td>type</td><td>id-set-content-OIData</td></tr>
</table></td></tr>
<tr><td>7</td><td>Construct PI-TBS:
<table>
<tr><td>hPIData</td><td>the result of Step 5</td></tr>
<tr><td>hOIData</td><td>the result of Step 6</td></tr>
</table></td></tr>
<tr><td>8</td><td>Invoke "Compose SignedData (SO)" on page 156 with the following input:
<table>
<tr><td>s</td><td>the Cardholder's certificate</td></tr>
<tr><td>t</td><td>the result of Step 7</td></tr>
<tr><td>type</td><td>id-set-content-PI-TBS</td></tr>
</table></td></tr>
<tr><td>9</td><td>Construct PIDualSigned:
<table>
<tr><td>piSignature</td><td>the result of Step 8</td></tr>
<tr><td>exPIData</td><td>the result of Step 3</td></tr>
</table></td></tr>
<tr><td>10</td><td>Invoke "Compose Linkage" on page 146 with the following input:
<table>
<tr><td>t1</td><td>oiData</td></tr>
<tr><td>t2</td><td>the result of Step 4</td></tr>
<tr><td>type</td><td>id-set-content-PIData</td></tr>
</table></td></tr>
<tr><td>11</td><td>Construct PReqDualSigned:
<table>
<tr><td>piDualSigned</td><td>the result of Step 9</td></tr>
<tr><td>oiDualSigned</td><td>the result of Step 10</td></tr>
</table></td></tr>
<tr><td>12</td><td>Append the result of Step 11 to the tag [0].</td></tr>
</table>

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReqDualSigned** (continued)

| Step | Action |
|---|---|
| 13 | Store in the transaction database:<br>*pReqSigned* = TRUE |
| 14 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** = the Merchant<br>***msg*** = the result of Step 12<br>***ext*** = any message extension(s) required to support additional business functions (optional)<br>***rrpid*** = ***oiData*.rrpid**<br>***lid-C*** = ***piHead*.transIDs.lid-C**<br>***lid-M*** = ***piHead*.transIDs.lid-M** (if present)<br>***xID*** = ***piHead*.transIDs.xID** |

*Continued on next page*

# Cardholder Generates PReq, continued

### Create PReqUnsigned

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***cert-PE*** — an instance of Certificate<br>***piHead*** — an instance of *PIHead*<br>***oiData*** — an instance of *OIData*<br>***pi-oiLink*** — a linkage |
| 2 | Construct the following contents of *PANToken* from the record in secure data storage identified by ***trans*.panRef**:<br>*pan* — PAN<br>*cardExpiry* — expiration date |
| 3 | Invoke "Compose *EXH*" on page 180 with the following input:<br>***r*** — ***cert-PE***<br>***t*** — ***pi-oiLink***<br>***p*** — the result of Step 2<br>***type-t*** — *id-set-content-PIUnsignedTBE*<br>***type-p*** — *id-set-content-PANToken* |
| 4 | Construct *PIDataUnsigned*:<br>*piHead* — ***piHead***<br>*panToken* — ***p*** (updated in Step 3) |

*Continued on next page*

# Cardholder Generates PReq, continued

**Create PReqUnsigned** (continued)

| Step | Action |
|---|---|
| 5 | Invoke "Compose Linkage" on page 146 with the following input:<br>***t1*** — ***oiData***<br>***t2*** — the result of Step 4<br>***type*** — *id-set-content-PIDataUnsigned* |
| 6 | Construct *PReqUnsigned*:<br>*piUnsigned* — the result of Step 3<br>*oiUnsigned* — the result of Step 5 |
| 7 | Append the result of Step 6 to the tag [1]. |
| 8 | Store in the transaction database:<br>*pReqSigned* — FALSE |
| 9 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — the Merchant<br>***msg*** — the result of Step 7<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — ***oiData*.rrpid**<br>***lid-C*** — ***piHead*.transIDs.lid-C**<br>***lid-M*** — ***piHead*.transIDs.lid-M** (if present)<br>***xID*** — ***piHead*.transIDs.xID** |

*Continued on next page*

# Cardholder Generates PReq, continued

**Overall PReq data**

The purchase request message supports Cardholders with or without certificates. The **PReq** data consists of:

- Order Instructions (**OI**) for the Merchant, and
- Payment Instructions (**PI**) which is tunneled encrypted through the Merchant to the Payment Gateway.

If the Cardholder has a certificate, authentication and integrity are achieved using a dual signature **(PReqDualSigned)**. If the Cardholder is operating without a signature certificate, integrity is achieved by using hashes protected in the OAEP envelope **(PReqUnsigned)**.

| | |
|---|---|
| **PReq** | **< PReqDualSigned, PReqUnsigned >** |
| **PReqDualSigned** | *See page 434.* |
| **PReqUnsigned** | *See page 435.* |

**Table 33: PReq Data**

**PReqDualSigned data**

The **PReqDualSigned** is created by Cardholders with certificates.

| | |
|---|---|
| **PReqDualSigned** | **{PIDualSigned, OIDualSigned}** |
| **PIDualSigned** | *See* "PI (Payment Instructions)" *on page 371.* |
| **OIDualSigned** | **L(OIData, PIData)** |
| **OIData** | *See page 436.* |
| **PIData** | **{PIHead, PANData}**<br>*See page 373 for* **PIHead**.<br>*See page 381 for* **PANData**. |

**Table 34: PReqDualSigned Data**

*Continued on next page*

# Cardholder Generates PReq, continued

**PReqUnsigned data**

The **PReqUnsigned** is created by Cardholders without certificates.

| | |
|---|---|
| **PReqUnsigned** | **{PIUnsigned, OIUnsigned}** |
| **PIUnsigned** | *See* "PI (Payment Instructions)" *on page 371.* |
| **OIUnsigned** | **L(OIData, PIDataUnsigned)** |
| **OIData** | *See page 436.* |
| **PIDataUnsigned** | **{PIHead, PANToken}**<br>*See page 373 for* **PIHead**.<br>*See page 382 for* **PANToken**. |

**Table 35: PReqUnsigned Data**

*Continued on next page*

# Cardholder Generates PReq, continued

**Common PReq data**

The following data is common to both **PReqDualSigned** and **PReqUnsigned**.

**OIData**, the order data, carries data to link the purchase request to the prior shopping and ordering dialogue between the Cardholder and the Merchant.

**HODInput**, the hash of the order description, provides a secure linkage of the shopping/ordering dialogue and the purchase request. All data in the hash must be exchanged between the Cardholder and the Merchant out-of-band to SET before the purchase request is sent.

| | |
|---|---|
| **OIData** | **{TransIDs, RRPID, Chall-C, HOD, ODSalt, [Chall-M], BrandID, BIN, [ODExtOIDs], [OIExtensions]}** |
| **TransIDs** | *Copied from* **PInitRes***, if present; see page 370.* |
| **RRPID** | *Request/response pair ID.* |
| **Chall-C** | *Copied from corresponding* **PInitReq**. |
| **HOD** | **DD(HODInput)**<br>*Links* **OIData** *to* **PurchAmt** *without copying* **PurchAmt** *into* **OIData***, which would create confidentiality problems.* |
| **ODSalt** | *Copied from* **HODInput**. |
| **Chall-M** | *Merchant's challenge to Cardholder's signature freshness.* |
| **BrandID** | *Cardholder's chosen payment card brand.* |
| **BIN** | *Bank Identification Number from the cardholder's account number (first six digits).* |
| **ODExtOIDs** | *List of object identifiers from* **ODExtensions** *in the same order as the extensions appeared in* **ODExtensions**. |
| **OIExtensions** | *The data in an extension to the* **OI** *should relate to the Merchant's processing of the order.*<br>*Note: The order information is not encrypted so this extension shall not contain confidential information.* |

**Table 36: OIData**

*Continued on next page*

# Cardholder Generates PReq, continued

**Common PReq data** (continued)

| | |
|---|---|
| **HODInput** | **{OD, PurchAmt, ODSalt, [InstallRecurData], [ODExtensions]}** |
| **OD** | *The Order Description. This information is exchanged between the Cardholder and the Merchant out-of-band to SET. The contents, which are determined by the Merchant's processing requirements, will include information such as the description of the items ordered (including quantity, size, price, etc.), the shipping address, and the cardholder's billing address (if required).* |
| **PurchAmt** | *The amount of the transaction as specified by the Cardholder; this must match the value in* **PIHead** *on page 373.* |
| **ODSalt** | *Fresh Nonce generated by Cardholder to prevent dictionary attacks on* **HOD**. |
| **InstallRecurData** | *See page 377.* |
| **ODExtensions** | *The data in an extension to the* **OD** *should relate to the Merchant's processing of the order.*<br>*The information in these extensions must be independently known to both the Cardholder and Merchant.* |

**Table 36: OIData,** continued

**HODInput extension guidelines**

The hash of the order description provides a secure linkage of the shopping/ordering dialogue and the purchase request. Extensions can be included in this linkage via **ODExtensions**. The Cardholder shall indicate in **ODExtOIDs** the extensions that are included in **HODInput**, and the order that they are specified in **HODInput**, so that the merchant can compute **HOD2**.

# Merchant Prepares for PReq

**Prepare for payment**

If the **PInitReq**/**PInitRes** message pair has not been processed, the Merchant application must receive the order out-of-band to SET. When that occurs, the Merchant must process the order prior to processing the **PReq** message.

| Step | Action |
|---|---|
| 1 | Receive as input (from an application-defined interface):<br>***order*** — an instance of *OrderRecord* (see page 407) |
| 2 | Store in the transaction database:<br>*order* — ***order***<br>*pInit* — FALSE<br>Note: The application must provide an initial mechanism to retrieve the record since *lid-M* will not be contained in the **PReq**. |

# Merchant Processes PReq

### Process PReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *PReq*<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>This procedure uses the following internal variables:<br>***completionCode*** — an instance of *CompletionCode*<br>***signedPReq*** — an instance of BOOLEAN |
| 2 | Set ***completionCode*** to *orderReceived.* |
| 3 | Examine the tag at the beginning of ***msg***.<br>• If the tag is [0], set ***signedPReq*** to TRUE and continue with Step 4.<br>• Otherwise, set ***signedPReq*** to FALSE and continue with Step 11. |
| | Processing steps for **PReqDualSigned** |
| 4 | Designate:<br>• ***msg*.pReqDualSigned.piDualSigned** as ***pi***,<br>• ***msg*.pReqDualSigned.oiDualSigned.t1** as ***oiData*** and<br>• ***msg*.pReqDualSigned.oiDualSigned.t2** as ***hPIData***. |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| 5 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *serialNumber* matches ***pi*.piDualSigned.recipientInfos[1].issuerAndSerialNumber**.<br>If found, designate the certificate as ***cert-PE***.<br>Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *missingCertificateCRLorBCI* |
| 6 | Invoke "Compose *DetachedDigest*" on page 143 with the following input:<br>***t*** = ***oiData***<br>***type*** = *id-set-content-OIData*<br>Designate the result as ***hOIData***. |
| 7 | Construct *PI-TBS*:<br>*hPIData* = ***hPIData***<br>*hOIData* = ***hOIData*** |
| 8 | ~~Create a signature with the SO operator~~ Invoke "Verify *SignedData (SO)*" on page 157 with the following input:<br>***t*** = the result of Step 7<br>***d*** = ***pi*.piSignature**<br>***type*** = *id-set-content-PI-TBS*<br>~~Compare the result with ***msg*.pReqDualSigned.piDualSigned.piSignature**. If they are not equal, return an Error message with ErrorCode set to *signatureFailure*.~~ |
| 9 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* is *digitalSignature*,<br>• *issuer* matches ***msg*.signerInfos[1].issuerAndSerialNumber.issuer**, and<br>• *serialNumber* matches ***msg*.signerInfos[1].issuerAndSerialNumber.serialNumber**.<br>Designate the certificate as ***cert-CS***. |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| 10 | Validate the following contents of ***oiData***:<br>*brandID* = ***cert-CS*.subject.organizationName**<br>If errors occur during validation, set ***completionCode*** to *orderRejected* and continue with Step 24.<br>Otherwise, continue with Step 15. |
| | Processing steps for **PReqUnsigned** |
| 11 | Designate:<br>• ***msg*.pReqUnsigned.piUnsigned** as ***pi*** and<br>• ***msg*.pReqUnsigned.oiUnsigned.t1** as ***oiData*** and<br>• ***msg*.pReqUnsigned.oiUnsigned.t2** as ***hPIData***. |
| 12 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *serialNumber* matches ***pi*.piUnsigned.recipientInfos[1].issuerAndSerialNumber**.<br>If found, designate the certificate as ***cert-PE***.<br>Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *missingCertificateCRLorBCI* |
| 13 | If ***cert-PE*.cardCertRequired** is TRUE, ~~return a PRes with CompletionCode set to *signatureRequired*~~ invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *signatureRequired* |
| 14 | Invoke "Compose *DetachedDigest*" on page 143 with the following input:<br>***t*** = ***oiData***<br>***type*** = *id-set-content-OIData*<br>Designate the result as ***hOIData***. |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| | Common processing steps |
| 15 | Validate the following contents of ***oiData***:<br>*rrpid* = ***hdr*.rrpid**<br>*transIDs.lid-C* = ***hdr*.messageIDs.lid-C**<br>*transIDs.lid-M* = ***hdr*.messageIDs.lid-M** (if present)<br>*transIDs.xID* = ***hdr*.messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 16 | Retrieve the transaction record based on ***oiData*.transIDs.xID** and designate it as ***trans***. If not found, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownXID*<br>~~Verify LID-C and LID-M with record. If mismatch, return an Error message with ErrorCode set to unknownLID. Otherwise, verify Chall-M with record. If mismatch, return an Error message with ErrorCode set to challengeMismatch.~~ If ***trans*.pInit** is FALSE, continue with Step 20. |
| 17 | From the message database, retrieve the instance of *PInitResData* that corresponds to ***hdr*.messageIDs.xID**.<br>• If found, designate it as ***initRes***.<br>• Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownXID* |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| 18 | Validate the following contents of ***oiData***:<br>*chall-M* = ***initRes*.chall-M**<br>*transIDs.lid-C* = ***initRes*.transIDs.lid-C**<br>*transIDs.lid-M* = ***initRes*.transIDs.lid-M** (if present)<br>*transIDs.xID* = ***initRes*.transIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input based on the field that failed:<br>***errorCode*** = *lid-C* → *unknownLID*<br>*lid-M* → *unknownLID*<br>*chall-M* → *challengeMismatch*<br>*xID* → *unknownXID* |
| 19 | Validate the following contents of ***oiData***:<br>*brandID* = ***trans*.brandID**<br>If errors occur during validation, set ***completionCode*** to *orderRejected* and continue with Step 24. |
| 20 | If ***trans*.order.purchAmt** is less than or equal to zero, set ***completionCode*** to *meaninglessRatio* and continue with Step 24. |
| 21 | Construct fresh *HODInput*:<br>*od* = ***trans*.order.od**<br>*purchAmt* = ***trans*.order.purchAmt**<br>*odSalt* = ***oiData*.odSalt**<br>*installRecurData* = ***trans*.order.installRecurData**<br>*odExtensions* = ***trans*.order.ext** |
| 22 | Invoke "Compose *DetachedDigest*" on page 143 with the following input:<br>***t*** = the result of Step 21<br>***type*** = *id-set-content-HODInput* |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| 23 | Validate the following contents of ***oiData***:<br>***hod*** — the result of Step 22<br>If errors occur during validation, set ***completionCode*** to *orderRejected*.<br>Note: An out-of-band mechanism may also decide that the order cannot be processed (for example, the item ordered may no longer be available). In this case also, set ***completionCode*** to *orderRejected*. |
| 24 | Construct *PResPayload*:<br>*completionCode* — ***completionCode***<br>*pRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 25 | If ***trans*.pInit** is FALSE, construct *TransIDs*:<br>*lid-C* — ***oiData*.transIDs.lid-C**<br>*lid-M* — a unique local identifier (optional)<br>*xID* — ***oiData*.transIDs.xID**<br>*pReqDate* — ***oiData*.transIDs.pReqDate**<br>*language* — ***oiData*.transIDs.language**<br>Otherwise, copy ***trans*.transIDs** to an instance of *TransIDs* and update the following components:<br>*pReqDate* — ***oiData*.transIDs.pReqDate** |

*Continued on next page*

# Merchant Processes PReq, continued

**Process PReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>26</td><td>Store in the transaction database:
<table>
<tr><td>chall-C</td><td>oiData.chall-C</td></tr>
<tr><td>chall-M</td><td>oiData.chall-M</td></tr>
<tr><td>completionCode</td><td>completionCode</td></tr>
<tr><td>hod</td><td>the result of Step 22</td></tr>
<tr><td>hOIData</td><td>hOIData</td></tr>
<tr><td>oiData</td><td>oiData</td></tr>
<tr><td>peSubject</td><td>cert-PE.subject</td></tr>
<tr><td>peThumb</td><td>the Thumbprint of cert-PE</td></tr>
<tr><td>pi</td><td>pi</td></tr>
<tr><td>pReqRRPID</td><td>oiData.rrpid</td></tr>
<tr><td>signedPReq</td><td>signedPReq</td></tr>
<tr><td>signer</td><td>cert-CS.subject</td></tr>
<tr><td>transIDs</td><td>the result of Step 25</td></tr>
</table>
In addition, if the trans.pInit is FALSE, store in the transaction database:
<table>
<tr><td>bin</td><td>oiData.bin</td></tr>
<tr><td>brand</td><td>oiData.brandID without Product</td></tr>
<tr><td>brandID</td><td>oiData.brandID</td></tr>
<tr><td>pBIN</td><td>cert-PE.subject.commonName.BIN</td></tr>
</table></td></tr>
<tr><td>27</td><td>If completionCode is not orderReceived, continue with Step 28.<br>Otherwise, determine if a response to the Cardholder will be generated immediately, or if additional processing will be attempted first.<br>• If the response will be generated immediately, continue with Step 28.<br>• If the response will be deferred, continue with Step 30.</td></tr>
</table>

*Continued on next page*

# **Merchant Processes PReq,** continued

**Process PReq** (continued)

| Step | Action |
|---|---|
| | Immediate response |
| 28 | Store in the transaction database:<br>*pResPending* = FALSE<br>Designate the resulting record as ***trans***. |
| 29 | Invoke "Create **PRes**" on page 447 with the following input:<br>***trans*** = ***trans***<br>***rrpid*** = ***oiData*.rrpid**<br>***chall-C*** = ***oiData*.chall-C**<br>***pRes*** = TRUE<br>Stop processing. |
| | Delayed response |
| 30 | Store in the transaction database:<br>*pResPending* = TRUE<br>Designate the resulting record as ***trans***. |
| 31 | Invoke "Preparation for authorization" on page 500 with the following input:<br>***trans*** = ***trans*** |

# Merchant Generates PRes

### Create PRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***rrpid*** — an instance of *RRPID*<br>***chall-C*** — an instance of *Challenge*<br>***pRes*** — an instance of *BOOLEAN* |
| 2 | Create an empty *PResPayloadSeq*. For each *perAuth* in ***trans***, append ***trans*.perAuth.pResPayload** (if present) to the *PResPayloadSeq*.<br>If none is found:<br>• Construct *PResPayload*:<br>*completionCode* = *orderReceived*<br>• Append it to *PResPayloadSeq*. |
| 3 | Retrieve the **BrandCRLIdentifier** for the brand identified by ***trans*.brand** and designate it as ***bci***; retrieve its Thumbprint and designate it as ***bciThumb***.<br>If ***trans*.plnitThumbs** includes ***bciThumb***, set ***bci*** to NULL. |
| 4 | Construct *PResData*:<br>*transIDs* = ***trans*.transIDs**<br>*rrpid* = ***trans*.pReqRRPID**<br>*chall-C* = ***trans*.chall-C**<br>*brandCRLIdentifier* = ***bci***<br>*pResPayloadSeq* = the result of Step 2 |

*Continued on next page*

# Merchant Generates PRes, continued

**Create PRes** (continued)

| Step | Action |
|---|---|
| 5 | Invoke "Compose *SignedData (S)*" on page 150 with the following input:<br>***s*** — the Merchant's signature certificate<br>***t*** — the result of Step 4<br>***type*** — *id-set-content-PResData* |
| 6 | Store in the message database:<br>*PResData* — the result of Step 4 |
| 7 | Store in the transaction database:<br>*pResPending* — FALSE |
| 8 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — the Cardholder<br>***msg*** — the result of Step 5 (see Note)<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — ***rrpid***<br>***lid-C*** — ***trans*.transIDs.lid-C**<br>***lid-M*** — ***trans*.transIDs.lid-M** (if present)<br>***xID*** — ***trans*.transIDs.xID**<br>Note: If ***pRes*** is TRUE, ***msg*** is **PRes**; otherwise, ***msg*** is **InqRes**. |

*Continued on next page*

# Merchant Generates PRes, continued

### PRes data

| | |
|---|---|
| **PRes** | **S(M, PResData)** |
| **PResData** | **{TransIDs, RRPID, Chall-C, [BrandCRLIdentifier], PResPayloadSeq}** |
| **TransIDs** | *Copied from* **PReq***; see page 370.* |
| **RRPID** | *Request/response pair ID.* |
| **Chall-C** | *Copied from corresponding* **PInitReq**. |
| **BrandCRLIdentifier** | *List of current CRLs for all CAs under a Brand CA. See page 347 in Part II.* |
| **PResPayloadSeq** | **{PResPayload +}**<br>*One entry per authorization performed. Note: A reversal removes the data from* **PResPayload**.<br>*If no authorizations have been performed, a single entry with the appropriate status appears.* |
| **PResPayload** | *See page 450.* |

**Table 37: PRes Data**

*Continued on next page*

## Merchant Generates PRes, continued

**PResPayload data**

| | |
|---|---|
| **PResPayload** | **{CompletionCode, [Results], [PRsExtensions]}** |
| **CompletionCode** | *Enumerated code indicating completion status of transaction. See page 452.* |
| **Results** | **{[AcqCardMsg], [AuthStatus], [CapStatus], [CredStatusSeq]}** |
| **PRsExtensions** | *Note: The purchase response is not encrypted so this extension shall not contain confidential information.* |
| **AcqCardMsg** | *Copied from* **AuthRes**. *See page 379.* |
| **AuthStatus** | **{AuthDate, AuthCode, AuthRatio, [CurrConv]}** |
| **CapStatus** | **{CapDate, CapCode, CapRatio}**<br>*Data only appears if* **CapReq** *corresponding to the authorization has been performed. Note: A* **CapRevReq** *removes the data.* |
| **CredStatusSeq** | **{CreditStatus +}**<br>*Data only appears if* **CredReq** *corresponding to the authorization has been performed. Note: A* **CredRevReq** *removes the data.* |
| **AuthDate** | *Date of authorization; copied from* **AuthRRTags.Date** *(see page 506).* |
| **AuthCode** | *Enumerated code indicating outcome of payment authorization processing (see page 541); copied from* **AuthResPayload** *(see page 539).* |
| **AuthRatio** | **AuthReqAmt ÷ PurchAmt**<br>*For* **AuthReqAmt**, *see "*AuthReqPayload*" on page 507 or* **AuthNewAmt**, *see "*AuthRevReq*" on page 568.*<br>*For* **PurchAmt**, *see "*OIData*" on page 436. After a partial reversal, the new amount replaces the original amount.* |
| **CurrConv** | **{CurrConvRate, CardCurr}**<br>*Currency conversion information; copied from* **AuthResPayload** *(see page 539).* |

**Table 38: PResPayload Data**

*Continued on next page*

## Merchant Generates PRes, continued

**PResPayload data** (continued)

| | |
|---|---|
| **CapDate** | *Date of capture; copied from* **CapPayload** *(see page 604).* |
| **CapCode** | *Enumerated code indicating status of capture (see page 620); copied from* **CapResPayload** *(see page 619).* |
| **CapRatio** | **CapReqAmt ÷ PurchAmt**<br>*For* **CapReqAmt**, *see* "CapPayload" *on page 604. For* **PurchAmt**, *see* "OIData" *on page 436.* |
| **CreditStatus** | **{CreditDate, CreditCode, CreditRatio}**<br>*Data only appears if corresponding* **CreditReq** *has been performed. Note: A* **CredRevReq** *removes the data.* |
| **CreditDate** | *Date of credit; copied from* **CapRevOrCredReqData. CapRevOrCredReqDate** *(see page 626).* |
| **CreditCode** | *Enumerated code indicating status of credit (see page 626); copied from* **CapRevOrCredResPayload.CapRevOrCredCode** *(see page 626).* |
| **CreditRatio** | **CapRevOrCredReqAmt ÷ PurchAmt**<br>*For* **CapRevOrCredReqAmt**, *see "" on page 626.*<br>*For* **PurchAmt**, *see* "OIData" *on page 436.* |

**Table 38: PResPayload Data,** continued

*Continued on next page*

## Merchant Generates PRes, continued

**CompletionCode** The following values are defined for **CompletionCode**.

| | |
|---|---|
| meaninglessRatio | *The purchase amount for the transaction was less than or equal to zero and therefore, the ratio cannot be computed.*<br>*This value will appear in the only instance of **PResPayload** and will not be accompanied by **Results**.* |
| orderReceived | *The order has been received by the merchant, but has not been authorized.*<br>*This value will appear in the only instance of **PResPayload** and will not be accompanied by **Results**.* |
| orderRejected | *The order (or some portion of it) cannot be processed.*<br>*This value may appear in any instance of **PResPayload** (although it will usually appear in only the final instance). It may be accompanied by **Results** if the reason for the order being rejected is the result of the authorization.* |
| orderNotReceived | *There is no transaction information with matching **TransIDs** available in the merchant database.*<br>*This value will appear in the only instance of **PResPayload** and will not be accompanied by **Results**.* |
| authorizationPerformed | *The transaction has been authorized.*<br>*This value may appear in any instance of **PResPayload** and will be accompanied by **Results**.* |
| capturePerformed | *The transaction has been authorized and submitted for payment.*<br>*This value may appear in any instance of **PResPayload** and will be accompanied by **Results**.* |
| creditPerformed | *The transaction has been authorized and submitted for payment and one or more credits has been issued for the transaction.*<br>*This value may appear in any instance of **PResPayload** and will be accompanied by **Results**.* |

**Table 39: Enumerated Values for CompletionCode**

# Cardholder Processes PRes

### Process PRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *SignedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Invoke "Verify *SignedData (S)*" on page 153 with the following input:<br>***d*** — ***msg***<br>***type*** — *id-set-content-PResData*<br>Designate the value of ***t*** returned as ***res***. |
| 3 | Validate the following contents of ***res***:<br>*rrpid* — ***hdr.*rrpid**<br>*transIDs.lid-C* — ***hdr.*messageIDs.lid-C**<br>*transIDs.lid-M* — ***hdr.*messageIDs.lid-M**<br>*transIDs.xID* — ***hdr.*messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *wrapperMsgMismatch* |
| 4 | Retrieve the transaction record based on ***res*.transIDs.lid-C**.<br>• If found, designate as ***trans***.<br>• Otherwise, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *unknownLID* |

*Continued on next page*

# Cardholder Processes PRes, continued

**Process PRes** (continued)

| Step | Action |
|---|---|
| 5 | Validate the following contents of ***res***:<br>*xID* — ***trans*.xID**<br>*rrpid* — ***trans*.pReqRRPID**<br>*chall-C* — ***trans*.chall-C**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input based on the field that failed:<br>***errorCode*** = *xID* → *unknownXID*; *rrpid* → *unknownRRPID*; *chall-C* → *challengeMismatch* |
| 6 | Copy ***res*.PResPayloadSeq** to an instance of *PResPayloadSeq* and designate it as ***payloadSeq***. |
| 7 | For each entry in ***payloadSeq***:<br>• Designate the entry as ***item***.<br>• If ***item*.results.acqCardMsg** is present, perform Steps 8 through 9. |
| 8 | Decrypt ***item*.results.acqCardMsg** using the algorithm and key specified in ***trans*.acqBackKeyData**.<br>Note: If ***trans*.acqBackKeyData** is not present or if the decryption fails, ignore **AcqCardMsg**. |
| 9 | Update the following contents of ***item***:<br>*acqCardMsg* — the result of Step 8 |
| 10 | Store in the transaction database:<br>*pResPayloadSeq* — ***payloadSeq*** |
| 11 | Delete from the message database the instance of *PIHead* and the instance of *OIData* in which **transIDs.lid-C** matches ***trans*.lid-C**. |
| 12 | Format and display the data stored in Step 10. See "Displaying **PResPayload**" on page 455 for additional information. |

*Continued on next page*

# Cardholder Processes PRes, continued

## Displaying PResPayload

The display of the data contained in the **PResPayloadSeq** of **PRes** presents a challenge to the application developer. The contents can be a single **CompletionCode** or a sequence of results containing data about multiple authorizations and captures; further, each authorization and capture pair may be accompanied by zero, one, or multiple credits.

The application developer should choose a display format that is consistent with the contents of **PResPayloadSeq**. For example:

- if a single **CompletionCode** is present, the display might consist of a textual representation of the code;
- if multiple authorization and capture pairs are present, the display might consist of a grid showing the date and amount of each authorization and capture as well as the dates and amounts of any credits.

All amount ratios shall be converted to amounts before displaying the data to the user. This conversion is performed by multiplying the amount ratio by the purchase amount from the transaction database.

If currency conversion data is available for any authorization, the amount should include the transaction currency and the cardholder's billing currency.

**AuthCode**, **CapCode** and **CreditCode** values should be converted to meaningful text unless the meaning is obvious. For example, an **AuthCode** of *approved* is implied by displaying the amount of the authorization, however, *declined* is not.

If **AcqCardMsg** is present, its decrypted contents should be formatted for display to the user:

- If **acqCardText** is present, it should be included in the display of other data from the **PResPayloadSeq**.
- If **acqCardURL** is present, the user interface should include a mechanism for the user to access the site indicated in the URL.
- If **acqCardPhone** is present, it should be included in the display of other data from the **PResPayloadSeq** along with an explanation consistent with **AcqCardCode** such as "Call customer service at …".

## Future displays and updates

The data from the latest **PRes** or **InqRes** should be maintained in the transaction database and available for the user to display at any time. In addition, the user should be provided with a convenient mechanism to request an update of the data.

If a failed message is subsequently successful, do not display any data about the failed message.

The application should ensure that a reasonable amount of time separates an **InqReq** from the prior request so that users choosing "Update" repeatedly do not flood the merchant system with messages. For example, the application may require at least one minute to pass between receiving a **PRes** or **InqRes** and submitting another **InqReq**.

# Section 3
# Inquiry Request/Response Processing

## Overview

### Introduction

The inquiry message pair enables the Cardholder to inquire as to the status of a purchase transaction.

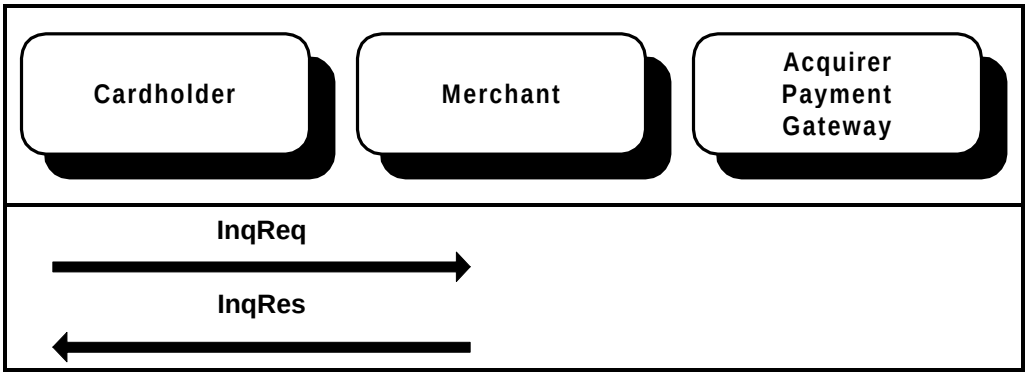


**Figure 5: InqReq/InqRes Message Pair**

### Purpose

This sequence of messages is optional. The cardholder may send the inquiry request message to the Merchant at any time after ~~**PReq**~~ **PRes**, to inquire as to the status of a transaction. Since it may be sent repeatedly, it includes its own challenge, unique to each invocation, and **TransIDs** to identify the intended transaction.

Note: Until the **PRes** is received, a cardholder who wishes to inquire about a **PReq** must re-send the **PReq** (which is an idempotent message)..

The response message is of the same format as **PRes**, but is a distinct message~~, since otherwise it would signal the Merchant's final report on the transaction~~.

The Merchant is required to verify that the certificate accompanying **InqRes** matches the certificate originally used with **PRes**. This prevents one cardholder from inquiring about another's purchases.

Cardholders without certificates do not sign inquiries, which means that the integrity of inquiry messages is not guaranteed.

### Variations

An inquiry request may be sent at any time after the receipt of a **PRes**. Multiple inquiry messages may be sent regarding the same transaction.

# Cardholder Generates InqReq

### Create InqReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record |
| 2 | Construct *TransIDs*:<br>*lid-C* — ***trans*.lid-C**<br>*lid-M* — ***trans*.lid-M** (if present)<br>*xID* — ***trans*.xID**<br>*pReqDate* — ***trans*.pReqDate**<br>*language* — ***trans*.language** |
| 3 | Construct *InqReqData*:<br>*transIDs* — the result of Step 2<br>*rrpid* — a fresh statistically unique **RRPID**<br>*chall-C2* — a fresh random challenge<br>*inqRqExtensions* — any message extension(s) required to support additional business functions (optional) |
| 4 | If ***trans*.pReqSigned** is FALSE, append the result of Step 3 to the tag [1] and continue with Step 7. |
| 5 | Invoke "Compose *SignedData (S)*" on page 150 with the following input:<br>***s*** — the Cardholder's certificate<br>***t*** — the result of Step 3<br>***type*** — *id-set-content-InqReqData* |
| 6 | Append the result of Step 5 to the tag [0]. |
| 7 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — the Merchant<br>***msg*** — the result of Step 4 or 6<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — *InqReqData.rrpid*<br>***lid-C*** — ***trans*.lid-C**<br>***lid-M*** — ***trans*.lid-M** (if present)<br>***xID*** — ***trans*.xID** |

*Continued on next page*

## Cardholder Generates InqReq, continued

### InqReq data

| | |
|---|---|
| **InqReq** | **< InqReqSigned, InqReqData >** |
| **InqReqSigned** | **S(C, InqReqData)** |
| **InqReqData** | **{TransIDs, RRPID, Chall-C2, [InqRqExtensions]}** |
| **TransIDs** | *Copied from the most recent of the following:* **PReq** *(see page 434),* **PRes** *(see page 449), or* **InqRes** *(see page 462).* |
| **RRPID** | *Request/response pair ID.* |
| **Chall-C2** | *Fresh Cardholder challenge to Merchant's signature.* |
| **InqRqExtensions** | *Note: The inquiry request is not encrypted so this extension shall not contain confidential information.* |

**Table 40: InqReq Data**

# Merchant Processes InqReq

### Process InqReq

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Receive as input:
<table>
<tr><td>hdr</td><td>an instance of MessageHeader</td></tr>
<tr><td>msg</td><td>an instance of InqReq</td></tr>
<tr><td>ext</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table>
This procedure uses the following internal variables:
<table>
<tr><td>signedInqReq</td><td>an instance of BOOLEAN</td></tr>
</table></td></tr>
<tr><td>2</td><td>Examine the tag at the beginning of msg.<br>• If the tag is [0], set signedInqReq to TRUE and continue with Step 3.<br>• Otherwise, set signedInqReq to FALSE and continue with Step 4.</td></tr>
<tr><td>3</td><td>Invoke "Verify SignedData (S)" on page 153 with the following input:
<table>
<tr><td>d</td><td>msg (without the leading tag [0])</td></tr>
<tr><td>type</td><td>id-set-content-InqReqData</td></tr>
</table>
Designate the value of t returned as req. Continue with Step 5.</td></tr>
<tr><td>4</td><td>Designate the portion of msg that follows the leading tag [1] as req.</td></tr>
<tr><td>5</td><td>Validate the following contents of req:
<table>
<tr><td>rrpid</td><td>hdr.rrpid</td></tr>
<tr><td>transIDs.lid-C</td><td>hdr.messageIDs.lid-C</td></tr>
<tr><td>transIDs.lid-M</td><td>hdr.messageIDs.lid-M</td></tr>
<tr><td>transIDs.xID</td><td>hdr.messageIDs.xID</td></tr>
</table>
If errors occur during validation, invoke "Create Error Message" on page 135 with the following input:
<table>
<tr><td>errorCode</td><td>wrapperMsgMismatch</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Merchant Processes InqReq, continued

**Process InqReq** (continued)

| Step | Action |
|---|---|
| 6 | Retrieve the transaction record whose **xID** matches ***req*.transIDs.xID** and designate it as ***trans***. If not found, continue with Step 10. |
| 7 | If ***trans*.signedPReq** is TRUE and ***signedInqReq*** is FALSE, ~~return an **InqRes** with CompletionCode set to *signatureRequired*~~ invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *signatureRequired* |
| 8 | If ***trans*.signedPReq** is FALSE, continue with Step 9.<br>From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* is *digitalSignature*,<br>• *issuer* matches ***msg*.signerInfos[1].issuerAndSerialNumber.issuer**, and<br>• *serialNumber* matches ***msg*.signerInfos[1].issuerAndSerialNumber.serialNumber**.<br>Designate the certificate as ***cert-CS***.<br>Compare the following values:<br>***trans*.signer.commonName** = ***cert-CS*.subject.commonName**<br>If mismatch, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownXID* |
| 9 | Invoke "Create **PRes**" on page 447 with the following input:<br>***trans*** = ***trans***<br>***rrpid*** = ***req*.rrpid**<br>***chall-C*** = ***req*.chall-C2**<br>***pRes*** = FALSE<br>This step completes the processing of **InqReq** when the record is found in the transaction database. |
| 10 | Invoke "Create **InqRes**" on page 461 with the following input:<br>***req*** = ***req*** |

# Merchant Generates InqRes

### Create InqRes

~~The creation of an **InqRes** is identical to the creation of a **PRes**.~~ The following processing steps are invoked when the transaction is not found in the database. If the transaction is found, "Create **PRes**" on page 447 is invoked.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***req*** — an instance of *InqReqData* |
| 2 | Construct *TransIDs*:<br>*lid-C* — ***req*.transIDs.lid-C**<br>*lid-M* — ***req*.transIDs.lid-M** (if present)<br>*xID* — ***req*.transIDs.xID**<br>*pReqDate* — ***req*.transIDs.pReqDate**<br>*paySysID* — ***req*.transIDs.paySysID** (if present)<br>*language* — ***req*.transIDs.language** |
| 3 | Construct *PResPayload*:<br>*completionCode* — *orderNotReceived* |
| 4 | Construct *PResData*:<br>*transIDs* — the result of Step 2<br>*rrpid* — ***req*.rrpid**<br>*chall-C* — ***req*.chall-C2**<br>*brandCRLIdentifier* — the **BrandCRLIdentifier** for the **BrandID** used in the **InqReq** (if available)<br>*pResPayloadSeq* — the result of Step 3 |
| 5 | Invoke "Compose *SignedData (S)*" on page 150 with the following input:<br>***s*** — the Merchant's signature certificate for the **BrandID** used in the **InqReq** is available or any supported brand if not<br>***t*** — the result of Step 4<br>***type*** — *id-set-content-PResData*<br>Note: If the merchant does not have a signature certificate for ***trans*.brandID**, any available merchant certificate may be used. |

*Continued on next page*

# Merchant Generates InqRes, continued

### Create InqRes (continued)

| Step | Action |
|---|---|
| 6 | Invoke "Send Message" on page 109 with the following input: |

| | |
|---|---|
| ***recip*** | the Cardholder |
| ***msg*** | the result of Step 5 |
| ***ext*** | any message extension(s) required to support additional business functions (optional) |
| ***rrpid*** | ***req*.rrpid** |
| ***lid-C*** | ***req*.transIDs.lid-C** |
| ***lid-M*** | ***req*.transIDs.lid-M** (if present) |
| ***xID*** | ***req*.transIDs.xID** |

### InqRes data

| | |
|---|---|
| **InqRes** | *This is identical to a* **PRes***; see page 449.* |

**Table 41: InqRes Data**

# Cardholder Processes InqRes

### Process InqRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *SignedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Invoke "Process **PRes**" on page 453 with the following input:<br>***hdr*** — ***hdr***<br>***msg*** — ***msg***<br>***ext*** — ***ext*** |

# Chapter 3
# Merchant/Payment Gateway Messages

## Overview

### Introduction

Chapter 3 describes messages exchanged between the Merchant and the Payment Gateway.

### Organization

The following sections are included:

| Section | Title | Contents | Page |
|---|---|---|---|
| 1 | Authorization Request/Response Processing | Presents the **AuthReq** and **AuthRes** messages, which support the authorization stage of the payment transaction. | 497 |
| 2 | Authorization Reversal Request/Response Processing | Presents the **AuthRevReq** and **AuthRevRes** messages, providing for the reduction or cancellation of a previous authorization. | 559 |
| 3 | Capture Request/Response Processing | Presents the **CapReq** and **CapRes** messages, which support the capture stage of the payment transaction. | 591 |
| 4 | Capture Reversal or Credit Data | Presents the data structures used by Capture Reversal, Credit, and Credit Reversal messages. The processing of these messages is discussed in the next three sections. | 626 |
| 5 | Capture Reversal Request/Response Processing | Presents the **CapRevReq** and **CapRevRes** messages, which support the reversal of previously captured transactions. | 626 |
| 6 | Credit Request/Response | Presents the **CredReq** and **CredRes** messages, which support credits against transactions which have been captured and cleared. | 626 |
| 7 | Credit Reversal Request/Response Processing | Presents the **CredRevReq** and **CredRevRes** messages, which support reversal of previously granted credits. | 626 |
| 8 | Payment Gateway Certificate Request/Response Processing | Presents the **PCertReq** and **PCertRes** messages, which enable a Merchant to request and receive Payment Gateway encryption certificates, which the Merchant uses to send encrypted messages to the Payment Gateway. | 626 |
| 9 | Batch Administration Request/Response Processing | Presents the **BatchAdminReq** and **BatchAdminRes** messages, which enable the Merchant to request the Payment Gateway to open and close capture batches, and to query their status and contents. | 626 |

*Continued on next page*

## Overview, continued

**One capture per authorization**

In SET Version 1.0, there is at most one capture for every authorization.

In general payment card processing, there are two notable exceptions to this one-to-one relationship:

- For hotel stays or car rentals, multiple authorizations may precede a single capture (for example, if the stay or rental is extended, or if incidentals such as phone calls exceed the original estimate). This does not apply to SET, as hotel/car rental transactions via SET are for prepaid hotel stays or car rentals only. If there are additional charges, a separate, non-SET authorization and capture are necessary.
- Multiple captures can be processed for a single authorization when the goods purchased are airline/railway tickets. Future versions of SET may support splitting the capture so that there is one authorization per itinerary with a separate capture record per passenger.

**Encryption**

The Payment Gateway will encrypt responses to the Merchant using the key from the most recently received Merchant key encryption certificate. A request message shall contain at most one Merchant key encryption certificate.

**Additional Payment Gateway records**

In addition to the message database and the transaction database, the Payment Gateway must maintain records of the objects listed in Table 42 on page 466 and Table 43 on page 467.

*Continued on next page*

## Overview, continued

**Payment Gateway PI records**

The data to be stored to determine whether an incoming **PI** belongs to one of the categories below is at the discretion of the Payment Gateway vendor. For example, the Payment Gateway could keep a list that contains the **XID** and **AuthRRPID** of an **AuthReq** and the SHA-1 hash of the **PI**.

The payment gateway must provide a mechanism to link the usage of a **PI** to a specific authorization request (**AuthRRPID**).

| data | conditions for inclusion | keep data: |
|---|---|---|
| used **PI**s | A **PI** (either Cardholder-created or an **authToken**) is on this list if:<br>• in response to the **AuthReq** which included it, the Payment Gateway sent an **AuthRes** with **authCode** = *approved* (and the **AuthReq** has not been completely reversed); or<br>• an initial **authCode** of *callIssuer* has been converted to *approved* (as described in "conditional PIs" below) | as long as the **PI** would otherwise be valid for an authorization or reversal request |
| conditional **PI**s | A **PI** (either Cardholder-created or an **authToken**) is on this list if, in response to the **AuthReq** that included it, the Payment Gateway sent an **AuthRes** with **authCode** = *callIssuer*, and the **AuthReq** has not been subsequently reversed.<br>If the Payment Gateway is subsequently notified that the authorization is approved (by receiving a **CapReq** including an *approvalCode*), the **PI** is moved to the list of used **PI**s. | as long as the **PI** would otherwise be valid for an authorization or reversal request |
| conditional **AuthToken**s | An **AuthToken** is on this list if the Payment Gateway included it in an **AuthRes** with **authCode** = *callIssuer*, and the **AuthReq** has not been subsequently reversed.<br>If the Payment Gateway is subsequently notified that the authorization is approved (by receiving a **CapReq** including an *approvalCode*), the **AuthToken** is moved to the list of used **PI**s. | as long as the **AuthToken** would otherwise be valid for an authorization request |
| invalid **AuthToken**s | An **AuthToken** is on this list if:<br>• the Payment Gateway included it in an **AuthRes** with **authCode** = *callIssuer*, and:<br>  • the Payment Gateway has received a new **AuthReq** with the original **PI**; or<br>  • the **AuthReq** in response to which the *callIssuer* **AuthRes** was returned has been completely reversed. | as long as the **AuthToken** would otherwise be valid for an authorization request |

**Table 42: Payment Gateway PI Records**

*Continued on next page*

## Overview, continued

**Payment Gateway RRPID records**

The Payment Gateway must keep track of when outstanding authorization requests have been captured. The data to be stored to determine whether an incoming **RRPID** corresponds to an authorization request and belongs to one of the categories below is at the discretion of the Payment Gateway vendor.

| data | conditions for inclusion | keep data: |
|---|---|---|
| captured **RRPID**s | An **RRPID** is on this list if a capture has been successfully processed (whether by **CapReq** or by **AuthReq** with **CaptureNow**) and has not been reversed. | as long as a capture request for the **RRPID** would be valid |
| fully reversed **RRPID**s | An **RRPID** is on this list if it identifies a capture reversal that was successfully processed, whether by **CapRevReq** or by **AuthRevReq** with **CaptureNow**. | as long as a capture reversal request for the **RRPID** would be valid |
| credited **RRPID**s | An **RRPID** is on this list if it identifies at least one credit that was successfully processed and has not been reversed. | as long as a capture reversal or credit reversal for the **RRPID** would be valid |
| **CaptureNow RRPID**s | An **RRPID** is on this list if it was submitted for capture in an **AuthReq** with **CaptureNow**. | as long as a capture reversal for the **RRPID** would be valid |
| **CapToken RRPID** | An **RRPID** is on this list if a **CapToken** was returned in the most recent **AuthRes** or **AuthRevRes** for this **RRPID** | as long as a capture request for the **RRPID** would be valid |

**Table 43: Payment Gateway RRPID Records**

# Section 1
# Batch Processing

## Overview

### Introduction

This section includes procedures for batch processing, which are invoked from subsequent sections. The following information is included

- BatchData
- Merchant Batch Procedures
- Shared Batch Procedures (that is, those that are invoked by both Merchant and Payment Gateway)
- Payment Gateway Batch Procedures

# BatchData

**BatchData**

For the purposes of this documentation, a logical record is defined containing data that applies to a capture batch. These records are stored in the batch database. The actual implementation of collecting and passing this data is at the discretion of the application developer.

This information must be retained while the batch is open and may need to be retained longer. For example, the Merchant or Acquirer may require that the information be retained for several days after the batch has been closed.

If information for multiple batches using the same **BatchID** appear in the database, the application developer must determine an appropriate method to retrieve the appropriate record. For example, the retrieval process may always retrieve the most recent batch for the given **BatchID** or it may only retrieve records for batches created within a specified range of days (such as the past seven days).

Note: when the Payment Gateway retrieves records, it must restrict its search to records for the Merchant making the request.

| | |
|---|---|
| **BatchData** | **{ batchID, brandAndBINSeq, availableSeqNum, [batchStatus], [remoteBatchStatus], [transactionDetailSeq]}, outstandingRequests, state, reconciled }** |
| **batchID** | identification of the settlement batch for Merchant-Acquirer accounting |
| **brandAndBINSeq** | an instance of *BrandAndBINSeq* that contains the brand and BIN combinations permitted within the batch |
| **availableSeqNum** | Optional: If **BatchSequenceNum** is a monotonically increasing number, this is an integer value of the next available value; otherwise, the implementation of this processing is at the discretion of the application developer, but it must ensure that every assigned value is unique within the batch. |
| **batchStatus** | optional: an instance of *BatchStatus* that contains the current status information for the batch |
| **remoteBatchStatus** | optional: an instance of *BatchStatus* that contains the status information for the batch supplied by the remote system |
| **transactionDetailSeq** | optional: an instance of *TransactionDetailSeq* that contains the current transaction details for the batch |
| **outstandingRequests** | a list of RRPID values that correspond to outstanding requests that affect the batch and a count of the items in that request<br>Note: The batch cannot be closed by the Merchant while any entries appear in this list. |

**Table 44: BatchData**

*Continued on next page*

# **BatchData,** continued

**BatchData** (continued)

<table>
<tr><td>state</td><td>an ENUMERATED value indicating the state of the batch:
<table>
<tr><td>open</td><td>the batch is open and available for accepting transactions</td></tr>
<tr><td>closing</td><td>the batch is in the process of being closed</td></tr>
<tr><td>closed</td><td>the batch is closed</td></tr>
<tr><td>transmitting</td><td>the data from the closed batch is being transmitted to an upstream system</td></tr>
<tr><td>transmitted</td><td>the data from the closed batch has been transmitted to an upstream system</td></tr>
</table>
Additional capture and credit requests can only be submitted against the batch when the state is open. Depending on the capabilities of the Payment Gateway, capture and credit reversals may be able to be submitted against a batch that has not been transmitted to an upstream system.<br><br>Note: transmitting and transmitted only apply to batches accumulated locally by the Payment Gateway.</td></tr>
<tr><td>reconciled</td><td>optional: an instance of BOOLEAN indicating whether the batch information has been reconciled using BatchStatus provided by the remote system.<br><br>Note: when this field is used, any operation that changes the contents of the batch must set it to FALSE so that a batch that is changed after reconciliation does not appear to be reconciled when it is not.</td></tr>
</table>

**Table 44: BatchData,** continued

*Continued on next page*

# Merchant Batch Procedures

## Overview

The Merchant uses the following batch procedures:

| Title | Function | Page |
|---|---|---|
| Determine batch identification | Determines the appropriate batch for processing an item | 472 |
| Open batch | Opens a batch and creates its **BatchData** record | 474 |
| Process batch information | Updates batch information based on response from Payment Gateway | 476 |
| Process BatchStatus | Store remote **BatchStatus** in *BatchData* record | 479 |

See also "Shared Batch Procedures" on page 480 for a description of batch procedures used by both Merchant and Payment Gateway.

*Continued on next page*

# Merchant Batch Procedures, continued

## Determine batch identification

This processing sequence applies to the Merchant and is invoked – only if the Merchant can specify batch identification – by the processing sequence that creates a request for the Payment Gateway (**AuthReq**, **AuthRevReq**, **CapReq**, etc.)

The Payment Gateway validates the proposed values or defines alternative values in "Process batch identification" (on page 487), "Update batch (add item)" (on page 493), and "Update batch (delete item)" (on page 496).

This processing sequence does the following:

- finds the batch identifier of an open batch; depending on the capabilities of the application, the batch may be opened automatically if an appropriate batch is not already open;
- optionally assigns a batch sequence number; and
- returns **BatchID** and **BatchSequenceNum**.

Note: These processing steps are written as though the representation of a batch identifier that is used internally by the application and the representation sent to the Payment Gateway are the same. However, when the range of batch identifiers used by the Payment Gateway is limited (for example, in the range of 1 to 999), the Merchant should use a different representation internally so that a batch is always uniquely identified. For example, the internal representation could include the date on which the batch was opened.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***brand*** — an instance of *BrandID* without *Product*<br>***pBIN*** — an instance of *BIN*<br>***rrpid*** — an instance of *RRPID*<br>***origBatchID*** — an instance of *BatchID* (optional) |
| 2 | If ***origBatchID*** is not specified, continue with Step 4. |
| 3 | From the batch database, retrieve the **BatchData** record that is identified by ***origBatchID*** and designate it as ***batchData***. If ***batchData*.state** is *open*:<br>• Designate ***origBatchID*** as ***batchID***.<br>• Continue with Step 6. |

*Continued on next page*

# Merchant Batch Procedures, continued

**Determine batch identification** (continued)

| Step | Action |
|---|---|
| 4 | From the batch database, retrieve a **BatchData** record where **state** is *open* and **brandAndBINSeq** contains an entry with ***brand*** and ***pBIN***. If found:<br>• Designate it as ***batchData***;<br>• Designate ***batchData*.batchID** as ***batchID***.<br>• Continue with Step 6.<br>Note: The mechanism to select a specific batch when multiple batches are open for the ***brand*** and ***pBIN*** is determined by Merchant or Acquirer policy. |
| 5 | If a batch must be explicitly opened with **BatchAdminReq**, abort processing.<br>Otherwise:<br>• invoke "Open Batch" on page 474 with appropriate input values and suspend processing until the batch is available;<br>• designate the value of ***batchID*** returned as ***batchID***; and<br>• retrieve from the batch database the **BatchData** record that is identified by ***batchID*** and designate it as ***batchData***.<br>Note: The mechanism to suspend and resume processing is at the discretion of the application developer. |
| 6 | If ***origBatchID*** is specified and designates the same batch as ***batchID***, continue with Step 7.<br>Optional: Designate an unused **batchSequenceNum** from ***batchData*.availableSeqNum** as ***sequenceNum*** and adjust ***batchData*.availableSeqNum** accordingly. |
| 7 | Update the following contents of ***batchData***:<br>*outstandingRequests* — Add ***rrpid*** if it is not already on the list and increment the item count.<br>*reconciled* — FALSE |
| 8 | Store ***batchData*** in the batch database.~~:~~ |
| 9 | Return the following:<br>***batchID*** — ***batchID***<br>***sequenceNum*** — ***sequenceNum*** |

*Continued on next page*

# Merchant Batch Procedures, continued

## Open batch

This processing sequence is invoked by "Determine batch identification" (on page 472) when the Merchant needs to open a batch. The Payment Gateway performs similar processing by invoking "Open gateway batch" on page 491.

This procedure does the following:

- If the merchant selects the batch identifier, optionally sends a **BatchAdminReq** to the payment gateway
- Stores information about the batch in the batch database using "Create **BatchData**."

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***brandAndBINSeq*** — an instance of *BrandAndBINSeq*<br>***pGwyBatchID*** — an instance of *BatchID* (optional)<br>Note: ***pGwyBatchID*** is specified if the payment gateway selects the batch identifier or if it has overridden a value selected by the merchant. |
| 2 | If ***pGwyBatchID*** is defined:<br>• Designate ***pGwyBatchID*** as ***batchID***.<br>• Continue with Step 5. |
| 3 | Designate an available value as ***batchID***.<br>Note: The Acquirer will specify when given values are available. For example:<br>• the Acquirer may require that a batch ID not be reused within a certain number of days; and/or<br>• the Acquirer may restrict the value of the batch ID to a certain number of digits. |
| 4 | If the Payment Gateway requires the batch to be explicitly opened:<br>• Invoke "Create **BatchAdminReq**" on page 626 with the following input:<br>***cert*** — the Merchant's signature certificate for any brand in ***brandAndBINSeq***<br>***batchID*** — ***batchID***<br>***operation*** — *open*<br>***brandAndBINSeq*** — ***brandAndBINSeq***<br>• Stop processing. |

*Continued on next page*

# Merchant Batch Procedures, continued

**Open batch** (continued)

| Step | Action |
|---|---|
| 5 | Invoke "Create **BatchData**" on page 481 with the following input:<br>***brandAndBINSeq*** — ***brandAndBINSeq***<br>***batchID*** — ***batchID*** |
| 6 | Return the following:<br>***batchID*** — ***batchID***<br>***batchData*** — the value of ***batchData*** returned in Step 5 |

*Continued on next page*

# Merchant Batch Procedures, continued

## Process batch information

This processing sequence applies to the Merchant. It is invoked to process the batch information in a response message from the Payment Gateway (**AuthRes**, **AuthRevRes**, **CapRes**, etc.).

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***propBatchID*** — an instance of *BatchID* (optional)<br>***propSeqNum*** — an instance of *BatchSequenceNum* (optional)<br>***batchID*** — an instance of *BatchID*<br>***seqNum*** — an instance of *BatchSequenceNum* (optional)<br>***brand*** — an instance of *BrandID* without *Product*<br>***pBIN*** — an instance of *BIN*<br>***rrpid*** — an instance of *RRPID*<br>***batchStatusSeq*** — an instance of *BatchStatusSeq* (optional)<br>***transAmt*** — an instance of *CurrencyAmount*<br>***transType*** — the message being processed; one of the following:<br>• *AuthReq*<br>• *AuthRevReq*<br>• *CapReq*<br>• *CapRevReq*<br>• *CredReq*<br>• *CredRevReq*<br>***origBatchID*** — an instance of *BatchID* (optional)<br>This procedure uses the following internal variable:<br>***sameBatch*** — an instance of *BOOLEAN* |
| 2 | If ***propBatchID*** is not specified, continue with Step 5.<br>Otherwise:<br>• From the batch database, retrieve the **BatchData** record corresponding to ***propBatchID*** and designate it as ***propBatchData***.<br>• If not found, abort processing. |

*Continued on next page*

# Merchant Batch Procedures, continued

**Process batch information** (continued)

| Step | Action |
|---|---|
| 3 | Update the following contents of ***propBatchData***:<br>*outstandingRequests* — Decrement the item count for ***rrpid*** and if the result is zero, remove ***rrpid*** from the list.<br>*reconciled* — FALSE<br>Store the updated ***propBatchData*** in the batch database. |
| 4 | If ***propBatchID*** and ***batchID*** designate the same batch, designate ***propBatchData*** as ***batchData*** and continue with Step 7. |
| 5 | From the batch database, retrieve the **BatchData** record corresponding to ***batchID***. If found, designate it as ***batchData*** and continue with Step 6.<br>Otherwise, invoke "Create **BatchData**" on page 481 with the following input:<br>***brandAndBINSeq*** — an instance of *BrandAndBINSeq* that contains ***brand*** and ***pBIN***. Optional: Include additional entries using criteria specified by the acquirer or in the merchant profile.<br>***batchID*** — ***batchID***<br>Designate the value of ***batchData*** returned as ***batchData***. |
| 6 | Update the following contents of ***batchData***:<br>*reconciled* — FALSE |
| 7 | Set ***sameBatch*** to FALSE.<br>If ***origBatchID*** is specified and ***origBatchID*** and ***batchID*** designate the same batch, set ***sameBatch*** to TRUE. |
| 8 | Invoke "Update **BatchStatus**" on page 483 with the following input:<br>***brand*** — ***brand***<br>***batchData*** — ***batchData***<br>***transAmt*** — ***transAmt***<br>***transType*** — ***transType***<br>***sameBatch*** — ***sameBatch*** |

*Continued on next page*

# Merchant Batch Procedures, continued

**Process batch information** (continued)

| Step | Action |
|---|---|
| 9 | If ***batchStatusSeq*** was specified, invoke "Process **BatchStatus**" on page 479 with the following input:<br>***batchStatusSeq*** \| ***batchStatusSeq*** |
| 10 | Return:<br>***batchData*** \| ***batchData*** |

*Continued on next page*

# Merchant Batch Procedures, continued

## Process BatchStatus

This processing sequence applies to the Merchant and is invoked by "Process batch information" (on page 476) to store **BatchStatus** that is received from the Payment Gateway.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***batchStatusSeq*** — an instance of *BatchStatusSeq* |
| 2 | For each **BatchStatus** in ***batchStatusSeq***:<br>• Designate it as ***batchStatus***.<br>• Perform Steps 3 through 5. |
| | Processing for each **BatchStatus** |
| 3 | Retrieve from the batch database the **BatchData** record that corresponds to ***batchStatus*** and designate it as ***batchData***. If not found, continue processing with the next item.<br>Note: The mechanism to match **BatchStatus** to **BatchData** is at the discretion of the application developer. |
| 4 | Update the following contents of ***batchData***:<br>*remoteBatchStatus* — ***batchStatus*** |
| 5 | Store the updated ***batchData*** in the batch database. |

# Shared Batch Procedures

## Overview

Both the Merchant and the Payment Gateway use the following batch procedures:

| Title | Function | Page |
|---|---|---|
| Create BatchData | Creates a new **BatchData** record for a newly opened batch | 481 |
| Update BatchStatus | Updates the status information in a **BatchData** record after an item has been added to or deleted from a batch | 483 |
| Update BatchTotals | Adjusts the totals within a **BatchData** record based on the type of transaction that has been added to or deleted from the batch. | 484 |

See also "Merchant Batch Procedures" on page 471 and "Payment Gateway Batch Procedures" on page 486 for a description of batch procedures used only by one entity.

*Continued on next page*

# Shared Batch Procedures, continued

## Create BatchData

This processing sequence applies to both the Merchant and the Payment Gateway. It is invoked by "Open batch" (on page 474) and "Open gateway batch" (on page 491) to create **BatchData** for the new batch, as well as by "Process batch information" (on page 476) to create **BatchData** for a batch that has been created by the Payment Gateway.

Note: The Merchant may accumulate **BatchStatus** locally or use information provided by the Payment Gateway. In the event that the information is accumulated locally, the Merchant may also choose to store the latest version received from the Payment Gateway.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***brandAndBINSeq*** — an instance of *BrandAndBINSeq*<br>***batchID*** — an instance of *BatchID* |
| 2 | If **BatchStatus** is not maintained in **BatchData**, continue with Step 7; otherwise, construct *BatchTotals*:<br>*transactionCountCredit* — 0<br>*transactionTotalAmountCredit* — a *CurrencyAmount* representing a value of zero<br>*transactionCountDebit* — 0<br>*transactionTotalAmountDebit* — a *CurrencyAmount* representing a value of zero |
| 3 | Create an empty *BrandBatchDetailsSeq* and designate it as ***brandBatchDetailsSeq***. |
| 4 | For each entry in ***brandAndBINSeq***:<br>• Designate the entry as ***brandAndBIN***.<br>• Construct *BrandBatchDetails*:<br>*brandID* — ***brandAndBIN*.brandID**<br>*batchTotals* — the result of Step 2<br>• Append the result to ***brandBatchDetailsSeq***. |
| 5 | Construct *BatchDetails*:<br>*batchTotals* — the result of Step 2<br>*brandBatchDetailsSeq* — the result of Step 4 |

*Continued on next page*

# Shared Batch Procedures, continued

**Create BatchData** (continued)

| Step | Action |
|---|---|
| 6 | Optional: Construct *BatchStatus*:<br>*openDateTime* — the current date and time<br>*batchDetails* — the result of Step 5 |
| 7 | Construct *BatchData*:<br>*batchID* — ***batchID***<br>*brandAndBINSeq* — ***brandAndBINSeq***<br>*availableSeqNum* — 1 if this field is an INTEGER; otherwise, at the discretion of the application developer<br>*batchStatus* — the result of Step 6<br>*transactionDetailSeq* — an empty instance of *TransactionDetailSeq* (optional)<br>*outstandingRequests* — an empty list<br>*state* — *open*<br>*reconciled* — FALSE |
| 8 | Store the result of Step 7 in the batch database. |
| 9 | Return the following:<br>***batchData*** — the result of Step 7 |

*Continued on next page*

# Shared Batch Procedures, continued

## Update BatchStatus

This processing sequence applies to both Payment Gateway and Merchant. It is invoked by several other batch procedures to update **BatchTotals** in the **BatchStatus**.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***brand*** — an instance of *BrandID* without *Product*<br>***batchData*** — an instance of *BatchData* (see page 469)<br>***transAmt*** — an instance of *CurrencyAmount*<br>***transType*** — the message being processed; one of the following:<br>• *AuthReq*<br>• *AuthRevReq*<br>• *CapReq*<br>• *CapRevReq*<br>• *CredReq*<br>• *CredRevReq*<br>***sameBatch*** — an instance of *BOOLEAN* (default FALSE)<br>Note: ***batchData*** is updated by these processing steps. |
| 2 | Invoke "Update **BatchTotals**" on page 484 with the following input:<br>***totals*** — ***batchData*.batchStatus.batchDetails.batchTotals**<br>***transAmt*** — ***transAmt***<br>***transType*** — ***transType***<br>***sameBatch*** — ***sameBatch***<br>Note: this will update components in ***totals***. |
| 3 | Designate the instance of ***batchData*.batchStatus.batchDetails.brandBatchDetailsSeq** whose **brandID** field matches ***brand*** as ***totals***. |
| 4 | Invoke "Update **BatchTotals**" on page 484 with the following input:<br>***totals*** — ***totals***<br>***transAmt*** — ***transAmt***<br>***transType*** — ***transType***<br>***sameBatch*** — ***sameBatch***<br>Note: this will update components in ***totals***. |
| 5 | Store the updated ***batchData*** in the batch database. |

*Continued on next page*

# Shared Batch Procedures, continued

## Update BatchTotals

This processing sequence applies to both Payment Gateway and Merchant. It is invoked by "Update **BatchStatus**" (on page 483) to adjust the totals within a **BatchData** record based on the type of transaction that has been added to or deleted from the batch.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>**totals** — an instance of *BatchTotals*<br>**transAmt** — an instance of *CurrencyAmount*<br>**transType** — the message being processed; one of the following:<br>• *AuthReq*<br>• *AuthRevReq*<br>• *CapReq*<br>• *CapRevReq*<br>• *CredReq*<br>• *CredRevReq*<br>**sameBatch** — an instance of *BOOLEAN* (default FALSE)<br>Note: **totals** is updated by these processing steps. |
| 2 | If the following conditions exist: |

| **transType** is: | and: | then: |
|---|---|---|
| *AuthReq* or *CapReq* | | Continue with Step 3 |
| *AuthRevReq* or *CapRevReq* | **sameBatch** is TRUE | Continue with Step 4 |
| *AuthRevReq* or *CapRevReq* | **sameBatch** is FALSE | Continue with Step 5 |
| *CredReq* | | Continue with Step 5 |
| *CredRevReq* | **sameBatch** is TRUE | Continue with Step 6 |
| *CredRevReq* | **sameBatch** is FALSE | Continue with Step 3 |

*Continued on next page*

# Shared Batch Procedures, continued

**Update BatchTotals** (continued)

| Step | Action |
|---|---|
| | This step applies when processing a capture request (either **CapReq** or **AuthReq** with **CaptureNow**) or when processing a **CredRevReq** against a different batch than the **CredReq**. |
| 3 | Update the following components of ***totals***:<br>*transactionCountCredit* — add one (1)<br>*transactionTotalAmtCredit* — add ***transAmt***<br>Stop processing. |
| | This step applies when processing a capture reversal against the same batch as the capture; either:<br>• **CapRevReq** against the same batch as the **CapReq**, or<br>• **AuthRevReq** against the same batch as the **AuthReq** with **CaptureNow**.<br>or when processing a **CredRevReq** against the same batch as the **CredReq**. |
| 4 | Update the following components of ***totals***:<br>*transactionCountCredit* — subtract one (1)<br>*transactionTotalAmtCredit* — subtract ***transAmt***<br>Stop processing. |
| | This step applies when processing a capture reversal against a different batch than the capture; either<br>• **CapRevReq** against a different batch than the **CapReq**, or<br>• **AuthRevReq** against a different batch than the **AuthReq** with **CaptureNow**. |
| 5 | Update the following components of ***totals***:<br>*transactionCountDebit* — add one (1)<br>*transactionTotalAmtDebit* — add ***transAmt***<br>Stop processing. |
| | This step applies when processing a **CredReq**. |
| 6 | Update the following components of ***totals***:<br>*transactionCountDebit* — subtract one (1)<br>*transactionTotalAmtDebit* — subtract ***transAmt*** |

# Payment Gateway Batch Procedures

## Overview

The Payment Gateway uses the following batch procedures:

| Title | Function | Page |
|---|---|---|
| Process batch identification | Determine a **BatchID** or confirm one supplied by the Merchant | 487 |
| Open gateway batch | Open a batch using information supplied by the Merchant or determined by the Payment Gateway | 491 |
| Update batch (add item) | Add an item to a batch | 493 |
| Update batch (delete item) | Delete an item from a batch | 496 |

See also "Shared Batch Procedures" on page 480 for a description of batch procedures used by both the Merchant and the Payment Gateway

*Continued on next page*

# Payment Gateway Batch Procedures, continued

## Process batch identification

This processing sequence applies to the Payment Gateway. It is invoked when processing a Merchant request (**AuthReq**, **AuthRevReq**, **CapReq**, etc.) to determine the **BatchID** or to confirm the one supplied by the Merchant.

The Payment Gateway will invoke "Update batch (add item)" (on page 493) or "Update batch (delete item)" (on page 496) to determine or confirm the **BatchSequenceNum**. The Merchant may have invoked "Determine batch identification" on page 472 to determine both **BatchID** and **BatchSequenceNum**.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***brand*** — an instance of *BrandID* without *Product*<br>***pBIN*** — an instance of *BIN*<br>***rrpid*** — an instance of *RRPID*<br>***mBatchID*** — an instance of *BatchID* (optional)<br>***transType*** — the message being processed; one of the following:<br>• *AuthReq*<br>• *AuthRevReq*<br>• *CapReq*<br>• *CapRevReq*<br>• *CredReq*<br>• *CredRevReq*<br>***origBatchID*** — an instance of *BatchID* (optional)<br>This procedure uses the following internal variables:<br>***capCode*** — an instance of *CapCode*<br>***reversalFlag*** — an instance of *BOOLEAN*<br>***sameBatch*** — an instance of *BOOLEAN* |
| 2 | Set ***capCode*** to *success* and ***sameBatch*** to FALSE.<br>If ***transType*** is: • *AuthReq*, • *CapReq*, or • *CredReq* — set ***reversalFlag*** to: FALSE<br>If ***transType*** is: • *AuthRevReq*, • *CapRevReq*, or • *CredRevReq* — set ***reversalFlag*** to: TRUE |
| 3 | If ***reversalFlag*** is FALSE, continue with Step 8. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Process batch identification** (continued)

| Step | Action |
|---|---|
| 4 | Retrieve from the batch database the **BatchData** record that is identified by ***origBatchID*** and designate it as ***origBatchData***. If not found, continue with Step 7. |
| 5 | If ***origBatchData*.state** is *open* or *closing*:<br>• Designate ***origBatchID*** as ***batchID***.<br>• Designate ***origBatchData*** as ***batchData***.<br>• Set ***sameBatch*** to TRUE.<br>• Continue with Step 18. |
| 6 | Optional: If **origBatchID.state** is not *transmitting* or *transmitted*:<br>• Designate ***origBatchID*** as ***batchID***.<br>• Designate ***origBatchData*** as ***batchData***.<br>• Set ***sameBatch*** to TRUE.<br>• Set ***batchData*.state** to *closing*.<br>• Continue with Step 18.<br>Note: If a reversal is submitted after a batch has been closed, but before the information has been transmitted to an upstream system, the Payment Gateway should temporarily reopen the batch to remove the item(s). |
| 7 | If a reversal can only be submitted against the same batch as the original item, set ***capCode*** to *batchClosed* and continue with Step 19. |
| 8 | If only the Payment Gateway may determine the **BatchID**, continue with Step 15.<br>If either the Payment Gateway or the Merchant may determine the **BatchID** and ***mBatchID*** is omitted, continue with Step 15. |
| | Merchant controls the selection of **BatchID** |
| 9 | If ***mBatchID*** is omitted, set ***capCode*** to *batchDataNeeded* and continue with Step 19. |
| 10 | Retrieve from the batch database the **BatchData** record that corresponds to ***mBatchID*** and designate it as ***batchData***. If found, continue with Step 13. |
| 11 | If batches must be explicitly opened using **BatchAdminReq**, set ***capCode*** to *batchUnknown* and continue with Step 15. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Process batch identification** (continued)

| Step | Action |
|---|---|
| 12 | Invoke "Open gateway batch" on page 491 with the following input:<br>***batchID*** = ***mBatchID***<br>***brand*** = ***brand***<br>***pBIN*** = ***pBIN***<br>If the result is *success*, designate the value of ***batchData*** returned as ***batchData***. Otherwise, set ***capCode*** to the result and continue with Step 15. |
| 13 | If ***batchData*.state** is *open*:<br>• If ***batchData*.brandAndBINSeq** contains an entry with ***brand*** and ***pBIN***, designate ***mBatchID*** as ***batchID*** and continue with Step 18.<br>• Otherwise, set ***capCode*** to *batchWrong*.<br>Otherwise, set ***capCode*** to *batchClosed*. |
| | Payment gateway controls the selection of **BatchID** |
| 14 | If the Payment Gateway is not permitted to override the **BatchID** specified by the Merchant, continue with Step 19. |
| 15 | From the batch database, retrieve a **BatchData** record where **state** is *open* and **brandAndBINSeq** contains an entry with ***brand*** and ***pBIN***.<br>If found:<br>• Designate it as ***batchData***.<br>• Designate ***batchData*.batchID** as ***batchID***.<br>• Continue with Step 18.<br>Note: The mechanism to select a specific batch when multiple batches are open for the ***brand*** and ***pBIN*** is determined by acquirer policy. |
| 16 | If batches must be explicitly opened using **BatchAdminReq**, set ***capCode*** to *batchUnknown* and continue with Step 19. |
| 17 | Invoke "Open gateway batch" on page 491 with the following input:<br>***brand*** = ***brand***<br>***pBIN*** = ***pBIN***<br>If the result is *success*, designate the value of ***batchData*** returned as ***batchData*** and the value of ***batchData*.batchID** as ***batchID***.<br>Otherwise, set ***capCode*** to the result and continue with Step 19. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Process batch identification** (continued)

| Step | Action |
|---|---|
| 18 | Update the following contents of ***batchData***:<br>*outstandingRequests* — Add ***rrpid*** if it is not already on the list and increment the item count.<br>*reconciled* — FALSE<br>Store the updated ***batchData*** in the batch database.± |
| 19 | Return the following:<br>***batchID*** — ***batchID***<br>***capCode*** — ***capCode***<br>***batchData*** — ***batchData***<br>***sameBatch*** — ***sameBatch***<br>Note: If ***transType*** is not *CapReq*, the caller must map the value of ***capCode*** to the appropriate response values. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

## Open gateway batch

This processing sequence applies to the Payment Gateway. It is invoked by "Process batch identification" (on page 487) to open a new batch if needed during processing. If the Payment Gateway is not permitted to open a new batch without explicit instructions, it will instead have returned a **capCode** that indicates that the Merchant must submit a **BatchAdminReq**.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***batchID*** — an instance of *BatchID* (optional)<br>***brand*** — an instance of *BrandID* without *Product* (optional)<br>***pBIN*** — an instance of *BIN* (optional)<br>***brandAndBINSeq*** — an instance of *BrandAndBINSeq* (optional)<br>This procedure uses the following internal variables:<br>***status*** — an enumerated field with a value of *success* or *failure* |
| 2 | If ***batchID*** is specified, verify that the value is permissible according to acquirer policy and is available. If not, set ***status*** to *invalidBatchID* and continue with Step 7.<br>Note: The Acquirer will designate when given values are available. For example:<br>• the Acquirer may require that a batch ID not be reused within a certain number of days; and/or<br>• the Acquirer may restrict the value of the batch ID to a certain number of digits. |
| 3 | If ***batchID*** is not specified, designate an available value as ***batchID***. |
| 4 | If ***brandAndBINSeq*** is specified, continue with Step 6.<br>Otherwise, construct an instance of *BrandAndBINSeq* that contains ***brand*** and ***pBIN*** (if specified). Designate it as ***brandAndBINSeq***.<br>Optional: Add entries to ***brandAndBINSeq*** using criteria specified by the acquirer or in the merchant profile. |
| 5 | If batches are not accumulated locally:<br>• Process batch open via existing payment card financial network.<br>• Set ***status*** based on the result.<br>• If ***status*** is not *success*, continue with Step 7. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Open gateway batch** (continued)

| Step | Action |
|---|---|
| 6 | Invoke "Create **BatchData**" on page 481 with the following input:<br>***brandAndBINSeq*** — ***brandAndBINSeq***<br>***batchID*** — ***batchID***<br>Set ***status*** to *success*. |
| 7 | Return a status of ***status*** and the following:<br>***batchData*** — the value of ***batchData*** returned in Step 6 |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

## Update batch (add item)

This processing sequence applies to the Payment Gateway. It is invoked when processing a Merchant request (**AuthReq**, **AuthRevReq**, **CapReq**, etc.) to add an item to a batch that is being accumulated locally. It will determine the **BatchSequenceNum** or confirm the one supplied by the Merchant.

The Payment Gateway previously invoked "Process batch identification" on page 487 to determine or confirm the **BatchID**. The Merchant may have invoked "Determine batch identification" on page 472 to determine both **BatchID** and **BatchSequenceNum**.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***rrpid*** — an instance of *RRPID*<br>***batchData*** — an instance of *BatchData* (see page 469)<br>***sequenceNum*** — an instance of *BatchSequenceNum* (optional)<br>***transAmt*** — an instance of *CurrencyAmount*<br>***transType*** — the message being processed; one of the following:<br>• *AuthReq*<br>• *AuthRevReq*<br>• *CapReq*<br>• *CapRevReq*<br>• *CredReq*<br>• *CredRevReq*<br>***payload*** — an instance of one of the following:<br>• *AuthReqPayload*<br>• *AuthRevReqData*<br>• *CapPayload*<br>• *CapRevOrCredReqItem*<br>corresponding to ***transType***.<br>This procedure uses the following internal variables:<br>***transAmtType*** — an instance of *AmountType*<br>***capCode*** — an instance of *CapCode* |
| 2 | If items in batches are not identified by a sequence numbers, continue with Step 8. |
| 3 | If only the Payment Gateway may determine the **BatchSequenceNum**, continue with Step 7.<br>If either the Payment Gateway or the Merchant may determine the **BatchSequenceNum** and ***sequenceNum*** is omitted, continue with Step 7. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Update batch (add item)** (continued)

| Step | Action |
|---|---|
| | Merchant controls the selection of **BatchSequenceNum** |
| 4 | If ***sequenceNum*** is omitted, set ***capCode*** to *batchDataNeeded* and continue with Step 13. |
| 5 | If ***batchData*.availableSeqNum** indicates that ***sequenceNum*** is available, designate ***sequenceNum*** as ***pGwySeqNum*** and continue with Step 8. |
| 6 | If the Payment Gateway cannot override the sequence number specified by the Merchant, set ***capCode*** to ~~batchUnknown~~ *badSeqNum* and continue with Step 13. |
| | Payment gateway controls the selection of **BatchSequenceNum** |
| 7 | From ***batchData*.availableSeqNum**, designate an unused value as ***pGwySeqNum***. |
| 8 | Execute brand and acquirer specific validation of the item indicated by ***perAuth*** (including appropriate components from ***payload***); for example, either the brand or the acquirer may restrict the allowable difference between the authorized amount and the captured amount. If errors occur, set ***capCode*** to an appropriate value and continue with Step 13. |
| 9 | Add the item to the batch identified by ***batchData.*** |
| 10 | Set ***capCode*** to *success*. |
| 11 | Update ***batchData*.availableSeqNum** to indicate that ***pGwySeqNum*** is no longer available. |
| 12 | Invoke "Update **BatchStatus**" on page 483 with the following input.<br>***batchData*** — ***batchData***<br>***transAmt*** — ***transAmt***<br>***transType*** — ***transType***<br>***sameBatch*** — FALSE<br>Note: this will update components in ***batchData***. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

**Update batch (add item)** (continued)

| Step | Action |
|---|---|
| 13 | If ***capCode*** is not *success* and the transaction details are only stored for successful items, continue with Step 15.<br>Otherwise, set ***transAmtType*** according to ***transType***:<br>**If *transType* is one of:** / **then set *transAmtType* to:**<br>*AuthReq*, *CapReq*, *CredRevReq* → *credit*<br>*AuthRevReq*, *CapRevReq*, *CredReq* → *debit* |
| 14 | Construct *TransactionDetail*:<br>*transIDs* → ***trans*.transIDs**<br>*authRRPID* → ***perAuth*.authRRPID**<br>*brandID* → ***trans*.brand**<br>*batchSequenceNum* → ***pGwySeqNum***<br>*reimbursementID* → a value from **Table** 28 on page 399 (optional)<br>*transactionAmt* → ***transAmt***<br>*transactionAmtType* → ***transAmtType***<br>*transactionStatus* → a value from Table 29 on page 399 consistent with ***capCode*** (optional)<br>*transExtensions* → any message extension(s) required to support additional business functions (optional)<br>Append the result to ***batchData*.transactionDetailSeq**. |
| 15 | Decrement the item count for ***rrpid*** in ***batchData*.outstandingRequests** and if the result is zero, remove ***rrpid*** from the list.<br>Store the updated ***batchData*** in the batch database. |
| 16 | Return:<br>***capCode*** → ***capCode***<br>***sequenceNum*** → ***pGwySeqNum***<br>Note: If ***transType*** is not *CapReq*, the value of ***capCode*** must be mapped from **CapCode** values to corresponding values for ***transType***. |

*Continued on next page*

# Payment Gateway Batch Procedures, continued

## Update batch (delete item)

This processing sequence applies to the Payment Gateway. It is invoked when processing a Merchant reversal request (**AuthRevReq**, **CapRevReq**, or **CredRevReq**) to delete an item from a batch that is being accumulated locally.

The Payment Gateway previously invoked "Process batch identification" on page 487 to determine or confirm the **BatchID**. The Merchant may have invoked "Determine batch identification" on page 472 to determine both **BatchID** and **BatchSequenceNum**.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***rrpid*** — an instance of *RRPID*<br>***batchData*** — an instance of *BatchData* (see page 469)<br>***sequenceNum*** — an instance of *BatchSequenceNum* (optional)<br>***transAmt*** — an instance of *CurrencyAmount*<br>***transType*** — the message being processed; one of the following:<br>• *AuthRevReq*<br>• *CapRevReq*<br>• *CredRevReq*<br>***payload*** — an instance of one of the following:<br>• *AuthRevReqData*<br>• *CapRevOrCredReqItem*<br>corresponding to ***transType***. |
| 2 | Locate the item indicated by ***perAuth*** (including appropriate components from ***payload***) and delete it from the batch identified by ***batchData***. |
| 3 | Invoke "Update **BatchStatus**" on page 483 with the following input.<br>***batchData*** — ***batchData***<br>***transAmt*** — ***transAmt***<br>***transType*** — ***transType***<br>***sameBatch*** — TRUE<br>Note: this will update components in ***batchData***. |
| 4 | Delete the entry that corresponds to the item from ***batchData*.transactionDetailSeq**. |
| 5 | Decrement the item count for ***rrpid*** in ***batchData*.outstandingRequests** and if the result is zero, remove ***rrpid*** from the list.<br>Store the updated ***batchData*** in the batch database. |

# Section 2
# Authorization Request/Response Processing

## Overview

### Introduction

The authorization processing consists of two messages, a request from a Merchant to a Payment Gateway and a response from the Payment Gateway back to the Merchant.

These messages are used both for authorization-only transactions and for authorization with capture (sale) transactions.

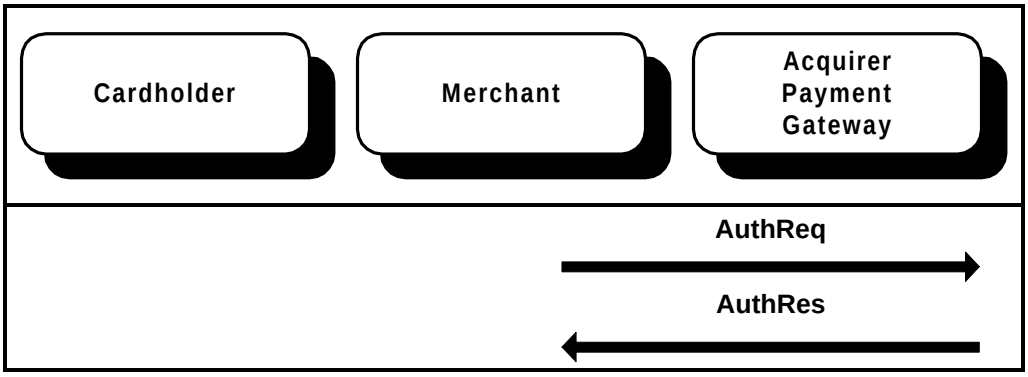


**Figure 6: AuthReq/AuthRes Message Pair**

### Purpose

The Authorization Request and Response message pair provide the mechanism for the Merchant to obtain authorization for a purchase.

In the **AuthReq**, the Merchant sends:

- its own data about the purchase, signed and encrypted, plus
- the **PI** (Payment Instructions) received from the Cardholder.

Since each contains the hash of the **OD** and the amount, the Payment Gateway can verify that the Merchant and Cardholder agree on the order description and the amount to be authorized. Since the **PI** includes the payment card data required for the authorization, the Payment Gateway can authorize the transaction using the existing payment card financial network.

In the **AuthRes**, the Payment Gateway returns the results of the authorization attempt (and the capture attempt, if applicable).

*Continued on next page*

## Overview, continued

**CaptureNow**

~~In some situations, the Acquirer may not be able to perform combined authorization and capture even if~~ **captureNow** ~~is TRUE.~~ If **captureNow** is specified and if the Acquirer does not support capture processing, ~~When this happens,~~ then an **AuthCode** of *captureNotSupported* will be returned. This indicates that the Payment Gateway only performs authorization processing; the Merchant ~~may~~ must submit an **AuthReq** with **captureNow** set to FALSE and subsequently issue an out-of-band ~~a~~ **CapReq** ~~message~~ to capture the payment.

If an operator specified **captureNow**, the operator can resubmit the authorization request without **captureNow**. However, in most cases, a SET transaction occurs without significant operator intervention, and the system is unlikely to be able to recover from this error.

It is therefore vital that the Merchant application be correctly configured as to whether capture processing is performed by a given Payment Gateway.

**Gateway processing of CaptureNow**

When **captureNow** is specified, the payment gateway shall either:

- perform concurrent authorization and capture processing by submitting a single request to the acquirer or payment card financial network (provided that such a request is supported); or
- submit an authorization request and if that request is approved, perform the appropriate capture processing for the transaction.

The merchant may want to force separate processing for batch reconciliation purposes. If the merchant specifies a **BatchID** in the authorization request, the payment gateway may choose to perform the authorization and capture processing separately.

# Merchant Prepares for AuthReq

**AuthInfo**

For the purposes of this documentation, a logical record is defined containing information that must be determined by the Merchant prior to requesting authorization. The actual implementation of collecting and passing this information is at the discretion of the application developer.

| **authInfo** | **{ authReqAmt, [subsequentAuthInd], [captureNow] }** |
|---|---|
| **authReqAmt** | the amount for which authorization is to be requested:<br>• the amount of this shipment (which may be the only shipment); or<br>• the amount of this payment in a series of installment payments; or<br>• the value of goods and services incurred for this recurring payment<br>Normally ***authReqAmt*** is no more than ***trans*.order.purchAmt** less any prior authorizations.<br>If ***trans*.order.installRecurData** exists, ***trans*.order.purchAmt** reflects the anticipated total of all authorizations; each individual authorization is likely to be much smaller, but the total of the authorizations may be greater. |
| **subsequentAuthInd** | a flag indicating Merchant requests an additional authorization; TRUE if this authorization represents a split shipment (except for the final shipment) |
| **captureNow** | a flag indicating whether combined authorization and capture is requested (optional) |

**Table 45: AuthInfo Data**

*Continued on next page*

# Merchant Prepares for AuthReq, continued

**Prepare for authorization**

The Merchant application requires certain information to begin authorization processing. The following processing steps describe one method of obtaining that information.

| Step | Action |
|---|---|
| 1 | Receive as input (from an application-defined interface):<br>***trans*** — the transaction record<br>***authReqAmt*** — an instance of *CurrencyAmount*<br>***subsequentAuthInd*** — an instance of *BOOLEAN* (default FALSE) |
| 2 | Retrieve the latest **perAuth** from ***trans***. If not found, continue with Step 3.<br>Designate it as ***priorAuth***. If ***priorAuth*.authCode** is *callIssuer*, prompt user to determine whether it is appropriate to send this authorization request. |
| 3 | Construct the following contents of *AuthInfo* (see page 499):<br>*authReqAmt* = ***authReqAmt***<br>*subsequentAuthInd* = ***subsequentAuthInd*** |
| 4 | Determine whether to request concurrent authorization and capture, as described in "Capture" on page 24 in Part I, based on:<br>• whether the Payment Gateway identified by ***trans*.peSubject** supports capture processing (see page 498), and<br>• Merchant or Acquirer guidelines.<br>If concurrent authorization and capture is desired, update the following contents of *AuthInfo*:<br>*captureNow* = TRUE |
| 5 | Invoke "Create **AuthReq**" with the following input:<br>***trans*** = ***trans***<br>***authInfo*** = ***authInfo*** |

# Merchant Generates AuthReq

### Create AuthReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***authInfo*** — an instance of *AuthInfo* (see page 499)<br>This procedure uses the following internal variables:<br>***batchID*** — an instance of *BatchID*<br>***batchSequenceNum*** — an instance of *BatchSequenceNum*<br>***authUsesBatch*** — an instance of *BOOLEAN* |
| 2 | If ***trans*.pi** is NULL, stop processing.<br>Note: If the authorization request was initiated by the operator, display a message indicating that the final authorization for this transaction has already been performed. |
| 3 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *subject* matches ***trans*.peSubject**.<br>If found, designate the certificate as ***cert-PE***.<br>Otherwise, stop processing and display a message to the operator indicating that corrective action must be taken to obtain a current copy of the Payment Gateway certificate.<br>Note: Under normal circumstances the certificate is retrieved every 24 hours using **PCertReq** and will be available in the trusted cache. |
| 4 | Construct *AuthRRTags*:<br>*rrpid* — a fresh, statistically unique RRPID<br>*merTermIDs* — from Merchant profile<br>*date* — the current date and time |
| 5 | If ***trans*.pi** is an **AuthToken** (that is, if the tag at the beginning of PI is [2]), retrieve from ***trans*** the **perAuth** record of the most recent successful authorization and designate it as ***priorAuth***. If not found, abort processing. |

*Continued on next page*

# Merchant Generates AuthReq, continued

**Create AuthReq** (continued)

| Step | Action |
|---|---|
| 6 | Construct *AuthTags*:<br>*authRRTags* — the result of Step 4<br>*transIDs* — ***trans*.transIDs**<br>*authRetNum* — ***priorAuth*.authRetNum** (if present) |
| 7 | If ***trans*.pi** is an **AuthToken**, continue with Step 8. Otherwise, construct *CheckDigests*:<br>*hOIData* — ***trans*.hOIData**<br>*hod2* — ***trans*.hod** |
| 8 | Construct *AuthReqPayload*:<br>*subsequentAuthInd* — ***authInfo*.subsequentAuthInd**<br>*authReqAmt* — ***authInfo*.authReqAmt**<br>*avsData* — ***trans*.order.avsData**<br>*specialProcessing* — if required by the brand and product combination identified by ***trans*.brandID**, a value from Table 48 on page 508<br>*cardSuspect* — if the Merchant is suspicious of the Cardholder, a value from Table 49 on page 508<br>*requestCardTypeInd* — TRUE if requesting information about the payment card type; otherwise FALSE<br>*installRecurData* — ***trans*.order.installRecurData**<br>*marketSpecAuthData* — ***trans*.order.marketData**<br>*merchData* — from the Merchant profile ~~(if required by brand policy)~~<br>*aRqExtensions* — any message extension(s) required to support additional business functions (optional) |
| 9 | Construct *AuthReqItem*:<br>*authTags* — the result of Step 6<br>*checkDigests* — the result of Step 7<br>*authReqPayload* — the result of Step 8 |

*Continued on next page*

# Merchant Generates AuthReq, continued

**Create AuthReq** (continued)

| Step | Action |
|---|---|
| 10 | Recommended: Invoke "Create set of Thumbprints for request" on page 118 with the following input:<br>***brand*** → ***trans*.brand**<br>***bin*** → ***trans*.pBIN** |
| 11 | Set ***authUsesBatch*** to FALSE. If ***authInfo*.captureNow** is FALSE, continue with Step 14. |
| 12 | If batch processing is used and the Merchant assigns **BatchID**:<br>• Set ***authUsesBatch*** to TRUE.<br>• Invoke "Determine batch identification" on page 472 with the following input:<br>***brand*** → ***trans*.brand**<br>***pBIN*** → ***trans*.pBIN**<br>***rrpid*** → *authRRTags.rrpid*<br>Designate the value of ***batchID*** returned as ***batchID*** and the value of ***sequenceNum*** returned as ***sequenceNum***. |
| 13 | Construct *SaleDetail*:<br>*batchID* → ***batchID***<br>*batchSequenceNum* → ***sequenceNum***<br>Populate other components of *SaleDetail* based on the type of transaction and according to brand policy. |
| 14 | Construct *AuthReqData*:<br>*authReqItem* → the result of Step 9<br>*mThumbs* → the result of Step 10<br>*captureNow* → ***authInfo*.captureNow**<br>*saleDetail* → the result of Step 13<br>~~*Note: In some situations, the Acquirer may not be able to perform combined authorization and capture even if CaptureNow is TRUE. When this happens, a "captureNotSupported" AuthCode will indicate authorization only; the Merchant may subsequently issue a CapReq message to capture the payment.*~~ |

*Continued on next page*

# Merchant Generates AuthReq, continued

**Create AuthReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>15</td><td>Invoke "Compose EncB" on page 198 with the following input:
<table>
<tr><td>s</td><td colspan="2">the Merchant's signature certificate</td></tr>
<tr><td>r</td><td colspan="2">cert-PE</td></tr>
<tr><td>t</td><td colspan="2">the result of Step 14</td></tr>
<tr><td>b</td><td colspan="2">trans.pi</td></tr>
<tr><td>type-t</td><td colspan="2">id-set-content-AuthReqTBE</td></tr>
<tr><td>type-s</td><td colspan="2">id-set-content-AuthReqTBS</td></tr>
<tr><td>type-b</td><td colspan="2">id-set-content-PI</td></tr>
<tr><td>certs</td><td colspan="2">the new Merchant key encryption certificate for trans.brandID, if received since the last time a message was sent to this Payment Gateway</td></tr>
</table></td></tr>
<tr><td>16</td><td>Store in the message database:
<table>
<tr><td>AuthReqData</td><td colspan="2">the result of Step 14</td></tr>
</table></td></tr>
<tr><td>17</td><td>Construct PerAuth:
<table>
<tr><td>authDate</td><td colspan="2">authRRTags.date</td></tr>
<tr><td>authReqData</td><td colspan="2">the result of Step 14</td></tr>
<tr><td>authRRPID</td><td colspan="2">authRRTags.rrpid</td></tr>
<tr><td>captureNow</td><td colspan="2">authInfo.captureNow</td></tr>
<tr><td>authUsesBatch</td><td colspan="2">authUsesBatch</td></tr>
<tr><td>pi</td><td colspan="2">trans.pi</td></tr>
<tr><td>pResPayload</td><td>completionCode</td><td>orderReceived</td></tr>
</table></td></tr>
<tr><td>18</td><td>Store in the transaction database:
<table>
<tr><td>perAuth</td><td colspan="2">the result of Step 17</td></tr>
<tr><td>pi</td><td colspan="2">NULL</td></tr>
</table>
Note: The perAuth record stored in this step is a new record; it does not replace the record of any prior authorization.</td></tr>
</table>

*Continued on next page*

# Merchant Generates AuthReq, continued

**Create AuthReq** (continued)

| Step | Action |
|---|---|
| 19 | Invoke "Send Message" on page 109 with the following input: |

| | |
|---|---|
| ***recip*** | ~~the Cardholder~~ the Payment Gateway |
| ***msg*** | the result of Step 15 |
| ***ext*** | any message extension(s) required to support additional business functions (optional) |
| ***rrpid*** | *authRRTags.rrpid* |
| ***lid-C*** | ***trans*.transIDs.lid-C** |
| ***lid-M*** | ***trans*.transIDs.lid-M** |
| ***xID*** | ***trans*.transIDs.xID** |

*Continued on next page*

# Merchant Generates AuthReq, continued

### AuthReq data

| | |
|---|---|
| **AuthReq** | **EncB(M, P, AuthReqData, PI )** |
| **AuthReqData** | **{AuthReqItem, [MThumbs], CaptureNow, [SaleDetail]}** |
| **PI** | *See page 372.* |
| **AuthReqItem** | **{AuthTags, [CheckDigests], AuthReqPayload}** |
| **MThumbs** | *Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.* |
| **CaptureNow** | *Boolean indicating that capture should be performed if authorization is approved.* |
| **SaleDetail** | *See page 383.* |
| **AuthTags** | **{AuthRRTags, TransIDs, [AuthRetNum]}** |
| **CheckDigests** | **{HOIData, HOD2}**<br>*Used by Payment Gateway to authenticate* **PI**. *Omit if* **PI** *is an* **AuthToken**. |
| **AuthReqPayload** | *See page 507.* |
| **AuthRRTags** | **RRTags,** *see page 395.*<br>*Note:* **RRPID** *is needed because there may be more than one authorization cycle per* **PReq**. |
| **TransIDs** | *copied from corresponding* **OIData***; see page 436.* |
| **AuthRetNum** | *Identification of the authorization request used within the financial network.* |
| **HOIData** | **DD(OIData)**<br>*See page 436 for the definition of* **OIData**.<br>*An independent hash computed by Merchant. Payment Gateway compares with Cardholder-produced copy in* **PI** *to verify linkage from* **PI** *to* **OIData**. |
| **HOD2** | **DD(HODInput)**<br>*See* "**OIData**" *on page 436 for definition of* **HODInput**.<br>*Independent computation by Merchant. Payment Gateway compares to Cardholder-produced copy in* **PI** *to verify out-of-band receipt by Merchant of relevant data.* |

**Table 46: AuthReq Data**

*Continued on next page*

# Merchant Generates AuthReq, continued

### AuthReqPayload data

| | |
|---|---|
| **AuthReqPayload** | **{SubsequentAuthInd, AuthReqAmt, [AVSData], [SpecialProcessing], [CardSuspect], RequestCardTypeInd, [InstallRecurData], [MarketSpecAuthData], MerchData, [ARqExtensions]}** |
| **SubsequentAuthInd** | *Boolean indicating Merchant requests an additional authorization because of a split shipment.* |
| **AuthReqAmt** | *May differ from* **PurchAmt***; Acquirer policy may place limitations on the permissible difference.* |
| **AVSData** | **{[StreetAddress], Location}**<br>*Cardholder billing address; contents are received from Cardholder using an out-of-band mechanism.*<br>*See page 394 for definition of* **Location**. |
| **SpecialProcessing** | *Enumerated field indicating the type of special processing requested. See page 508.* |
| **CardSuspect** | *Enumerated code indicating that Merchant is suspicious of the Cardholder and the reason for the suspicion. See page 508.* |
| **RequestCardTypeInd** | *Indicates that the type of card should be returned in* **CardType** *in the response; if the information is not available, the value* **unavailable(0)** *is returned.* |
| **InstallRecurData** | *See page 377.* |
| **MarketSpecAuthData** | **< MarketAutoAuth, MarketHotelAuth, MarketTransportAuth >**<br>*Market-specific authorization data.* |
| **MerchData** | **{ [MerchCatCode], [MerchGroup]}** |
| **ARqExtensions** | *The data in an extension to the authorization request ~~must~~ shall be financial and should be related to the processing of an authorization (or subsequent capture) by the Payment Gateway, the financial network, or the Issuer.* |
| **StreetAddress** | *The street address of the cardholder.* |
| **MarketAutoAuth** | **{Duration}** |
| **MarketHotelAuth** | **{Duration, [Prestige]}** |
| **MarketTransportAuth** | **{}**<br>*There is currently no authorization data for this market segment.* |

**Table 47: AuthReqPayload Data**

*Continued on next page*

# Merchant Generates AuthReq, continued

**AuthReqPayload data** (continued)

| | |
|---|---|
| **MerchCatCode** | *Four-byte code (defined in ANSI X9.10) describing merchant's type of business, product, or service.* |
| **MerchGroup** | *Enumerated code identifying the general category of the merchant.* |
| **Duration** | *The anticipated duration of the transaction (in days). This information assists the Issuer by indicating how much time is likely to elapse between the authorization and the capture.* |
| **Prestige** | *Enumerated type of prestigious property; the meaning of the various levels are defined by the payment card brand.* |

**Table 47: AuthReqPayload Data,** continued

**SpecialProcessing**

The following values are defined for **SpecialProcessing**. The processing defined for each value is brand-specific.

| | |
|---|---|
| directMarketing | *The Merchant requests the transaction be processed with direct marketing rules.* |
| preferredCustomer | *The Merchant requests the transaction be processed with preferred customer rules.* |

**Table 48: Enumerated Values for SpecialProcessing**

**CardSuspect**

The following values are defined for **CardSuspect**.

| | |
|---|---|
| unspecifiedReason | *Either the Merchant does not differentiate reasons for suspicion, or the specific reason does not appear in the list.* |

**Table 49: Enumerated Values for CardSuspect**

# Payment Gateway Processes AuthReq

**Process AuthReq**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>**hdr** — an instance of *MessageHeader*<br>**msg** — an instance of *EnvelopedData*<br>**ext** — any message extension(s) required to support additional business functions (optional)<br>This procedure uses the following internal variables:<br>**authCode** — an instance of *AuthCode*<br>**authAmt** — an instance of *CurrencyAmt*<br>**authRetNum** — an instance of *AuthRetNum*<br>**paySysID** — an instance of *PaySysID*<br>**transExists** — an instance of *BOOLEAN*<br>**needCapture** — an instance of *BOOLEAN*<br>**usesBatch** — an instance of *BOOLEAN* |
| 2 | Invoke "Verify *EncB*" on page 199 with the following input:<br>**d** — **msg**<br>**type-t** — *id-set-content-AuthReqTBE*<br>**type-s** — *id-set-content-AuthReqTBS*<br>**type-b** — *id-set-content-PI*<br>Designate:<br>• the value of **t** returned as **req**, and<br>• the value of **b** returned as **pi**. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3</td><td>Validate the following contents of req:
<table>
<tr><td>authReqItem.authTags.<br>authRRTags.rrTags.rrpid</td><td>hdr.rrpid</td></tr>
<tr><td>authReqItem.authTags.<br>transIDs.lid-C</td><td>hdr.messageIDs.lid-C</td></tr>
<tr><td>authReqItem.authTags.<br>transIDs.lid-M</td><td>hdr.messageIDs.lid-M</td></tr>
<tr><td>authReqItem.authTags.<br>transIDs.xID</td><td>hdr.messageIDs.xID</td></tr>
</table>
If errors occur during validation, invoke "Create Error Message" on page 135 with the following input:
<table>
<tr><td>errorCode</td><td>wrapperMsgMismatch</td></tr>
</table></td></tr>
<tr><td>4</td><td>Verify that req.authReqItem.authTags.transIDs.language is a valid RFC 1766 language code. If not, invoke "Create Error Message" on page 135 with the following input:
<table>
<tr><td>errorCode</td><td>unsupportedLanguage</td></tr>
</table></td></tr>
<tr><td>5</td><td>Set usesBatch to FALSE.<br>If req.captureNow is TRUE:<br>• If the Payment Gateway does not support capture processing, set authCode to captureNotSupported and continue with Step 20.<br>• Set needCapture to TRUE.<br>Otherwise, set needCapture to FALSE.</td></tr>
<tr><td>6</td><td>If pi is in the list of:
<table>
<tr><td>used PIs</td><td>• Set authCode to piPreviouslyUsed<br>• Continue with Step 20.</td></tr>
<tr><td>invalid AuthTokens</td><td>• Set authCode to piPreviouslyUsed.<br>• Continue with Step 20.</td></tr>
<tr><td>conditional PIs</td><td>• Delete the PI from that list.<br>• If an authToken with the same authRRPID appears in the list of conditional authTokens, move the AuthToken to the list of invalid authTokens.</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7</td><td>From the trusted cache, retrieve the certificate whose:<br>• keyUsage is digitalSignature,<br>• issuer matches msg.signerInfos[1].issuerAndSerialNumber.issuer, and<br>• serialNumber matches msg.signerInfos[1].issuerAndSerialNumber.serialNumber.<br>Designate the certificate as cert-MS.</td></tr>
<tr><td>8</td><td>Invoke "Process PI" on page 516 with the following input:
<table>
<tr><td>pi</td><td>pi</td></tr>
<tr><td>cert-MS</td><td>cert-MS</td></tr>
<tr><td>checkDigests</td><td>req.authReqItem.checkDigests</td></tr>
<tr><td>installRecurData</td><td>req.authReqItem.authReqPayload.<br>installRecurData</td></tr>
<tr><td>transIDs</td><td>req.authReqItem.authTags.transIDs</td></tr>
</table>
Designate:<br>• the value of authCode returned as authCode,<br>• the value of acqCardCodeMsg returned as acqCardCodeMsg,<br>• the value of trans returned as trans,<br>• the value of transExists returned as transExists, and<br>• the value of authTokenData returned as authTokenData.<br>If authCode is not approved, continue with Step 20.</td></tr>
<tr><td>9</td><td>If req.captureNow is FALSE, continue with Step 11.<br>Otherwise:<br>• Validate components of req.saleDetail (other than batchID and batchSequenceNum) according to brand policy. If errors occur, set authCode and capCode to appropriate values and continue with Step 20.<br>• If batch processing is not used, continue with Step 11.<br>• Set usesBatch to TRUE.</td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

| Step | Action |
|---|---|
| 10 | Invoke "Process batch identification" on page 487 with the following input:<br>***brand*** → ***trans*.brand**<br>***pBIN*** → ***cert-MS*.merchantData.merAcquirerBIN**<br>***rrpid*** → ***req*.authReqItem.authTags.authRRTags.rrTags.rrpid**<br>***mBatchID*** → ***req*.saleDetail.batchID**<br>If the value of ***capCode*** returned is not *success*:<br>• Set ***authCode*** to *captureFailure*.<br>• Designate the value of ***capCode*** returned as ***capCode***.<br>• Continue with Step 20.<br>Otherwise, designate:<br>• the value of ***batchID*** returned as ***batchID***, and<br>• the value of ***batchData*** returned as ***batchData***.<br>Note: In the case of concurrent authorization and capture processing, the batch identification is used for reporting and accounting purposes only. |
| 11 | Process authorization (either through the existing payment card financial networks or locally by the Payment Gateway if allowed by payment brand rules).<br>If ***req*.captureNow** is TRUE, the request should be formatted to request concurrent authorization and capture processing provided that the acquirer and payment card financial network support such processing. The decision about concurrent processing may be affected by whether the merchant specified ***req*.saleDetail.batchID**.<br>Set ***authCode***, ***authRetNum***, and ***paySysID*** and format an instance of *ResponseData* based on the results of the authorization process.<br>If concurrent authorization and capture was attempted:<br>• If ***authCode*** is *success*, set ***capCode*** based on the results of the capture process.<br>• set ***needCapture*** to FALSE.<br>~~Otherwise, if ***req*.captureNow** is TRUE and capture cannot be executed, set ***authCode*** to *captureNotSupported* to indicate successful authorization and continue with Step 20.~~ |
| 12 | If ***authCode*** is *approved*:<br>• Set ***authAmt*** to the amount authorized in Step 11.<br>• Otherwise, set ***authAmt*** to ***req*.authReqItem.authReqPayload.authReqAmt**. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

| Step | Action |
|---|---|
| 13 | If ***authCode*** is not *approved*:<br>• If ***usesBatch*** is TRUE, remove ***req*.authReqItem.authTags.authRRTags.rrTags.rrpid** from ***batchData*.outstandingRequests**.<br>Note: This processing intentionally avoids updating **transactionDetailSeq** in ***batchData***.<br>• Set ***needCapture*** to FALSE.<br>• If ***authCode*** is not *callIssuer*, continue with Step 20. |
| 14 | If ***needCapture*** is TRUE and batches are not accumulated locally:<br>• Process capture via existing payment card financial network.<br>• Set ***capCode*** based on the results of the capture process.<br>• Set ***needCapture*** to FALSE. |
| 15 | Construct the following contents of *PerAuth*: |

| | |
|---|---|
| *authAmt* | ***authAmt*** |
| *authCode* | ***authCode*** |
| *authReqItem* | ***req*.authReqItem** (if it is Payment Gateway policy to store this data) |
| *paySysID* | from the result of Step 11 (if provided) |
| *authRetNum* | ***authRetNum*** |
| *authRRPID* | ***req*.authReqItem.authTags.authRRTags.rrTags.rrpid** |
| *responseData* | from the result of Step 11 (if provided) |
| *batchID* | ***batchID*** (if set) |
| *batchSequenceNum* | ***batchSequenceNum*** (if set) |
| *capCode* | ***capCode*** (if set) |
| *captureNow* | ***req*.captureNow** |
| *responseData* | from the result of Step 11 |
| *saleDetail* | ***req*.saleDetail** |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>16</td><td>If needCapture is TRUE, invoke "Update batch (add item)" on page 493 with the following input:
<table>
<tr><td>trans</td><td>trans</td></tr>
<tr><td>perAuth</td><td>the result of Step 15</td></tr>
<tr><td>rrpid</td><td>req.authReqItem.authTags.authRRTags.rrTags.rrpid</td></tr>
<tr><td>batchData</td><td>batchData</td></tr>
<tr><td>sequenceNum</td><td>req.saleDetail.batchSequenceNum</td></tr>
<tr><td>transAmt</td><td>authAmt</td></tr>
<tr><td>transType</td><td>AuthReq</td></tr>
<tr><td>payload</td><td>req.authReqItem.authReqPayload</td></tr>
</table>
Designate the value of capCode returned as capCode and the value of sequenceNum returned as sequenceNum.</td></tr>
<tr><td>17</td><td>If paySysID is defined:
<ul>
<li>Update the following contents of trans.transIDs:
<table>
<tr><td>paySysID</td><td>from the result of Step 11 (if provided)</td></tr>
</table>
</li>
<li>Store in the result in the transaction database.</li>
</ul></td></tr>
<tr><td>18</td><td>Store in the transaction database:
<table>
<tr><td>perAuth</td><td>the result of Step 15</td></tr>
</table></td></tr>
<tr><td>19</td><td>If authCode is approved or if brand or acquirer policy requires the transaction record to be retained, set transExists to TRUE.</td></tr>
<tr><td>20</td><td>Store in the message database:
<table>
<tr><td>AuthReqData</td><td>req</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process AuthReq** (continued)

| Step | Action |
|---|---|
| 21 | Optional: If ***acqCardCodeMsg*** has not been defined, construct an instance of *AcqCardCodeMsg* to provide additional information about the status of the transaction to the cardholder:<br>*acqCardText* — optional: a textual message to be displayed to Cardholder (using ***req*.authReqItem.authTags.transIDs.language** if available)<br>*acqCardURL* — optional: the URL that references a message to be displayed to Cardholder<br>*acqCardPhone* — optional: a phone number to be presented to Cardholder |
| 22 | Invoke "Create **AuthRes**" on page 526 with the following input:<br>***trans*** — ***trans***<br>***perAuth*** — the result of Step 15<br>***req*** — ***req***<br>***pi*** — ***pi***<br>***cert-MS*** — ***cert-MS***<br>***acqCardCodeMsg*** — from the result of Step 8 or Step 21<br>***batchData*** — ***batchData***<br>***transExists*** — ***transExists*** |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

### Process PI

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***pi*** — an instance of *PI*<br>***cert-MS*** — an instance of *Certificate*<br>***checkDigests*** — an instance of *CheckDigests* (optional)<br>***installRecurData*** — an instance of *InstallRecurData* (optional)<br>***transIDs*** — an instance of *TransIDs* (optional)<br>***reversalFlag*** — an instance of *BOOLEAN* (default FALSE)<br>Note: If ***reversalFlag*** is FALSE, ***installRecurData***, and ***transIDs*** are required; in addition, ***checkDigests*** is required if ***pi*** is not an **AuthToken**.<br>This procedure uses the following internal variables:<br>***authCode*** — an instance of *AuthCode*<br>***transExists*** — an instance of *BOOLEAN* |
| 2 | Set ***authCode*** to *approved*. |
| 3 | Examine the tag at the beginning of ***pi***.<br>• If the tag is [0], continue with Step 4.<br>• If the tag is [1], continue with Step 12.<br>• Otherwise, continue with Step 33. |
| | Processing steps for unsigned ***pi*** |
| 4 | Invoke "Verify *EXH*" on page 180 with the following input:<br>***d*** — ***pi*** (without the leading tag [0])<br>***type-t*** — *id-set-content-PIUnsignedTBE*<br>***type-p*** — *id-set-content-PANToken*<br>Designate:<br>• the value of ***t*** returned as ***pi-oiLink,***<br>• the value of ***p*** returned as ***panToken***, and<br>• ***pi-oiLink.t1*** as ***piHead***. |
| 5 | If ***reversalFlag*** is TRUE, continue with Step 10. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

| Step | Action |
|---|---|
| 6 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *serialNumber* matches ***pi*.piUnsigned.recipientInfos[1].issuerAndSerialNumber**.<br>Designate the certificate as ***cert-PE***. If not found, abort processing. |
| 7 | If ***cert-PE*.cardCertRequired** is TRUE, set ***authCode*** to *signatureRequired* and continue with Step 42. |
| 8 | Invoke "Compare **BrandID**s" on page 119 with the following input:<br>***hier*** — TRUE<br>***brand1*** — ***cert-PE*.subject.organizationName**<br>***brand2*** — ***cert-MS*.subject.organizationName**<br>If the values do not match, set ***authCode*** to *cardMerchBrandMismatch* and continue with Step 42. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

| Step | Action |
|---|---|
| 9 | Validate the following contents of ***panToken***:<br>• *pan* — If required by brand policy, verify using the check digit algorithm described in Appendix N.<br>• *cardExpiry* — If required by acquirer or brand policy, not before today's date<br>If errors occur during validation:<br>• Set ***authCode*** based on the field that failed:<br>• *pan* — *invalidPAN*<br>• *cardExpiry* — *expiredCard*<br>• Continue with Step 42. |
| 10 | Store in the transaction database:<br>• *backKeyData* — ***piHead*.acqBackKeyData**<br>• *brand* — ***pi-oiLink*.oiData.brandID** without Product<br>• *brandID* — ***pi-oiLink*.oiData.brandID**<br>• *cardExpiry* — ***panToken*.cardExpiry**<br>• *pan* — ***panToken*.pan**<br>• *purchAmt* — ***piHead*.inputs.purchAmt**<br>Designate the resulting transaction record as ***trans***. Set ***transExists*** to FALSE. |
| 11 | Continue with Step 28. |
| | Processing steps for signed ***pi*** |
| 12 | Invoke "Verify *EXL*" on page 176 with the following input:<br>• ***d*** — ***pi*** (without the leading tag [1])<br>• ***type-t*** — *id-set-content-PIDualSignedTBE*<br>• ***type-p*** — *id-set-content-PANData*<br>Designate:<br>• the value of ***t*** returned as ***pi-oiLink***,<br>• the result of ***p*** returned as ***panData***, and<br>• ***pi-oiLink.t1*** as ***piHead***. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>13</td><td>If reversalFlag is TRUE, continue with Step 27.</td></tr>
<tr><td>14</td><td>From the trusted cache, retrieve the certificate whose:<br>• keyUsage includes keyEncipherment and<br>• serialNumber matches pi.piDualSigned.recipientInfos[1].issuerAndSerialNumber.<br>Designate the certificate as cert-PE. If not found, abort processing.</td></tr>
<tr><td>15</td><td>From the trusted cache, retrieve the certificate whose:<br>• keyUsage is digitalSignature,<br>• serialNumber matches pi.piDualSigned.signerInfos[1].issuerAndSerialNumber.<br>Designate the certificate as cert-CS. If not found, abort processing.</td></tr>
<tr><td>16</td><td>Construct PIData:<table><tr><td>piHead</td><td>piHead</td></tr><tr><td>panData</td><td>panData</td></tr></table></td></tr>
<tr><td>17</td><td>Invoke “Compose DetachedDigest” on page 143 with the following input:<table><tr><td>t</td><td>the result of Step 16</td></tr><tr><td>type</td><td>id-set-content-PIData</td></tr></table></td></tr>
<tr><td>18</td><td>Construct PI-TBS:<table><tr><td>hPIData</td><td>the result of Step 17</td></tr><tr><td>hOIData</td><td>pi-oiLink.t2</td></tr></table></td></tr>
<tr><td>19</td><td>Invoke “Verify SignedData (SO)” on page 157 with the following input:<table><tr><td>t</td><td>the result of Step 18</td></tr><tr><td>d</td><td>pi.piDualSigned.piSignature</td></tr><tr><td>type</td><td>id-set-content-PI-TBS</td></tr></table></td></tr>
<tr><td>20</td><td>Invoke “Compare BrandIDs” on page 119 with the following input:<table><tr><td>hier</td><td>TRUE</td></tr><tr><td>brand1</td><td>cert-PE.subject.organizationName</td></tr><tr><td>brand2</td><td>cert-CS.subject.organizationName</td></tr></table>If the values do not match, set authCode to cardMerchBrandMismatch and continue with Step 42.</td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

| Step | Action |
|---|---|
| 21 | Invoke "Compare **BrandID**s" on page 119 with the following input:<br>***hier*** = TRUE<br>***brand1*** = ***cert-MS*.subject.organizationName**<br>***brand2*** = ***cert-CS*.subject.organizationName**<br>If the values do not match, set ***authCode*** to *cardMerchBrandMismatch* and continue with Step 42. |
| 22 | Invoke "Compare **BrandID**s" on page 119 with the following input:<br>***hier*** = TRUE<br>***brand1*** = ***cert-PE*.subject.organizationName**<br>***brand2*** = ***cert-MS*.subject.organizationName**<br>If the values do not match, set ***authCode*** to *cardMerchBrandMismatch* and continue with Step 42. |
| 23 | Validate the following contents of ***panData***:<br>*pan*: If required by brand policy, verify using the check digit algorithm described in Appendix N.<br>*cardExpiry*: If required by acquirer or brand policy, not before today's date<br>If errors occur during validation, set ***authCode*** based on the field that failed and continue with Step 42.<br>*pan* = *invalidPAN*<br>*cardExpiry* = *expiredCard* |
| 24 | Construct *HMACPanData*:<br>*pan* = ***panData*.pan**<br>*cardExpiry* = ***panData*.cardExpiry** |
| 25 | Invoke "Keyed-Hash" on page 142 with the following input:<br>***t*** = the result of Step 24<br>***k*** = ***panData*.panSecret**<br>Designate value returned as ***cardholderID***. |
| 26 | Validate ***cardholderID***:<br>*cardholderID* = ***cert-CS*.commonName**<br>If errors occur during validation, ~~send a *signatureFailure* Error message~~ set ***authCode*** to *signatureFailure* and continue with Step 42. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>27</td><td>Store in the transaction database:
<table>
<tr><td>backKeyData</td><td>piHead.acqBackKeyData</td></tr>
<tr><td>brand</td><td>pi-oiLink.oiData.brandID without Product</td></tr>
<tr><td>brandID</td><td>pi-oiLink.oiData.brandID</td></tr>
<tr><td>cardExpiry</td><td>panData.cardExpiry</td></tr>
<tr><td>pan</td><td>panData.pan</td></tr>
<tr><td>purchAmt</td><td>piHead.inputs.purchAmt</td></tr>
<tr><td>transStain</td><td>piHead.transStain (optional)</td></tr>
</table>
Designate the resulting transaction record as trans. Set transExists to FALSE.</td></tr>
<tr><td></td><td>Common processing steps for unsigned and signed pi</td></tr>
<tr><td>28</td><td>If reversalFlag is TRUE, continue with Step 31.</td></tr>
<tr><td>29</td><td>Validate the following contents of checkDigests:
<table>
<tr><td>hOIData</td><td>pi-oiLink.t2</td></tr>
<tr><td>hod2</td><td>piHead.inputs.hod</td></tr>
</table>
If errors occur during validation, set authCode based on the field that failed and continue with Step 42.
<table>
<tr><td>HOIData</td><td>piAuthMismatch</td></tr>
<tr><td>hod2</td><td>~~return a "HODMismatch" Error message~~<br>piAuthMismatch</td></tr>
</table></td></tr>
<tr><td>30</td><td>Validate the following contents of piHead:
<table>
<tr><td>transIDs.xID</td><td>transIDs.xID</td></tr>
<tr><td>transIDs.lid-C</td><td>transIDs.lid-C</td></tr>
<tr><td>transIDs.lid-M</td><td>transIDs.lid-M</td></tr>
<tr><td>merchantID</td><td>cert-MS.merchantData.merID</td></tr>
<tr><td>installRecurData</td><td>installRecurData</td></tr>
<tr><td>transIDs.pReqDate</td><td>transIDs.pReqDate</td></tr>
</table>
If errors occur during validation, set authCode based on the field that failed and continue with Step 42.
<table>
<tr><td>installRecurData</td><td>installRecurMismatch</td></tr>
<tr><td>all other fields</td><td>piAuthMismatch</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

| Step | Action |
|---|---|
| 31 | Store in the transaction database:<br>*installRecurData* — ***piHead*.installRecurData**<br>*merchantID* — ***piHead*.merchantID**<br>*transIDs* — ***piHead*.transIDs** |
| 32 | Continue with Step 40. |
| | Processing steps for ***authToken*** |
| 33 | Invoke "Verify *EncX*" on page 195 with the following input:<br>***d*** — ***pi*** (without the leading tag [2])<br>***type-t*** — *id-set-content-AuthTokenTBE*<br>***type-s*** — *id-set-content-AuthTokenTBS*<br>***type-p*** — *id-set-content-PANToken*<br>Note: As used here ***type-p*** will never appear in any message; it is only used to correctly set BC in the OAEP block.<br>Designate:<br>• the value of ***t*** returned as ***authTokenData***, and<br>• the value of ***p*** returned as ***panToken***. |
| 34 | If ***reversalFlag*** is TRUE, continue with Step 39. |
| 35 | Verify that the entity identified by ***pi*.authToken.signerInfos[1].IssuerAndSerialNumber** is the payment gateway. If not, set ***authCode*** to *piAuthMismatch* and continue with Step 42. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>36</td><td>Validate the following contents of panToken:
<table>
<tr><td>cardExpiry</td><td>if required by acquirer or brand policy, not before today's date</td></tr>
</table>
If errors occur during validation, set authCode to expiredCard and continue with Step 42.</td></tr>
<tr><td>37</td><td>Validate the following contents of authTokenData:
<table>
<tr><td>xID</td><td>transIDs.xID</td></tr>
<tr><td>lid-C</td><td>transIDs.lid-C</td></tr>
<tr><td>lid-M</td><td>transIDs.lid-M</td></tr>
<tr><td>merchantID</td><td>cert-MS.merchantData.merID</td></tr>
</table>
If errors occur during validation, set authCode to piAuthMismatch and continue with Step 42.</td></tr>
<tr><td>38</td><td>If authTokenData.installRecurData.recurring is not present, continue with Step 39.<br>Otherwise, validate the following contents of authTokenData. installRecurData:
<table>
<tr><td>recurringExpiry</td><td>greater than or equal to the current date</td></tr>
<tr><td>recurringFrequency</td><td>less than or equal to the number of days between authTokenData.prevAuthDateTime and the current date</td></tr>
</table>
If errors occur during validation, set authCode based on the field that failed and continue with Step 42.
<table>
<tr><td>recurringExpiry</td><td>recurringExpired</td></tr>
<tr><td>recurringFrequency</td><td>recurringTooSoon</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

| Step | Action |
|---|---|
| 39 | From the transaction database, retrieve the record for ***transIDs*.xID**. If found, designate it as ***trans***.<br><br>Otherwise,<br>• Store in the transaction database:<br><br>*backKeyData* → ***authTokenData*.acqBackKeyData**<br>*brand* → ***cert-MS*.subject.organizationName** without Product<br>*brandID* → ***cert-MS*.subject.organizationName**<br>*cardExpiry* → ***panToken*.cardExpiry**<br>*installRecurData* → ***authTokenData*.installRecurData**<br>*merchantID* → ***authTokenData*.merchantID**<br>*pan* → ***panToken*.pan**<br>*purchAmt* → ***authTokenData*.purchAmt**<br>*transIDs* → ***authTokenData*.transIDs**<br><br>• Designate the resulting transaction record as ***trans***. |
| | Common processing steps |
| 40 | If ***reversalFlag*** is TRUE, continue with Step 43. |
| 41 | Execute additional procedures for installment payments as defined by brand policy. |

*Continued on next page*

# Payment Gateway Processes AuthReq, continued

**Process PI** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>42</td><td>Optional: If an error occurred during this processing, construct AcqCardCodeMsg to provide additional information about the failure to the cardholder:
<table>
<tr><td>acqCardText</td><td>optional: a textual message to be displayed to Cardholder (using transIDs.language if available)</td></tr>
<tr><td>acqCardURL</td><td>optional: the URL that references a message to be displayed to Cardholder</td></tr>
<tr><td>acqCardPhone</td><td>optional: a phone number to be presented to Cardholder</td></tr>
</table></td></tr>
<tr><td>43</td><td>Return:
<table>
<tr><td>authCode</td><td>authCode</td></tr>
<tr><td>acqCardCodeMsg</td><td>the result of Step 42</td></tr>
<tr><td>trans</td><td>trans</td></tr>
<tr><td>transExists</td><td>transExists</td></tr>
<tr><td>authTokenData</td><td>authTokenData (if defined)</td></tr>
</table></td></tr>
</table>

# Payment Gateway Generates AuthRes

### Create AuthRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***req*** — an instance of *AuthReqData*<br>***pi*** — an instance of *PI*<br>***cert-MS*** — an instance of *Certificate*<br>***acqCardCodeMsg*** — an instance of *AcqCardCodeMsg* (optional)<br>***batchData*** — an instance of *BatchData*<br>***transExists*** — an instance of *BOOLEAN*<br>This procedures uses the following internal variables:<br>***capTokenSent*** — an instance of *BOOLEAN* |
| 2 | If ***perAuth*.authAmt** is specified in a currency other than the one used by the cardholder and if currency conversion data is available (for example, because the payment system returned it), construct *CurrConv*:<br>*currConvRate* — either:<br>• the current conversion rate between ***perAuth*.authAmt** currency and Cardholder's requested currency, received from the payment system, or<br>• if the payment system returns the amount in the billing currency, *amountBillingCurrency* / ***perAuth*.authAmt**<br>*cardCurr* — the Cardholder's billing currency, received from the payment system |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3</td><td>Construct AuthHeader:
<table>
<tr><td>authAmt</td><td>perAuth.authAmt</td></tr>
<tr><td>authCode</td><td>perAuth.authCode</td></tr>
<tr><td>responseData</td><td>perAuth.responseData</td></tr>
<tr><td>batchStatus</td><td>optional: if perAuth.capCode is success and perAuth.batchID is defined, batchData.batchStatus</td></tr>
<tr><td>currConv</td><td>the result of Step 2</td></tr>
</table></td></tr>
<tr><td>4</td><td>If perAuth.capCode is specified, construct CapResPayload:
<table>
<tr><td>capCode</td><td>perAuth.capCode</td></tr>
<tr><td>capAmt</td><td>perAuth.authAmt</td></tr>
<tr><td>batchID</td><td>perAuth.batchID (if perAuth.capCode is success)</td></tr>
<tr><td>batchSequenceNum</td><td>perAuth.batchSequenceNum (if perAuth.capCode is success)</td></tr>
<tr><td>cRsPayExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table></td></tr>
<tr><td>5</td><td>Construct AuthResPayload:
<table>
<tr><td>authHeader</td><td>the result of Step 3</td></tr>
<tr><td>capResPayload</td><td>the result of Step 4</td></tr>
<tr><td>aRsExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

| Step | Action |
|---|---|
| 6 | If ***perAuth*.authCode** is *approved*, add to the list of used **PI**s:<br>\| ***pi*** \| ***pi*** \|<br>Include also necessary identifying data, as discussed in "Payment Gateway PI records" on page 466. |
| 7 | If ***perAuth*.authCode** is *callIssuer*, add to the list of conditional **PI**s:<br>\| ***pi*** \| ***pi*** \|<br>Include also necessary identifying data, as discussed in "Payment Gateway PI records" on page 466. |
| 8 | Optional: If ***perAuth*.authCode** is *approved* or *callIssuer* and ***perAuth*.capCode** is not defined, set ***capTokenSent*** to TRUE and invoke "Create **CapToken**" on page 533 with the following input:<br>\| ***trans*** \| ***trans*** \|<br>\| ***perAuth*** \| ***perAuth*** \|<br>Designate:<br>• the value of ***capToken*** returned as ***capToken***, and<br>• the value of ***tokenOpaque*** returned as ***tokenOpaque***. |
| 9 | If ***perAuth*.authCode** is *approved* or *callIssuer* and one of the following conditions exists:<br>• ***req*.authReqItem.authReqPayload.subsequentAuthInd** is TRUE<br>• ***trans*.installRecurData.installTotalTrans** exists and fewer than ***trans*.installRecurData.installTotalTrans** transactions have been processed<br>• ***trans*.installRecurData.recurring** exists and the current date is greater than or equal to **recurringFrequency** days before the earlier of **recurringExpiry** and ***trans*.cardExpiry**<br>then invoke "Create **AuthToken**" on page 535 with the following input:<br>\| ***trans*** \| ***trans*** \|<br>\| ***oldTokenData*** \| ***authTokenData*** \|<br>\| ***authAmt*** \| ***perAuth*.authAmt** \| |
| 10 | If ***perAuth*.authCode** is *callIssuer* and if an **AuthToken** was created in Step 9, add to the list of conditional **authToken**s:<br>\| ***authToken*** \| the result of Step 9 \|<br>Include also necessary identifying data, as discussed in "Payment Gateway PI records" on page 466. |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

| Step | Action |
|---|---|
| 11 | If ***trans*.backKeyData** does not exist or if no message is to be returned to the Cardholder (tunneled through the Merchant), continue with Step 14.<br>If ***acqCardCodeMsg*** is specified, continue with Step 13.<br>Otherwise, construct an instance of *AcqCardMsgData*:<br>*acqCardText* — optional: a textual message to be displayed to Cardholder (using ***trans*.transIDs.language** if available)<br>*acqCardURL* — optional: the URL that references a message to be displayed to Cardholder<br>*acqCardPhone* — optional: a phone number to be presented to Cardholder |
| 12 | Construct *acqCardCodeMsg*:<br>*acqCardCode* — a value from **Table** 9 on page 379<br>*acqCardMsgData* — the result of Step 11 |
| 13 | Invoke "Compose *EncK*" on page 190 with the following input:<br>***k*** — ***trans*.backKeyData.acqBackKey**<br>***s*** — the Payment Gateway's signature certificate<br>***t*** — the result of Step 12<br>***type-t*** — *id-set-content-AcqCardCodeMsgTBE*<br>***type-s*** — *id-set-content-AcqCardCodeMsg*<br>***aid*** — ***trans*.backKeyData.acqBackAlg** |
| 14 | Construct *AuthResBaggage*:<br>*capToken* — ***capToken***<br>*acqCardMsg* — the result of Step 13<br>*authToken* — the result of Step 9 |
| 15 | Copy ***req*.authReqItem.authTags** to an instance of *AuthTags* and update the following components:<br>*transIDs.paySysID* — ***perAuth*.paySysID**<br>*authRetNum* — ***perAuth*.authRetNum** |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

| Step | Action |
|---|---|
| 16 | Retrieve the current Payment Gateway key encryption certificate for the brand identified by ***trans*.brandID** and ***trans*.pBIN**. If not found, abort processing.<br>If ***req*.mThumbs** is absent or if ***req*.mThumbs** is present and does not include the thumbprint of the certificate, designate the certificate as ***cert-PE*** and its Thumbprint as ***peThumb***; otherwise, set ***cert-PE*** and ***peThumb*** to NULL. |
| 17 | Retrieve the **BrandCRLIdentifier** for the brand identified by ***trans*.brand** and designate it as ***bci***; retrieve its Thumbprint and designate it as ***bciThumb***. If not found, abort processing.<br>If ***req*.mThumbs** is present and includes ***bciThumb***, set ***bci*** to NULL. |
| 18 | Construct *AuthResData*:<br>*authTags* — the result of Step 15<br>*brandCRLIdentifer* — ***bci***<br>*peThumb* — ~~GKThumb~~ ***peThumb***<br>*authResPayload* — the result of Step 5 |
| 19 | Invoke "Retrieve Merchant key encryption certificate" on page 537 with the following input:<br>***brandID*** — ***trans*.brandID**<br>***merchantID*** — ***trans*.merchantID** |
| 20 | If ***perAuth*.authCode** is *approved* or *callIssuer* and if ***cert-MS*.merchantData.merAuthFlag** is TRUE and if Acquirer policy allows PANToken to be returned for this transaction, continue with Step 23. |
| 21 | Invoke "Compose *EncB*" on page 198 with the following input:<br>***s*** — the Payment Gateway's signature certificate<br>***r*** — the result of Step 19<br>***t*** — the result of Step 18<br>***b*** — the result of Step 14<br>***type-t*** — *id-set-content-AuthResTBE*<br>***type-s*** — *id-set-content-AuthResTBS*<br>***type-b*** — *id-set-content-AuthResBaggage*<br>***certs*** — ***cert-PE*** |
| 22 | Append the result of Step 21 to the tag [0], then continue with Step 26. |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

| Step | Action |
|---|---|
| 23 | Construct the following contents of *PANToken*:<br>*pan* — ***trans*.pan**<br>*cardExpiry* — ***trans*.cardExpiry** |
| 24 | Invoke "Compose *EncBX*" on page 203 with the following input:<br>***s*** — the Payment Gateway's signature certificate<br>***r*** — the result of Step 19<br>***t*** — the result of Step 18<br>***b*** — the result of Step 14<br>***p*** — the result of Step 23<br>***type-t*** — *id-set-content-AuthResTBEX*<br>***type-s*** — *id-set-content-AuthResTBSX*<br>***type-p*** — *id-set-content-panToken*<br>***type-b*** — *id-set-content-AuthResBaggage*<br>***certs*** — ***cert-PE*** |
| 25 | Append the result of Step 24 to the tag [1]. |
| 26 | Store in the message database:<br>*authResData* — the result of Step 18<br>*authResBaggage* — the result of Step 14 |
| 27 | If ***transExists*** is FALSE, delete the transaction record and continue with Step 30. |
| 28 | Construct the following contents of *PerAuth*:<br>*authResPayload* — the result of Step 5 (if it is Payment Gateway policy to store this data)<br>*tokenOpaque* — ***tokenOpaque*** |
| 29 | Store in the transaction database:<br>*perAuth* — the result of Step 28 |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthRes** (continued)

| Step | Action |
|---|---|
| 30 | Invoke "Send Message" on page 109 with the following input: |

| | |
|---|---|
| ***recip*** | ~~the Cardholder~~ the Merchant |
| ***msg*** | the result of Step 22 or Step 25 |
| ***ext*** | any message extension(s) required to support additional business functions (optional) |
| ***rrpid*** | ***req*.authReqItem.authTags.authRRTags.rrTags.rrpid** |
| ***lid-C*** | ***req*.authReqItem.authTags.transIDs.lid-C** |
| ***lid-M*** | ***req*.authReqItem.authTags.transIDs.lid-M** |
| ***xID*** | ***req*.authReqItem.authTags.transIDs.xID** |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

### Create CapToken

This version of SET supports encryption only to the same Payment Gateway; that is, only the Payment Gateway that created a given **CapToken** will be able to read it.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data |
| 2 | Construct *CapTokenData*:<br>*authRRPID* — ***perAuth*.authRRPID**<br>*authAmt* — ***perAuth*.authAmt**<br>*tokenOpaque* — data from ***trans*** that will be required to process a capture or credit request (included in the event that the transaction record is purged prior to receipt of the request) |
| 3 | If **panToken** is to be included, continue with Step 5.<br>Otherwise, invoke "Compose *Enc*" on page 186 with the following input:<br>***s*** — the Payment Gateway's signature certificate<br>***r*** — the Payment Gateway's key encryption certificate<br>***t*** — the result of Step 2<br>***type-t*** — *id-set-content-CapTokenTBE*<br>***type-s*** — *id-set-content-CapTokenData* |
| 4 | Append the result of Step 3 to the tag [1]. Continue with Step 8. |
| 5 | Construct the following contents of *PANToken*:<br>*pan* — ***trans*.pan**<br>*cardExpiry* — ***trans*.cardExpiry** |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create CapToken** (continued)

| Step | Action |
|---|---|
| 6 | Invoke "Compose *EncX*" on page 194 with the following input:<br>***s*** — the Payment Gateway's signature certificate<br>***r*** — the Payment Gateway's key encryption certificate<br>***t*** — the result of Step 2<br>***p*** — the result of Step 5<br>***type-t*** — *id-set-content-CapTokenTBEX*<br>***type-s*** — *id-set-content-CapTokenTBS*<br>***type-p*** — *id-set-content-PANToken*<br>Note: As used here ***type-p*** will never appear in any message; it is only used to correctly set BC in the OAEP block. |
| 7 | Append the result of Step 6 to the tag [0]. Continue with Step 8. |
| 8 | Return the following:<br>***capToken*** — the result of Step 4 or Step 7<br>***tokenOpaque*** — *capTokenData.tokenOpaque* |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

### Create AuthToken

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***oldTokenData*** — an instance of *AuthTokenData* (optional)<br>***authAmt*** — an instance of *CurrencyAmount*<br>***priorAmt*** — an instance of *CurrencyAmount* (optional) |
| 2 | Construct the following contents of *AuthTokenData*:<br>*transIDs* — ***trans*.transIDs**<br>*purchAmt* — ***trans*.purchAmt**<br>*merchantID* — ***trans*.merchantID**<br>*acqBackKeyData* — ***trans*.backKeyData**<br>*installRecurData* — ***trans*.installRecurData**<br>*authTokenOpaque* — data from ***trans*** that will be required to process a subsequent authorization request (included in the event that the transaction record is purged prior to receipt of the request)<br>Designate the result as ***authTokenData***. |
| 3 | If ***oldTokenData*** is not specified, continue with Step 4. Otherwise:<br>• if ***priorAmt*** is specified, continue with Step 6,<br>• otherwise, continue with Step 5. |
| | Creating first **AuthToken** |
| 4 | Update the following components in ***authTokenData***:<br>*recurringCount* — *1*<br>*prevAuthDateTime* — the current date and time<br>*totalAuthAmount* — ***authAmt***<br>Continue with Step 7. |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Create AuthToken** (continued)

| Step | Action |
|---|---|
| | Creating subsequent **AuthToken** |
| 5 | Update the following components in ***authTokenData***:<br>*recurringCount* — ***oldTokenData*.recurringCount** plus 1<br>*prevAuthDateTime* — the current date and time<br>*totalAuthAmount* — ***oldTokenData*.totalAuthAmount** plus ***authAmt***<br>Continue with Step 7. |
| | Creating **AuthToken** as part of processing a partial reversal<br>Note: A full reversal does not generate a new AuthToken; instead, it restores the previous PI. |
| 6 | Update the following components in ***authTokenData***:<br>*recurringCount* — ***oldTokenData*.recurringCount**<br>*prevAuthDateTime* — ***oldTokenData*.prevAuthDateTime**<br>*totalAuthAmount* — ***oldTokenData*.totalAuthAmt** decreased by ***priorAmt*** then increased by ***authAmt*** |
| | Common processing steps |
| 7 | Construct the following contents of *PANToken*:<br>*pan* — ***trans*.pan**<br>*cardExpiry* — ***trans*.cardExpiry** |
| 8 | Invoke "Compose *EncX*" on page 194 with the following input:<br>***s*** — the Payment Gateway's ~~key-encryption~~ signature certificate<br>***r*** — the Payment Gateway's key encryption certificate<br>***t*** — ***authTokenData***<br>***p*** — the result of Step 7<br>***type-t*** — *id-set-content-AuthTokenTBE*<br>***type-s*** — *id-set-content-AuthTokenTBS*<br>***type-p*** — *id-set-content-PANToken* |
| 9 | Return the following:<br>***authToken*** — the result of Step 8 |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**Retrieve Merchant key encryption certificate**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>**_brandID_** — an instance of *BrandID*<br>**_merchantID_** — an instance of *MerchantID* |
| 2 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment*,<br>• *merchantData.merID* matches **_merchantID_**, and<br>• *subject.organizationName* matches **_brandID_** (as indicated by the result of "Compare BrandIDs" on page 119).<br>If found, designate it as **_cert-ME_** and continue with Step 5. |
| 3 | From the untrusted cache, retrieve the certificate that matches the criteria listed in Step 2.<br>If not found, invoke "Create **Error** Message" on page 135 with the following input:<br>**_errorCode_** — *missingCertificateCRLorBCI* |
| 4 | Designate the certificate retrieved in Step 3 as **_cert-ME_**.<br>Invoke "Verify certificate" on page 129 with the following input:<br>**_cert_** — **_cert-ME_** |
| 5 | Return the following:<br>**_cert-ME_** — **_cert-ME_** |

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

### AuthRes data

| | |
|---|---|
| **AuthRes** | **< EncB(P, M, AuthResData, AuthResBaggage), EncBX(P, M, AuthResData, AuthResBaggage, PANToken) >** |
| **AuthResData** | **{AuthTags, [BrandCRLIdentifier], [PEThumb], AuthResPayload}** |
| **AuthResBaggage** | **{[CapToken], [AcqCardMsg], [AuthToken]}** |
| **PANToken** | *See page 382. Sent if Merchant certificate indicates Merchant is entitled to the information.* |
| **AuthTags** | *Copied from corresponding* **AuthReq***;* **TransIDs** *and* **AuthRetNum** *may be updated with current information.* |
| **BrandCRLIdentifier** | *List of current CRLs for all CAs under a Brand CA. See page 347 in Part II.* |
| **PEThumb** | *Thumbprint of Payment Gateway certificate provided if* **AuthReq.MThumbs** *indicates Merchant needs one.* |
| **AuthResPayload** | *See page 539.* |
| **CapToken** | *See page 380.* |
| **AcqCardMsg** | *If Cardholder included* **AcqBackKeyData** *in* **PIHead***, the Payment Gateway may send this field to the Merchant containing a message (encrypted using the key data) for the Cardholder. The Merchant is required to copy* **AcqCardMsg** *to any subsequent* **PRes** *or* **InqRes** *sent to the Cardholder. See page 379.* |
| **AuthToken** | *Merchant uses as the* **PI** *in a subsequent* **AuthReq***. See page 378.* |

**Table 50: AuthRes Data**

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**AuthResPayload data**

| | |
|---|---|
| **AuthResPayload** | **{AuthHeader, [CapResPayload], [ARsExtensions]}** |
| **AuthHeader** | **{AuthAmt, AuthCode, ResponseData, [BatchStatus], [CurrConv]}** |
| **CapResPayload** | *See page 619.*<br>*Returned if* **CaptureNow** *had a value of TRUE in* **AuthReq**. |
| **ARsExtensions** | *The data in an extension to the authorization response shall ~~must~~ be financial and should be important for the processing of the authorization response or a subsequent authorization reversal or capture request by the Payment Gateway, the financial network, or the Issuer.* |
| **AuthAmt** | *Copied from* **AuthReqPayload.AuthReqAmt**. |
| **AuthCode** | *Enumerated code indicating outcome of payment authorization processing. See page 541.* |
| **ResponseData** | **{[AuthValCodes], [RespReason], [CardType], [AVSResult], [LogRefID]}** |
| **BatchStatus** | *See page 396.* |
| **CurrConv** | **{CurrConvRate, CardCurr}** |
| **AuthValCodes** | **{[ApprovalCode], [AuthCharInd], [ValidationCode], [MarketSpecDataID]}** |
| **RespReason** | *Enumerated code that indicates authorization service entity and (if appropriate) reason for decline. See page 543.* |
| **CardType** | *Enumerated code indicating the type of card used for the transaction. See page 544.* |
| **AVSResult** | *Enumerated Address Verification Service response code. See page 545.* |
| **LogRefID** | *Alphanumeric data assigned to the authorization transaction (used for matching to reversals).* |

**Table 51: AuthResPayload Data**

*Continued on next page*

## Payment Gateway Generates AuthRes, continued

**AuthResPayload data** (continued)

| | |
|---|---|
| **CurrConvRate** | *Currency Conversion Rate: value with which to multiply* **AuthReqAmt** *to provide an amount in the Cardholder's currency.* |
| **CardCurr** | *ISO 4217 currency code of Cardholder.* |
| **ApprovalCode** | *Approval code assigned to the transaction by the Issuer.* |
| **AuthCharInd** | *Enumerated value that indicates the conditions present when the authorization was performed. See page 545.* |
| **ValidationCode** | *Four-byte alphanumeric code calculated to ensure that required fields in the authorization messages are also present in their respective clearing messages.* |
| **MarketSpecDataID** | *Enumerated code that identifies the type of market-specific data supplied on the authorization (as determined by the financial network). See page 545.* |

**Table 51: AuthResPayload Data,** continued

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**AuthCode**

The following values are defined for **AuthCode**.

| | |
|---|---|
| approved | *The authorization request was approved.* |
| unspecifiedFailure | *The authorization request could not be processed for a reason that does not appear elsewhere in this list.* |
| declined | *The authorization request was declined.* |
| noReply | *The Issuer did not respond to the authorization request. This value frequently indicates a temporary system outage in the Issuer's data processing facility. (Payment Gateway generated response)* |
| callIssuer | *The Issuer requests a telephone call from the merchant.* |
| amountError | *The transaction amount could not be processed by a non-SET system (Acquirer, financial network, Issuer, etc.).* |
| expiredCard | *The card has expired.* |
| invalidTransaction | *The request could not be processed by a non-SET system (Acquirer, financial network, Issuer, etc.) because the type of transaction is not allowed.* |
| systemError | *The request could not be processed by a non-SET system (Acquirer, financial network, Issuer, etc.) because data in the request is invalid.* |
| piPreviouslyUsed | *The Payment Instructions in the authorization request have been used for a prior authorization request which was approved and has not been subsequently reversed (Payment Gateway generated response).* |
| recurringTooSoon | *The minimum time between authorizations has not elapsed for a recurring transaction (Payment Gateway generated response).* |
| recurringExpired | *The expiration date for a recurring transaction has passed (Payment Gateway generated response).* |
| piAuthMismatch | *The data in the* **PI** *from the Cardholder does not correspond with the data in the* **OD** *from the Merchant. (Payment Gateway generated response)* |
| installRecurMismatch | **InstallRecurData** *in the* **PI** *from the Cardholder does not correspond with* **InstallRecurData** *in the* **OD** *from the Merchant. (Payment Gateway generated response)* |
| captureNotSupported | *The Payment Gateway does not support capture. (Payment Gateway generated response)* |

**Table 52: Enumerated Values for AuthCode**

*Continued on next page*

# Payment Gateway Generates AuthRes, continued

**AuthCode** (continued)

| | |
|---|---|
| signatureRequired | *The unsigned* **PI** *option is not supported by the Payment Gateway for this brand. (Payment Gateway generated response)* |
| cardMerchBrandMismatch | *The brand in the Cardholder or Merchant signature certificate does not match the brand of the Payment Gateway encryption certificate. (Payment Gateway generated response)* |

**Table 52: Enumerated Values for AuthCode,** continued

**Future values for AuthCode**

The following conditions were identified after the ASN.1 for version 1.0 was completed. They are currently defined as constants mapping to *unspecifiedFailure*. In a future version of the ASN.1, these values will be added to the ENUMERATED **AuthCode**. Application developers are encouraged to use these symbolic names in place of *unspecifiedFailure*.

| | |
|---|---|
| captureFailure | Capture was not attempted as it would have failed as a result of the supplied **batchID** and/or **batchSequenceNum**. |
| invalidPANInfo | Supplied PAN does not conform to check digit algorithm or card has expired. |
| signatureFailure | The **cardholderID** recreated by the Payment Gateway from **panData** does not match that in the Cardholder certificate. |

**Table 53: Future Enumerated Values for AuthCode**

*Continued on next page*

## Payment Gateway Generates AuthRes, continued

**RespReason**

The following values are defined for **RespReason**.

| | |
|---|---|
| issuer | *The authorization was processed by the Issuer.* |
| standInTimeOut | *The authorization was processed by the Stand-In Processing System after it timed out waiting for the Issuer.* |
| standInFloorLimit | *The authorization was processed by the Stand-In Processing System because the transaction amount is below the Issuer limit.* |
| standInSuppressInquiries | *The authorization was processed by the Stand-In Processing System because the Issuer had suppressed incoming authorization traffic.* |
| standInIssuerUnavailable | *The authorization was processed by the Stand-In Processing System because no connection to the Issuer was available.* |
| standInIssuerRequest | *The authorization was processed by the Stand-In Processing System at the Issuer's request.* |

**Table 54: Enumerated Values for RespReason**

*Continued on next page*

## Payment Gateway Generates AuthRes, continued

**CardType**

The following values are defined for **CardType**.

| | |
|---|---|
| unavailable | *Unknown card type* |
| classic | *Brand-specific classic card product* |
| gold | *Brand-specific gold card product* |
| platinum | *Brand-specific platinum card product* |
| premier | *Brand-specific premier card product* |
| debit | *Brand-specific debit card product* |
| pinBasedDebit | *Brand-specific PIN-based debit card product* |
| atm | *Brand-specific ATM card product* |
| electronicOnly | *Brand-specific electronic-only card product* |
| unspecifiedConsumer | *Brand-specific unspecified consumer card product* |
| corporateTravel | *Brand-specific corporate travel card product* |
| purchasing | *Brand-specific purchasing card product* |
| business | *Brand-specific business card product* |
| unspecifiedCommercial | *Brand-specific unspecified commercial card product* |
| privateLabel | *Brand-specific private label card product* |
| proprietary | *Brand-specific proprietary card product* |

**Table 55: Enumerated Values for CardType**

*Continued on next page*

## Payment Gateway Generates AuthRes, continued

**AVSResult**

The following values are defined for **AVSResult**.

| | |
|---|---|
| resultUnavailable | *AVS service was unavailable.* |
| noMatch | *Neither address nor postal code matches.* |
| addressMatchOnly | *Address matches, postal code does not.* |
| postalCodeMatchOnly | *Postal code matches, address does not.* |
| fullMatch | *Both address and postal code match.* |

**Table 56: Enumerated Values for AVSResult**

**AuthCharInd**

The following values are defined for **AuthCharInd**.

| | |
|---|---|
| directMarketing | *Meet Direct Marketing requirements for card not present* |
| recurringPayment | *Meet Direct Marketing recurring payment qualification without address verification request* |
| addressVerification | *Meet requirements for address verification; verification requested for card not present: Direct Marketing, Transport market segments* |
| preferredCustomer | *Meet requirements for Preferred Customer, card not present: Hotel/Auto Rental and Transport market segments* |
| incrementalAuth | *Incremental authorization qualified for Custom Payment Service, card may or may not be present: Hotel/Auto Rental market segments* |

**Table 57: Enumerated Values for AuthCharInd**

**MarketSpecDataID**

The following values are defined for **MarketSpecDataID**.

| | |
|---|---|
| failedEdit | *Invalid value specified; market-specific processing will not be performed.* |
| auto | *Auto Rental market* |
| hotel | *Hotel/Motel Lodging market* |
| transport | *Passenger Transport market* |

**Table 58: Enumerated Values for MarketSpecDataID**

# Merchant Processes AuthRes

### Process AuthRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *EnvelopedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>This procedure uses the following internal variables:<br>***completionCode*** — an instance of *CompletionCode* |
| 2 | Examine the tag at the beginning of ***msg***.<br>• If the tag is [0], continue with Step 3.<br>• Otherwise, continue with Step 4. |
| 3 | Invoke "Verify *EncB*" on page 199 with the following input:<br>***d*** — ***msg*** (without the leading tag [0])<br>***type-t*** — *id-set-content-AuthResTBE*<br>***type-s*** — *id-set-content-AuthResTBS*<br>***type-b*** — *id-set-content-AuthResBaggage*<br>Designate:<br>• the value of ***t*** returned as ***res***, and<br>• the value of ***b*** returned as ***baggage***.<br>Continue with Step 5. |
| 4 | Invoke "Verify *EncBX*" on page 205 with the following input.<br>***d*** — ***msg*** (without the leading tag [1])<br>***type-t*** — *id-set-content-AuthResTBEX*<br>***type-s*** — *id-set-content-AuthResTBSX*<br>***type-p*** — *id-set-content-panToken*<br>***type-b*** — *id-set-content-AuthResBaggage*<br>Designate:<br>• the value of ***t*** returned as ***res***,<br>• the value of ***b*** returned as ***baggage***, and<br>• the value of ***p*** returned as ***panToken***. |

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

| Step | Action |
|---|---|
| 5 | Validate the following contents of ***res***:<br>*authTags.rrpid* = ***hdr*.rrpid**<br>*authTags.transIDs.xID* = ***hdr*.messageIDs.xID**<br>*authTags.transIDs.lid-C* = ***hdr*.messageIDs.lid-C**<br>*authTags.transIDs.lid-M* = ***hdr*.messageIDs.lid-M**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 6 | Retrieve the transaction record that is identified by ***res*.authTags.transIDs.xid** and designate it as ***trans***.<br>If not found, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownXID* |

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

| Step | Action |
|---|---|
| 7 | Retrieve from ***trans*** the **perAuth** record of the outstanding authorization request and designate it as ***perAuth***. If not found, abort processing. |
| 8 | Validate the following contents of ***res*.authTags**:<br>*lid-C* — ***trans*.lid-C**<br>*lid-M* — ***trans*.lid-M**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *unknownLID* |
| 9 | If ~~GKThumb~~ ***res*.peThumb** is present, verify that it matches the thumbprint of an existing Payment Gateway key encryption certificate in the trusted cache. If not:<br>• From the untrusted cache, retrieve the key encryption certificate whose Thumbprint matches ***res*.peThumb** and designate it as ***cert-PE***. If not found, abort processing.<br>• Invoke "Verify certificate" on page 129 with the following input:<br>***cert*** — ***cert-PE*** |
| 10 | Designate ***res*.authResPayload.authHeader.authCode** as ***authCode***. |
| 11 | Process the following contents of *ResponseData* according to brand or Merchant policy:<br>• *cardType*<br>• *avsResult* |

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>12</td><td>If perAuth.authUsesBatch is FALSE, continue with Step 15.</td></tr>
<tr><td>13</td><td>Invoke "Process batch information" on page 476 with the following input:
<table>
<tr><td>propBatchID</td><td>perAuth.authReqData.saleDetail.batchID</td></tr>
<tr><td>propSeqNum</td><td>perAuth.authReqData.saleDetail.batchSequenceNum</td></tr>
<tr><td>batchID</td><td>res.authResPayload.capResPayload.batchID</td></tr>
<tr><td>seqNum</td><td>res.authResPayload.capResPayload.batchSequenceNum</td></tr>
<tr><td>brandID</td><td>trans.brandID</td></tr>
<tr><td>pBIN</td><td>trans.pBIN</td></tr>
<tr><td>rrpid</td><td>perAuth.authRRPID</td></tr>
<tr><td>batchStatusSeq</td><td>res.authHeader.batchStatus</td></tr>
<tr><td>transAmt</td><td>res.authResPayload.authHeader.authAmt</td></tr>
<tr><td>transType</td><td>AuthReq</td></tr>
</table>
Designate the value of batchData returned as batchData.</td></tr>
<tr><td>14</td><td>Optional:
<ul>
<li>Construct the following contents of TransactionDetail:
<table>
<tr><td>transIDs</td><td>trans.transIDs</td></tr>
<tr><td>authRRPID</td><td>perAuth.authRRPID</td></tr>
<tr><td>brandID</td><td>trans.brand</td></tr>
<tr><td>batchSequenceNum</td><td>a SEQUENCE with one item res.authResPayload.capResPayload.batchSequenceNum</td></tr>
<tr><td>transactionAmt</td><td>res.authResPayload.capResPayload.capAmt</td></tr>
<tr><td>transactionAmtType</td><td>credit</td></tr>
<tr><td>transExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table>
</li>
<li>Append the result to batchData.transactionDetailSeq. Store the updated batchData in the batch database.</li>
</ul></td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>15</td><td>Based on authCode, set completionCode:
<table>
<tr><th>authCode</th><th>completionCode</th></tr>
<tr><td>approved</td><td>• if perAuth.captureNow is TRUE, capturePerformed<br>• otherwise authorizationPerformed</td></tr>
<tr><td>callIssuer<br>noReply<br>recurringTooSoon</td><td>orderReceived</td></tr>
<tr><td>any other value</td><td>orderRejected</td></tr>
</table></td></tr>
<tr><td>16</td><td>If completionCode is authorizationPerformed or capturePerformed, construct AuthStatus:
<table>
<tr><td>authDate</td><td>perAuth.authDate</td></tr>
<tr><td>authCode</td><td>authCode</td></tr>
<tr><td><s>authAmt</s><br>authRatio</td><td>res.authResPayload.authHeader.authAmt ÷ trans.order.purchAmt</td></tr>
<tr><td>currConv</td><td>res.authResPayload.authHeader.currConv</td></tr>
</table></td></tr>
<tr><td>17</td><td>If completionCode is capturePerformed, construct CapStatus:
<table>
<tr><td>capDate</td><td>perAuth.authDate</td></tr>
<tr><td>capCode</td><td>res.authResPayload.capResPayload.capCode</td></tr>
<tr><td><s>capAmt</s><br>capRatio</td><td>res.authResPayload.capResPayload.capAmt ÷ trans.order.purchAmt</td></tr>
</table></td></tr>
<tr><td>18</td><td>Construct Results:
<table>
<tr><td>acqCardMsg</td><td>baggage.acqCardMsg</td></tr>
<tr><td>authStatus</td><td>the result of Step 16</td></tr>
<tr><td>capStatus</td><td>the result of Step 17</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>19</td><td>Construct PResPayload:
<table>
<tr><td>completionCode</td><td>the result of Step 15</td></tr>
<tr><td>results</td><td>the result of Step 18</td></tr>
<tr><td>pRsExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table></td></tr>
<tr><td>20</td><td>Copy trans.transIDs to an instance of TransIDs and update the following contents:
<table>
<tr><td>paySysID</td><td>res.authTags.transIDs.paySysID</td></tr>
</table></td></tr>
<tr><td>21</td><td>Copy perAuth.capPayload to an instance of CapPayload and update the following contents:
<table>
<tr><td>saleDetail.batchID</td><td>res.authResPayload.capResPayload.batchID</td></tr>
<tr><td>saleDetail.batchSequenceNum</td><td>res.authResPayload.capResPayload.batchSequenceNum</td></tr>
</table>
~~Process SaleDetail according to brand policy and processing steps.~~</td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

| Step | Action |
|---|---|
| 22 | If ***panToken*** is present, store ***panToken*.pan** and ***panToken*.cardExpiry** in secure data storage and designate its location as ***panRef***. |
| 23 | Construct the following contents of *PerAuth*: |

| | |
|---|---|
| *authAmt* | ***res*.authResPayload.authHeader.authAmt** |
| *authCode* | ***res*.authResPayload.authHeader.authCode** |
| *approvalCode* | ***res*.authResPayload.authHeader.responseData.authValCodes.approvalCode** |
| *authResPayload* | ***res*.authResPayload** |
| *authRetNum* | ***res*.authTags.authRetNum** |
| *batchID* | ***batchID*** |
| *batchSequenceNum* | ***batchSequenceNum*** |
| *captureNow* | ***req*.captureNow** |
| *capAmt* | ***res*.authResPayload.capResPayload.capAmt** |
| *capCode* | ***res*.authResPayload.capResPayload.capCode** |
| *capPayload* | the result of Step 21 |
| *capResPayload* | ***res*.authResPayload.capResPayload** |
| *acqCardMsg* | ***baggage*.acqCardMsg** |
| *capToken* | ***baggage*.capToken** |
| *pResPayload* | the result of Step 19 |

| | |
|---|---|
| 24 | Store in the transaction database: |

| | |
|---|---|
| *perAuth* | the result of Step 20 |
| *pi* | ***baggage*.authToken** |
| *panRef* | ***panRef*** |

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>25</td><td>
<table>
<tr><th>If authCode is:</th><th>then:</th></tr>
<tr><td>approved<br>declined<br>expiredCard<br>invalidPANInfo<br>signatureFailure</td><td>Continue with Step 27.</td></tr>
<tr><td>callIssuer</td><td>Advise the operator that manual processing is required. See “Referral Processing” on page 555 for additional information.</td></tr>
<tr><td>noReply</td><td>Continue with Step 26.</td></tr>
<tr><td>any other value</td><td>Advise the operator and stop processing.</td></tr>
</table>
</td></tr>
<tr><td>26</td><td>Perform one of the following actions:<ul><li>inform the operator that manual processing is required to resubmit the authorization;</li><li>suspend processing for a period of time and invoke “Prepare for authorization” on page 500 with appropriate values.</li></ul>Note: The mechanism to suspend and resume processing is at the discretion of the application developer.<br>Note: The amount of time to suspend processing for a noReply response is determined by merchant or acquirer policy; a reasonable value is in the range of ten minutes to one hour.</td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRes, continued

**Process AuthRes** (continued)

| Step | Action |
|---|---|
| 27 | Delete from the message database the *AuthReqData* whose key is ***res*.authTags.authRRTags.rrpid**. |
| 28 | If ***trans*.pResPending** is TRUE, invoke "Create **PRes**" on page 447 with the following input: |

| | |
|---|---|
| ***trans*** | ***trans*** |
| ***rrpid*** | ***trans*.PReqRRPID** |
| ***chall-C*** | ***trans*.chall-C** |
| ***pRes*** | TRUE |

**Capture processing**

If the authorization was approved and the capture was not performed as part of the authorization process (**CaptureNow** was FALSE), it will be necessary to do so after the order has been filled. At that time, invoke "Prepare for capture" on page 593 with the following input:

| | |
|---|---|
| ***trans*** | the transaction record |

# Referral Processing

### Overview

When an Issuer processes an authorization request, the response can indicate three possible results: approved, declined, or conditionally declined. This latter result is commonly called a *referral* and is indicated by an **AuthCode** value of *callIssuer* (4).

Upon receiving a referral response:

- An operator may call the Acquirer using a telephone number supplied out-of-band.
- After identifying the transaction, the Acquirer connects the operator to the Issuer.
- The Issuer may convert the authorization to an approval and provide the operator with **ApprovalCode** over the phone.

### Creating AuthRes for referral

When the Payment Gateway receives notification of a referred authorization, it:

- shall not attempt to perform capture processing, even if **AuthReq.CaptureNow** was set to TRUE, and
- shall create an **AuthRes** with **AuthCode** set to *callIssuer*.

The **AuthRes** shall in all other ways be identical to the **AuthRes** that would be returned on an approved authorization (without capture); that is, it includes **CapToken** and **AuthToken**, if they would otherwise have been included. (If the referral does not receive voice authorization, the Payment Gateway will not honor the **CapToken** or **AuthToken**.)

### Operator contacts referral center

~~Merchant software shall not attempt to authorize a referred authorization by issuing an additional **AuthReq** message.~~ An operator may contact the referral center designated by the Acquirer and attempt to get an **ApprovalCode** from the Issuer.

*Continued on next page*

# Referral Processing, continued

**Issuer converts to approved**

If the Issuer converts the authorization to *approved*:

- Merchant software shall allow the operator to enter an approval code. The software will then process the transaction as though the **AuthCode** had been *approved*, as follows:

  Invoke "Process Referral (Merchant)" on page 557 with the following input:

  | | |
  |---|---|
  | ***trans*** | the transaction record |
  | ***perAuth*** | authorization-specific transaction data |
  | ***approved*** | TRUE |
  | ***approvalCode*** | the **ApprovalCode** entered by the operator |

  Note: The original value of **AuthCode** remains available in the *PerAuth* record.

- To capture the transaction, the Merchant software may issue a **CapReq** with **authResPayload.approvalCode** set to the new approval number (or may request capture via an out-of-band message).

**Issuer converts to declined**

If the Issuer converts the authorization to *declined*:

Merchant software shall allow the operator to indicate that the authorization is declined. The software will then process the transaction as though the **AuthCode** had been *declined*, as follows:

Invoke "Process Referral (Merchant)" on page 557 with the following input:

| | |
|---|---|
| ***trans*** | the transaction record |
| ***perAuth*** | authorization-specific transaction data |
| ***approved*** | FALSE |

**Payment Gateway processes referral**

The Payment Gateway shall process referred authorizations in the same manner as *approved* transactions (such as generating an **AuthToken** or **CapToken**) with one exception:

> All subsequent messages, including capture requests, shall be processed as if the transaction had been approved ***if and only if*** the Merchant provides a valid **ApprovalCode**.

*Continued on next page*

# Referral Processing, continued

**Process referral (Merchant)**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***approved*** — an instance of *BOOLEAN*<br>***approvalCode*** — an instance of *ApprovalCode* (optional) |
| 2 | If ***approved*** is FALSE, continue with Step 9. |
| | Issuer converted referral to an approval |
| 3 | Construct *AuthStatus*:<br>*authDate* — ***perAuth*.authDate**<br>*authCode* — *approved*<br>*authRatio* — ***perAuth*.authAmt ÷ *trans*.order.purchAmt**<br>*currConv* — ***perAuth*.authResPayload.currConv** |
| 4 | Construct *Results*:<br>*acqCardMsg* — ***perAuth*.acqCardMsg**<br>*authStatus* — the result of Step 3 |
| 5 | Construct *PResPayload*:<br>*completionCode* — *authorizationPerformed*<br>*results* — the result of Step 4<br>*pRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 6 | Update the following contents of ***perAuth***:<br>*approvalCode* — ***approvalCode***<br>*pResPayload* — the result of Step 5 |
| 7 | Store in the transaction database:<br>*perAuth* — ***perAuth*** |

*Continued on next page*

# Referral Processing, continued

**Process referral (Merchant)** (continued)

| Step | Action |
|---|---|
| 8 | If ***perAuth*.captureNow** is TRUE, invoke "Create **CapReq**" on page 597 with the following input:<br>***perAuthArray*** — a SEQUENCE consisting of a single item: ***perAuth***<br>Stop processing. |
| | Issuer converted referral to a decline |
| 9 | Invoke "Create **AuthRevReq**" on page 564 with the following input:<br>***trans*** — ***trans***<br>***perAuth*** — ***perAuth***<br>***newAmt*** — zero (0)<br>Note: The contents of the transaction record and the *PerAuth* entry will be changed so ***trans*** and ***perAuth*** must be refreshed. |
| 10 | Construct *PResPayload*:<br>*completionCode* — *orderRejected*<br>*pRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 11 | Update the following contents of ***perAuth***:<br>*pResPayload* — the result of Step 10 |
| 12 | Delete the following contents of ***perAuth***:<br>• *authToken*<br>• *capToken* |
| 13 | Store in the transaction database:<br>*perAuth* — ***perAuth*** |

# Section 3
# Authorization Reversal Request/Response Processing

## Overview

### Introduction

The authorization reversal message pair is used to reduce or cancel a previously approved authorization, or to split a previously unsplit authorization.

Note: **AuthRevReq/Res** cannot be used to unsplit a previously split transaction.

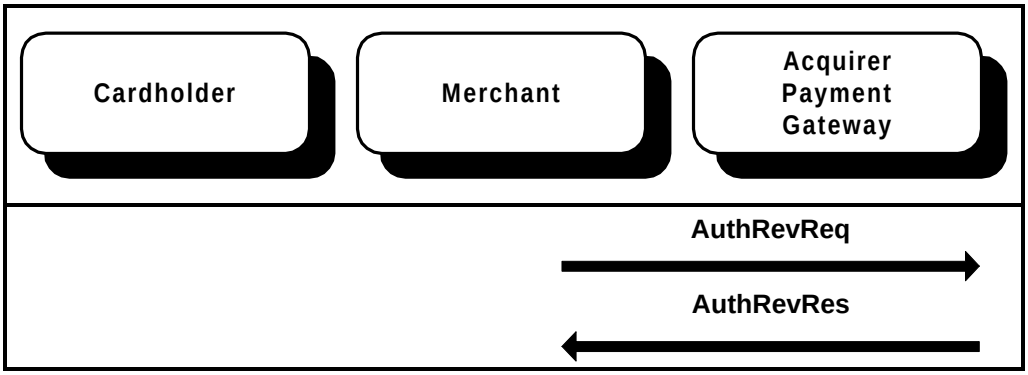


**Figure 7: AuthRevReq/AuthRevRes Message Pair**

*Continued on next page*

# Overview, continued

**Purpose**

**AuthRevReq** carries information from the Merchant necessary for the Payment Gateway to produce an authorization reversal request message that can be processed by the Acquirer or financial network for transmission to the Issuer.

This message pair is optional and is used only if change or elimination of an authorization is required:

- **AuthRevReq** may be sent to the Payment Gateway at any time after authorization but before capture in order:
  - to ~~change~~ decrease the amount of the authorization,
  - to completely remove the authorization whenever an order with an outstanding authorization will not be completed, or
  - to request a subsequent authorization when one was not previously requested (for details, see page 561).
- If the authorization request included **CaptureNow**, **AuthRevReq** may be sent at any time after the authorization to completely reverse both the capture and the authorization.
- If the original **AuthCode** was *callIssuer*, and if the Issuer subsequently declined the authorization, the Merchant software should send an **AuthRevReq** so that the Payment Gateway knows that the **CapToken** and, if applicable, the **AuthToken**, are no longer valid. This is the only situation in which the original authorization was not *approved* that an **AuthRevReq** should be sent.

**Caveats**

An authorization reversal message is simply an advice to the Issuer. Whether the Issuer actually does any processing is a business decision. For example:

- Some Issuers adjust the cardholder's open-to-buy only for the most recent authorization performed and ignore all other requests.
- Some payment systems do not support partial reversals. In this case, the Payment Gateway should format a response message to the merchant without sending any message to the Issuer via the payment system. The cardholder's open-to-buy is lower than it needs to be, but the merchant has done everything possible to correct the situation.

*Continued on next page*

# Overview, continued

**Split shipments**

If, after an initial **AuthReq** for which a subsequent authorization was not requested, an operator discovers that a shipment must be split, **AuthRevReq** shall be used to request a subsequent authorization, as follows:

- The operator shall submit an **AuthRevReq** which reduces the **AuthAmt** to reflect the value of the initial shipment and with **SubsequentAuthInd** set to TRUE.
- The Payment Gateway shall return an **AuthToken** in the **AuthRevRes**.
- The Merchant shall use this **AuthToken** in the authorization request for the next partial shipment.

**Reverse most recent authorization only**

There may be more than one uncaptured authorization for a given **XID** at a given time. For example, in the case of a split shipment, two authorizations might be outstanding if the merchant split the shipment and then was suddenly able to fill both parts of the order.

If the merchant should then send an **AuthRevReq** message for the first authorization, the **TotalAuthAmt** (and other components) of the **AuthToken** for the second authorization would be incorrect.

In order to prevent this and similar data problems, authorizations must be reversed in the opposite of the order in which they were applied. This is true even if some of the authorizations have been captured; in this case, the captures and any related credits must be reversed as well. Note that the Merchant must keep whatever data is necessary to ensure that it can create the messages to reapply all the other messages that were reversed.

**Processing requirements**

The Merchant Software shall:

- include either a **CapToken** or an exact copy of **AuthReqData** and **AuthResPayload** from the original **AuthReq/AuthRes** pair (with those field modifications indicated in the following processing steps);
- if the original **AuthReq** included **CaptureNow**:
  - use **AuthRevReq** with **CaptureNow** to reverse the transaction (no other means of reversing such a transaction are permitted);
  - request only a full reversal.

# Merchant Prepares for AuthRevReq

**Prepare for authorization reversal**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***perAuth*** — authorization-specific transaction data<br>***newAmt*** — an instance of *CurrencyAmount*<br>Note: ***newAmt*** must always be less than the amount of the most recently authorized amount (whether that amount was established by **AuthReq** or **AuthRevReq**). Once the merchant releases the hold on the funds by doing a partial reversal, it can only be recovered by doing another **AuthReq**. |
| 2 | If any of the following conditions is TRUE, stop processing:<br>• ***perAuth*.authCode** is not *approved* or *callIssuer*<br>• ***perAuth*.authCode** is *callIssuer* and ***perAuth*.approvalCode** has not been defined (See "Referral Processing" on page 555.)<br>• ***perAuth*.capCode** is defined and ***perAuth*.captureNow** is FALSE (Note: The transaction was captured using **CapReq** rather than **AuthReq** with **CaptureNow**; use **CapRevReq** to reverse it.)<br>• ***perAuth*.capCode** is defined and ***newAmt*** is not zero<br>• ***perAuth*.completionCode** is *creditPerformed* |
| 3 | Retrieve the transaction record that corresponds to ***perAuth*** and designate it as ***trans***. If not found, abort processing. |
| 4 | If ***perAuth*** is not the most recently approved authorization that has not been completely reversed, present the operator with a list of all authorizations that followed the processing of ***perAuth*** along with the corresponding captures and credits. Inform the operator that all of these transactions must be reversed in order to continue.<br>Note: There is no guarantee that these transactions can be successfully resubmitted after being reversed so a manual confirmation from the operator should be required to effect the reversals. |

*Continued on next page*

# Merchant Prepares for AuthRevReq, continued

**Prepare for authorization reversal** (continued)

| Step | Action |
|---|---|
| 5 | Invoke "Create **AuthRevReq**" on page 564 with the following input: |
| 6 | Reapply all authorizations, captures, and credits that were reversed in Step 4 in order to get to the target authorization. Apply other authorizations in their original sequence. Captures must be applied after authorizations and credits must be applied after captures. |

| | |
|---|---|
| ***trans*** | ***trans*** |
| ***perAuth*** | ***perAuth*** |
| ***newAmt*** | ***newAmt*** |
| ***requestSub*** | a flag indicating Merchant requests an additional authorization; TRUE if this authorization represents a split shipment (except for the final shipment) |

# Merchant Generates AuthRevReq

### Create AuthRevReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***newAmt*** — an instance of *CurrencyAmount*<br>***requestSubs*** — an instance of *BOOLEAN*<br>This procedure uses the following internal variables:<br>***batchID*** — an instance of *BatchID*<br>***batchSequenceNum*** — an instance of *BatchSequenceNum* |
| 2 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *subject* matches ***trans*.peSubject**.<br>If found, designate the certificate as ***cert-PE***.<br>Otherwise, stop processing and display a message to the operator indicating that corrective action must be taken to obtain a current copy of the Payment Gateway certificate.<br>Note: Under normal circumstances the certificate is retrieved every 24 hours using **PCertReq** and will be available in the trusted cache. |
| 3 | Construct *AuthRevRRTags*:<br>*rrpid* — a fresh, statistically unique **RRPID**<br>*merTermIDs* — from Merchant profile<br>*date* — current date and time |
| 4 | Construct *AuthRevTags*:<br>*authRevRRTags* — the result of Step 3<br>*authRetNum* — ***perAuth*.authRetNum** |
| 5 | Recommended: Invoke "Create set of Thumbprints for request" on page 118 with the following input:<br>***brand*** — ***trans*.brand**<br>***bin*** — ***trans*.pBIN** |

*Continued on next page*

# Merchant Generates AuthRevReq, continued

**Create AuthRevReq** (continued)

| Step | Action |
|---|---|
| 6 | If ***perAuth*.capToken** exists:<br>• if ***requestSubs*** is TRUE and ***perAuth*.authReqData.authReqItem.authReqPayload.subsequentAuthInd** is FALSE, continue with Step 7;<br>• if ***perAuth*.captureNow** is TRUE and ***perAuth*.batchID** is assigned and the merchant assigns **BatchID**, continue with Step 7;<br>• if ***perAuth*.authCode** is *callIssuer*, continue with Step 11;<br>• otherwise, continue with Step 12.<br>Note: If the authorization included a **CapToken**, the Merchant does not ordinarily need to include data from the authorization request. The exceptions to this rule are tested here. |
| 7 | Copy ***perAuth*.authReqData** to an instance of *AuthReqData* and update the following field:<br>*authReqItem.authTags.transIDs.paySysID* → ***perAuth*.transIDs.paySysID**<br>Designate the result as ***authReqData***. |
| 8 | If ***requestSubs*** is TRUE, update the following contents of ***authReqData***:<br>*authReqItem.authReqPayload.subsequentAuthInd* → TRUE |
| 9 | If ***perAuth*.batchID** is not defined or if the merchant does not assign **BatchID**, continue with Step 11.<br>Otherwise, invoke "Determine batch identification" on page 472 with the following input:<br>***brand*** → ***trans*.brand**<br>***pBIN*** → ***trans*.pBIN**<br>***rrpid*** → *authRevRRTags.rrpid*<br>***origBatchID*** → ***perAuth*.batchID**<br>Designate the value of ***batchID*** returned as ***batchID*** and the value of ***batchSequenceNum*** returned as ***batchSequenceNum***. |

*Continued on next page*

# Merchant Generates AuthRevReq, continued

**Create AuthRevReq** (continued)

| Step | Action |
|---|---|
| 10 | If ***perAuth*.batchID** is defined, update the following components in ***authReqData***:<br>*saleDetail.batchID* → ***batchID***<br>*saleDetail.batchSequenceNum* → ***batchSequenceNum*** |
| 11 | Copy ***perAuth*.authResPayload** to an instance of *AuthResPayload*. If ***perAuth*.authCode** is *callIssuer*, update the following field:<br>*authHeader.responseData.authValCodes.approvalCode* → ***perAuth*.approvalCode** |
| 12 | Construct *AuthRevReqData*:<br>*authRevTags* — the result of Step 4<br>*mThumbs* — the result of Step 5<br>*authReqData* — the result of Step 7 (as updated in Step 8)<br>*authResPayload* — the result of Step 11<br>*authNewAmt* — ***newAmt***<br>*aRvRqExtensions* — any message extension(s) required to support additional business functions (optional) |
| 13 | Construct *AuthRevReqBaggage*:<br>*pi* — ~~***perAuth*.authToken if it exists; otherwise***~~ ***perAuth*.pi**<br>*capToken* — ***perAuth*.capToken** |
| 14 | Invoke "Compose *EncB*" on page 198 with the following input:<br>***s*** — the Merchant's signature certificate<br>***r*** — ***cert-PE***<br>***t*** — the result of Step 12<br>***b*** — the result of Step 13<br>***type-t*** — *id-set-content-AuthRevReqTBE*<br>***type-s*** — *id-set-content-AuthRevReqTBS*<br>***type-b*** — *id-set-content-AuthRevReqBaggage*<br>***certs*** — the new Merchant key encryption certificate for ***trans*.brandID**, if received since the last time a message was sent to this Payment Gateway |

*Continued on next page*

# Merchant Generates AuthRevReq, continued

**Create AuthRevReq** (continued)

| Step | Action |
|---|---|
| 15 | Store in the message database:<br>*authRevReqData* — the result of Step 12<br>*authRevReqBaggage* — the result of Step 13 |
| 16 | Update the following contents of ***perAuth***:<br>*authNewAmt* — ***newAmt***<br>*authRevDate* — *authRevRRTags.date*<br>*authRevRRPID* — *authRevRRTags.rrpid* |
| 17 | Store in the transaction database:<br>*perAuth* — the result of Step 16 |
| 18 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — ~~the Cardholder~~ the Payment Gateway<br>***msg*** — the result of Step 14<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — *AuthRevRRTags.rrpid*<br>***lid-C*** — ***trans*.transIDs.lid-C**<br>***lid-M*** — ***trans*.transIDs.lid-M**<br>***xID*** — ***trans*.transIDs.xID** |

*Continued on next page*

# Merchant Generates AuthRevReq, continued

### AuthRevReq data

| | |
|---|---|
| **AuthRevReq** | **EncB(M, P, AuthRevReqData, AuthRevReqBaggage)** |
| **AuthRevReqData** | **{AuthRevTags, [MThumbs], [AuthReqData], [AuthResPayload], AuthNewAmt, [ARvRqExtensions]}** |
| **AuthRevReqBaggage** | **{PI, [CapToken]}** |
| **AuthRevTags** | **{AuthRevRRTags, [AuthRetNum]}** |
| **MThumbs** | *Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.* |
| **AuthReqData** | *Copied from prior, corresponding* **AuthReq.** *Not required in message if* **CapToken** *generated by Payment Gateway contains all relevant data.* |
| **AuthResPayload** | *Copied from prior, corresponding* **AuthRes.** *Not required in message if* **CapToken** *generated by Payment Gateway contains all relevant data.* |
| **AuthNewAmt** | *New authorization amount requested. A value of zero indicates that the entire authorization should be reversed; any other value less than the ~~original~~ most recent authorized amount indicates a partial reversal. Full or partial reversals are used by Issuers to adjust the Cardholder's open to buy.* |
| **ARvRqExtensions** | *The data in an extension to the authorization reversal request shall ~~must~~ be financial and should be related to the processing of an authorization reversal (or subsequent capture) by the Payment Gateway, the financial network, or the Issuer.* |
| **PI** | *Copied from prior, corresponding* **AuthReq**. |
| **CapToken** | *Copied from prior, corresponding* **AuthRes**. |
| **AuthRevRRTags** | **RRTags** *page 395.*<br>*Fresh* **RRPID** *and* **Date** *for* **AuthRev** *pair.* |
| **AuthRetNum** | *Identification of the authorization request used within the financial network.* |

**Table 59: AuthRevReq Data**

# Payment Gateway Processes AuthRevReq

**Process AuthRevReq**

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *EnvelopedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>This procedure uses the following internal variables:<br>***transExists*** — an instance of *BOOLEAN*<br>***perAuthExists*** — an instance of *BOOLEAN*<br>***authRetNum*** — an instance of *AuthRetNum*<br>***authRevCode*** — an instance of *AuthRevCode* |
| 2 | Invoke "Verify *EncB*" on page 199 with the following input:<br>***d*** — ***msg***<br>***type-t*** — *id-set-content-AuthRevReqTBE*<br>***type-s*** — *id-set-content-AuthRevReqTBS*<br>***type-b*** — *id-set-content-AuthRevReqBaggage*<br>Designate:<br>• the value of ***t*** returned as ***req***,<br>• the value of ***b*** returned as ***baggage***, and<br>• the value of ***baggage*.pi** as ***pi***. |
| 3 | Validate the following contents of ***req*.authRevTags**:<br>*authRevRRTags.rrpid* — ***hdr*.rrpid**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** — *wrapperMsgMismatch* |
| 4 | ~~Verify PI by decrypting and matching against stored, previously verified PI from most recent authorization request. If they do not match, reject the reversal by sending a "piMismatch" AuthRevCode.~~<br>If ***pi*** is not in the list of used **PI**s or conditional **PI**s, set ***authRevCode*** to *piNeverAuthorized* and continue with Step 34. |
| 5 | If ***pi*** was used for an authorization request with an **AuthRRPID** other than ***req*.authReqItem.authTags.authRRTags.rrpid**, set ***authRevCode*** to *piAuthMismatch* and continue with Step 34. |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 6 | If ***pi*** was used for an authorization request other than the most recently approved authorization that has not been completely reversed, set ***authRevCode*** to *invalidReversal* and continue with Step 34. |
| 7 | If ***pi*** is in the list of conditional **PI**s and ***req*.authResPayload.authHeader.responseData.authValCodes.approvalCode** is not defined, set ***authRevCode*** to *piNeverAuthorized* and continue with Step 34. |
| 8 | If ***req*.authReqItem.authTags.authRRTags.rrpid** appears in the list of credited **RRPID**s or in the list of captured **RRPID**s, set ***authRevCode*** to *alreadyCaptured* and continue with Step 34.<br>If ***req*.authReqItem.authTags.authRRTags.rrpid** appears in the list of **CaptureNow RRPID**s, set ***authRevCode*** to *authDataMismatch* and continue with Step 34. |
| 9 | From the transaction database, retrieve the record for ~~*TransIDs in* **AuthRevTags**~~ ***hdr*.messageIDs.xid**. If not found:<br>• Set ***transExists*** to FALSE;<br>• Set ***perAuthExists*** to FALSE;<br>• Continue with Step 11.<br>Otherwise:<br>• Designate it as ***trans***.<br>• Set ***transExists*** to TRUE. |
| 10 | Retrieve from ***trans*** the **perAuth** record for the authorization request that corresponds to ***pi***. If found, designate it as ***perAuth***. Otherwise, set ***perAuthExists*** to FALSE. |
| 11 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* is *digitalSignature*,<br>• *issuer* matches ***msg*.signerInfos[1].issuerAndSerialNumber.issuer**, and<br>• *serialNumber* matches ***msg*.signerInfos[1].issuerAndSerialNumber.serialNumber**.<br>Designate it as ***cert-MS***. |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 12 | Invoke "Process **PI**" on page 516 with the following input:<br>***pi*** = ***pi***<br>***cert-MS*** = ***cert-MS***<br>***reversalFlag*** = TRUE<br>Designate:<br>• the value of ***trans*** returned as ***trans***, and<br>• the value of ***authTokenData*** returned as ***authTokenData***. |
| 13 | If ***baggage*.capToken** is not present:<br>• If a **CapToken** was returned in the most recent authorization response or authorization reversal response for the corresponding **AuthRRPID**, set ***authRevCode*** to *missingCapToken* and continue with Step 34.<br>• If either ***req*.authReqData** or ***req*.authResPayload** is not present, set ***authRevCode*** to *authDataMissing* and continue with Step 34.<br>• Otherwise, continue with Step 16. |
| 14 | Invoke "Process **CapToken**" on page 614 with the following input:<br>***capToken*** = ***baggage*.capToken**<br>***authRRPID*** = ***req*.authReqItem.authTags.authRRTags.rrpid**<br>If the value of ***capCode*** returned is not *success*:<br>• Map the value of ***capCode*** returned to a corresponding **AuthRevCode** value and set ***authRevCode*** to the result.<br>• Continue with Step 34. |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 15 | If both ***req*.authReqData** and ***req*.authResPayload** are not present, continue with Step 17. |
| 16 | Validate that the contents of ***req*.authReqData** and ***req*.authResPayload** match the data returned in the most recent authorization response or authorization reversal response for the corresponding **AuthRRPID** with the following exceptions:<br>• ***req*.authResPayload.authHeader.responseData.authValCodes.approvalCode** may contain a value that was not returned in the authorization response if ***req*.authResPayload.authHeader.authcode** is *callIssuer*; -<br>• ***req*.authReqData.saleDetail.batchID** and ***req*.authReqData.saleDetail.batchSequenceNum** may contain new values if the Merchant may determine the **BatchID**;<br>• ***req*.authReqData.authReqItem.authTags.transIDs.paySysID** may contain the value returned in the authorization response; and<br>• ***req*.authReqData.authReqItem.authReqPayload.subsequentAuthInd** may contain a different value.<br>If errors occur during validation, set ***authRevCode*** to *authDataMismatch* and continue with Step 34. |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>17</td><td>
If perAuthExists is FALSE:
<ul>
<li>Construct the following contents of PerAuth from trans, req.authReqData, req.authResPayload and capToken:
<table>
<tr><td>authAmt</td><td>a CurrencyAmount representing the amount of the corresponding authorization request (or the remaining amount after the most recent authorization reversal)</td></tr>
<tr><td>authCode</td><td>an AuthCode representing the result of the corresponding authorization request</td></tr>
<tr><td>authReqItem</td><td>an AuthReqItem representing the data of the corresponding authorization request</td></tr>
<tr><td>authRRPID</td><td>the RRPID of the corresponding authorization request</td></tr>
<tr><td>responseData</td><td>a ResponseData representing the results of the corresponding authorization response</td></tr>
<tr><td>captureNow</td><td>the CaptureNow flag of the corresponding authorization request</td></tr>
</table>
</li>
<li>If the CaptureNow flag of the corresponding authorization request was TRUE, construct the following contents of PerAuth from trans, req.authReqData, req.authResPayload and capToken:
<table>
<tr><td>batchID</td><td>the BatchID of the corresponding authorization request (optional)</td></tr>
<tr><td>batchSequenceNum</td><td>the BatchSequenceNum of the corresponding authorization request (optional)</td></tr>
<tr><td>capCode</td><td>a CapCode representing the result of the capture processing</td></tr>
<tr><td>saleDetail</td><td>a SaleDetail representing the data of the authorization request</td></tr>
</table>
</li>
</ul>
Designate the result as perAuth.
</td></tr>
<tr><td>18</td><td>
If perAuthExists is FALSE, store in the transaction record (created by "Process PI"):
<table>
<tr><td>perAuth</td><td>perAuth</td></tr>
</table>
Designate the resulting transaction record as trans.
</td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 19 | If ***req*.authNewAmt** is greater than or equal to ***perAuth*.authAmt** or if ***req*.authNewAmt** is not zero and ***perAuth*.captureNow** is TRUE:<br>• Set ***authRevCode*** to *invalidAmount*.<br>• Continue with Step 34. |
| 20 | If ***perAuth*.captureNow** is FALSE:<br>• If ***perAuth*.capCode** is defined, set ***authRevCode*** to ~~*originalProcessed*~~ *alreadyCaptured* and continue with Step 34.<br>• Otherwise, continue with Step 25. |
| 21 | If ***req*.authNewAmt** is not zero, set ***authRevCode*** to *invalidAmount* and continue with Step 34. |
| 22 | If the authorization processing corresponding to **AuthRRPID** was performed as a concurrent authorization and capture request, continue with Step 25. |
| 23 | If ***perAuth*.batchID** is not defined, continue with Step 25. |
| 24 | Invoke "Process batch identification" on page 487 with the following input:<br>***brand*** = ***trans*.brand**<br>***pBIN*** = ***cert-MS*.subject.commonName.BIN**<br>***rrpid*** = ***req*.authRevTags.rrpid**<br>***mBatchID*** = ***req*.authReqData.saleDetail.batchID**<br>***transType*** = *AuthRevReq*<br>***origBatchID*** = ***perAuth*.batchID**<br>Designate the value of ***batchID*** returned as ***batchID***, the value of ***capCode*** returned as ***capCode***, the value of ***batchData*** returned as ***batchData*** and the value of ***sameBatch*** returned as ***sameBatch***.<br>If ***capCode*** is not *success*:<br>• map the value of ***capCode*** to a corresponding **AuthRevCode** value and set ***authRevCode*** to the result,<br>• continue with Step 34. |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 25 | If ***req*.authNewAmt** is zero or if the payment card brand supports partial reversals:<br>• Process authorization reversal (either through existing payment card financial network or locally by the Payment Gateway if allowed by payment brand rules).<br>• Set ***authRevCode*** and ***authRetNum*** and format an instance of *ResponseData* based on the results of the authorization reversal process.<br>Otherwise, set ***authRevCode*** to *approved*. |
| 26 | If ***perAuth*.batchID** is not defined, continue with Step 30. |
| 27 | If ***authRevCode*** is not *approved*:<br>• Remove ***req*.authReqItem.authTags.authRRTags.rrpid** from ***batchData*.outstandingRequests**.<br>Note: This processing intentionally avoids updating **transactionDetailSeq** in ***batchData***.<br>• Continue with Step 30. |
| 28 | If batches are not accumulated locally:<br>• Process capture reversal via existing payment card financial network.<br>• Update ***capCode*** and set ***sequenceNum*** based on the results of the capture reversal process.<br>• Continue with Step 30. |
| 29 | If ***sameBatch*** is TRUE, invoke "Update batch (delete item)" on page 496 with the following input:<br>Otherwise, invoke "Update batch (add item)" on page 493 with the following input:<br>***trans*** — ***trans***<br>***perAuth*** — ***perAuth***<br>***rrpid*** — ***req*.authReqItem.authTags.authRRTags.rrpid**<br>***batchData*** — ***batchData***<br>***transAmt*** — ***req*.authNewAmt**<br>***transType*** — *AuthRevReq*<br>***payload*** — ***req*** |

*Continued on next page*

# Payment Gateway Processes AuthRevReq, continued

**Process AuthRevReq** (continued)

| Step | Action |
|---|---|
| 30 | If ***authRevCode*** is *approved* and if ***req*.authNewAmt** is zero:<br>• delete ***pi*** from list of used **PI**s or the list of conditional **PI**s, and<br>• if an **authToken** with the same **authRRPID** appears in the list of conditional **authToken**s, move it to the list of invalid **authToken**s. |
| 31 | If ***authRevCode*** is *approved* and if ***req*.authNewAmt** is not zero:<br>• delete ***pi*** from list of conditional **PI**s,<br>• add ***pi*** to list of used **PI**s (if it is not already on the list); and<br>• if an **authToken** with the same **authRRPID** appears in the list of conditional **authToken**s, move it to the list of invalid **authToken**s. |
| 32 | Copy ***perAuth*.authAmt** to an instance of *CurrencyAmount* and designate the result as ***priorAmt***. |
| 33 | If ***authRevCode*** is *approved* or if brand or acquirer policy requires the transaction record to be retained:<br>• Set ***transExists*** to TRUE and<br>• Set ***perAuthExists*** to TRUE. |
| 34 | Update the following contents of ***perAuth***:<br>*authAmt* — ***req*.authNewAmt** (if ***authRevCode*** is *success*)<br>*authRetNum* — ***authRetNum*** (if ***authRevCode*** is *success*)<br>*authRevCode* — ***authRevCode***<br>*capCode* — from the result of Step 25<br>*responseData* — from the result of Step 25 |
| 35 | Invoke "Create **AuthRevRes**" on page 577 with the following input:<br>***trans*** — ***trans***<br>***perAuth*** — ***perAuth***<br>***req*** — ***req***<br>***authTokenData*** — ***authTokenData***<br>***priorAmt*** — ***priorAmt***<br>***batchData*** — ***batchData***<br>***transExists*** — ***transExists***<br>***perAuthExists*** — ***perAuthExists***<br>***msgIds*** — ***hdr.messageIDs*** |

# Payment Gateway Generates AuthRevRes

### Create AuthRevRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***trans*** — the transaction record<br>***perAuth*** — authorization-specific transaction data<br>***req*** — an instance of *AuthRevReqData*<br>***authTokenData*** — an instance of *AuthTokenData* (optional)<br>***priorAmt*** — an instance of *CurrencyAmount*<br>***batchData*** — an instance of *BatchData*<br>***transExists*** — an instance of *BOOLEAN*<br>***perAuthExists*** — an instance of *BOOLEAN*<br>***msgIDs*** — an instance of *MessageIDs* |
| 2 | If **req.authNewAmt** is zero, continue with Step 7. |
| 3 | If ***perAuth*.authAmt** is greater than zero and is specified in a currency other than the one used by the cardholder and if the payment system returned the currency conversion data, construct *CurrConv*:<br>*currConvRate* — either:<br>• the current conversion rate between **AuthAmt** currency and Cardholder's requested currency, received from the payment system, or<br>• if the payment system returns the amount in the billing currency, *amountBillingCurrency* / ***perAuth*.authAmt**<br>*cardCurr* — the Cardholder's billing currency, received from the payment system |

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

**Create AuthRevRes** (continued)

| Step | Action |
|---|---|
| 4 | Construct *AuthHeader:*<br>*authAmt* — ***perAuth*.authAmt**<br>*authCode* — if ***perAuth*.authRevCode** is *success* then *success*; otherwise, ***perAuth*.authCode**<br>*responseData* — ***perAuth*.responseData**<br>*batchStatus* — optional: if ***perAuth*.batchID** is defined, ***batchData*.batchStatus**<br>*currConv* — the result of Step 3 |
| 5 | If ***capCode*** is specified, construct *CapResPayload*:<br>*capCode* — ***perAuth*.capCode**<br>*capAmt* — ***perAuth*.authAmt**<br>*batchID* — ***perAuth*.batchID** (if ***capCode*** is *success)*<br>*batchSequenceNum* — ***perAuth*.batchSequenceNum** (if ***capCode*** is *success)*<br>*cRsPayExtensions* — any message extension(s) required to support additional business functions (optional) |
| 6 | Construct *AuthResPayload:*<br>*authHeader* — the result of Step 4<br>*capResPayload* — the result of Step 5<br>*aRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 7 | Construct *AuthResDataNew*:<br>*transIDs* — ***trans*.transIDs**<br>*authResPayloadNew* — the result of Step 6 |
| 8 | Retrieve the current Payment Gateway key encryption certificate for ***trans*.brandID** and ***trans*.pBIN**.<br>If ***req*.mThumbs** is absent or if ***req*.mThumbs** is present and does not include the thumbprint of the certificate, designate the certificate as ***cert-PE*** and its Thumbprint as ***peThumb***; otherwise, set ***cert-PE*** and ***peThumb*** to NULL. |

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

**Create AuthRevRes** (continued)

| Step | Action |
|---|---|
| 9 | Retrieve the **BrandCRLIdentifier** for the brand identified by ***trans*.brand** and designate it as ***bci***; retrieve its Thumbprint and designate it as ***bciThumb***.<br>If ***req*.mThumbs** is present and includes ***bciThumb***, set ***bci*** to NULL. |
| 10 | Construct the following contents of *AuthRevTags:*<br>*authRevRRTags* — ***req*.authRevTags.authRevRRTags**<br>*authRetNum* — ***perAuth*.authRetNum** if it exists |
| 11 | Construct *AuthRevResData:*<br>*authRevCode* — ***perAuth*.authRevCode**<br>*authRevTags* — the result of Step 10<br>*brandCRLIdentifier* — ***bci***<br>*peThumb* — ***peThumb***<br>*authNewAmount* — ***perAuth*.authAmt**<br>*authResDataNew* — the result of Step 7<br>*aRvRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 12 | Invoke "Retrieve Merchant key encryption certificate" on page 537 with the following input:<br>***brandID*** — ***trans*.brandID**<br>***merchantID*** — ***trans*.merchantID** |
| 13 | If ***authRevCode*** is not *approved*, continue with Step 16. |
| 14 | ~~Optional:~~ If ***perAuth*.authAmt** is not zero and a **CapToken** was generated on the corresponding authorization request, invoke "Create **CapToken**" on page 533 with the following input:<br>***trans*** — ***trans***<br>***perAuth*** — ***perAuth***<br>Designate the value of ***capToken*** returned as ***capToken***. |

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

**Create AuthRevRes** (continued)

| Step | Action |
|---|---|
| 15 | If ***perAuth*.authAmt** is not zero and one of the following conditions exists:<br>• ***req*.subsequentAuthInd** is TRUE; or<br>• an **AuthToken** was generated for the corresponding authorization request from information in ***trans*.installRecurData**:<br>Then invoke "Create **AuthToken**" on page 535 with the following input:<br>***trans*** = ***trans***<br>***oldTokenData*** = ***authTokenData***<br>***authAmt*** = ***req*.authNewAmt**<br>***priorAmt*** = ***priorAmt***<br>Designate the value of ***authToken*** returned as ***authToken***. |
| 16 | If either ***capToken*** or ***authToken*** is defined, continue with Step 18.<br>Otherwise, invoke "Compose *Enc*" on page 186 with the following input:<br>***s*** = the Payment Gateway's signature certificate<br>***r*** = the result of Step 12<br>***t*** = the result of Step 11<br>***type-t*** = *id-set-content-AuthRevResTBE*<br>***type-s*** = *id-set-content-AuthRevResData*<br>***certs*** = ***cert-PE*** |
| 17 | Append the result of Step 16 to the tag [1]. Continue with Step 21. |
| 18 | Construct *AuthRevResBaggage:*<br>*capTokenNew* = ***capToken***<br>*authTokenNew* = ***authToken*** |

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

**Create AuthRevRes** (continued)

| Step | Action |
|---|---|
| 19 | Invoke "Compose *EncB*" on page 198 with the following input:<br>***s*** — the Payment Gateway's signature certificate<br>***r*** — the result of Step 12<br>***t*** — the result of Step 11<br>***b*** — the result of Step 18<br>***type-t*** — *id-set-content-AuthRevResTBEB*<br>***type-s*** — *id-set-content-AuthRevResTBS*<br>***type-b*** — *id-set-content-AuthRevResBaggage*<br>***certs*** — ***cert-PE*** |
| 20 | Append the result of Step 19 to the tag [0]. |
| 21 | Store in the message database:<br>*authRevResData* — the result of Step 11<br>*authRevResBaggage* — the result of Step 18 |
| 22 | If ***transExists*** is FALSE, delete the transaction record. Otherwise, if ***perAuthExists*** is FALSE, delete the **perAuth** entry in the transaction record.<br>Note: These actions remove unnecessary records created as a side effect of processing invalid authorization reversal requests. |
| 23 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — ~~the Cardholder~~ the Merchant<br>***msg*** — the result of Step 17 or 20<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — ***req*.authRevTags.rrpid**<br>***lid-C*** — ***msgIDs*.lid-C**<br>***lid-M*** — ***msgIDs*.lid-M**<br>***xID*** — ***msgIDs*.xID** |

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

### AuthRevRes data

| | |
|---|---|
| **AuthRevRes** | **< EncB(P, M, AuthRevResData, AuthRevResBaggage), Enc(P, M, AuthRevResData) >** |
| **AuthRevResData** | **{AuthRevCode, AuthRevTags, [BrandCRLIdentifier], [PEThumb], AuthNewAmt, AuthResDataNew, [ARvRsExtensions]}** |
| **AuthRevResBaggage** | **{[CapTokenNew], [AuthTokenNew]}** |
| **AuthRevCode** | *Enumerated code indicating outcome of payment authorization reversal processing. See page 583.* |
| **AuthRevTags** | *Copied from corresponding* **AuthRevReq** |
| **BrandCRLIdentifier** | *List of current CRLs for all CAs under a Brand CA. See page 347 in Part II.* |
| **PEThumb** | *Thumbprint of Payment Gateway certificate provided if* **AuthRevReq.MThumbs** *indicates Merchant needs one.* |
| **AuthNewAmt** | *Copied from corresponding* **AuthRevReq.** |
| **AuthResDataNew** | **{TransIDs, [AuthResPayloadNew]}**<br>*If* **AuthNewAmt** *is not 0, Payment Gateway creates a new instance of* **AuthResData** *(see "***AuthRes***" on page 538).* |
| **ARvRsExtensions** | *The data in an extension to the authorization reversal response shall ~~must~~ be financial and should be important for the processing of the authorization reversal response or a subsequent capture request by the Payment Gateway, the financial network, or the Issuer.* |
| **CapTokenNew** | *New Capture Token (with updated fields), if* **AuthNewAmt** *is not 0. This replaces the* **CapToken** *returned in the corresponding* **AuthRes**. |
| **AuthTokenNew** | *New Authorization Token (with updated fields). Merchant uses as the* **PI** *in a subsequent* **AuthReq**. *See "*AuthToken*" on page 378.* |
| **TransIDs** | *Copied from corresponding* **AuthRevReq.** |
| **AuthResPayloadNew** | *Formally identical to* **AuthResPayload** *(see page 539); if* **AuthNewAmt** *is not 0.* |

**Table 60: AuthRevRes Data**

*Continued on next page*

# Payment Gateway Generates AuthRevRes, continued

**AuthRevCode**

The following values are defined for **AuthRevCode**.

| | |
|---|---|
| approved | *The reversal is approved as requested.* |
| unspecifiedFailure | *The* **AuthRevReq** *could not be processed for a reason that does not appear elsewhere in this list.* |
| noReply | *No error is found with the* **AuthRevReq** *but the system is unable to process a reversal at this time. submit a new* **AuthRevReq** *later.* |
| expiredCard | *Supplied card expiration date (in Cardholder certificate or keyed by user) has passed.* |
| invalidTransaction | *No matching authorization transaction is found.* |
| systemError | *The request could not be processed by a non-SET system (Acquirer, financial network, Issuer, etc.) because data in the request is invalid.* |
| missingCapToken | **CapToken** was sent in **AuthRes** but not included in **AuthRevReq**. |
| invalidCapToken | Submitted **CapToken** does not match the original data. |
| invalidAmount | The **AuthNewAmt** is invalid. |

**Table 61: Enumerated Values for AuthRevCode**

# Merchant Processes AuthRevRes

**Process AuthRevRes**

Notes:

- A full **AuthRevReq** (that is, one in which **AuthNewAmt** is zero) makes the **PI** available for use again.
- If the **AuthRevReq** is a complete reversal in order to make it possible to reverse an earlier authorization, be sure to save data from the transaction record that you will need to construct the replacement authorization request.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *EnvelopedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Examine the tag at the beginning of ***msg***.<br>• If the tag is [0], continue with Step 3.<br>• Otherwise, continue with Step 4. |
| 3 | Invoke "Verify *EncB*" on page 199 with the following input:<br>***d*** — ***msg*** (without the leading tag [0])<br>***type-t*** — *id-set-content-AuthRevResTBEB*<br>***type-s*** — *id-set-content-AuthRevResTBS*<br>***type-b*** — *id-set-content-AuthRevResBaggage*<br>Designate:<br>• the value of ***t*** returned as ***res,*** and<br>• the value of ***b*** returned as ***baggage***.<br>Continue with Step 5. |
| 4 | Invoke "Verify *Enc*" on page 187 with the following input:<br>***d*** — ***msg*** (without the leading tag [1])<br>***type-t*** — *id-set-content-AuthRevResTBE*<br>***type-s*** — *id-set-content-AuthRevResData*<br>Designate the value of ***t*** returned as ***res***. |

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

| Step | Action |
|---|---|
| 5 | Validate the following contents of ***res***:<br>*authRevTags.rrpid* = ***hdr*.rrpid**<br>*authResDataNew.transIDs.lid-C* = ***hdr*.messageIDs.lid-C**<br>*authResDataNew.transIDs.lid-M* = ***hdr*.messageIDs.lid-M**<br>*authResDataNew.transIDs.xID* = ***hdr*.messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 6 | From the message database retrieve the instance of *AuthRevReqData* whose **authRevTags.rrpid** matches ***res*.authRevTags.rrpid** and designate it as ***req***. If not found, abort processing. |
| 7 | Retrieve the transaction record that is identified by ***res*.authResDataNew.transIDs.xid**, and designate it as ***trans***. If not found, abort processing. |
| 8 | Retrieve from ***trans*** the **perAuth** record whose **authRRPID** is ***req*.authReqData.rrpid** and designate it as ***perAuth***. If not found, abort processing. |
| 9 | Validate the following contents of ***res*.authRevTags**:<br>*lid-C* = ***trans*.lid-C**<br>*lid-M* = ***trans*.lid-M**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *unknownLID* |
| 10 | If ~~**GKThumb**~~ ***res*.peThumb** is present, verify that it matches the thumbprint of an existing Payment Gateway encryption certificate in the trusted cache. If not:<br>• From the untrusted cache, retrieve the key encryption certificate whose Thumbprint matches ***res*.peThumb** and designate it as ***cert-PE***. If not found, abort processing.<br>• Invoke "Verify certificate" on page 129 with the following input:<br>***cert*** = ***cert-PE*** |

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>11</td><td>If res.authRevCode is not approved, continue with Step 27.</td></tr>
<tr><td>12</td><td>If perAuth.captureNow is FALSE, continue with Step 15.</td></tr>
<tr><td>13</td><td>Invoke "Process batch information" on page 476 with the following input:
<table>
<tr><td>propBatchID</td><td>perAuth.authReqData.saleDetail.batchID</td></tr>
<tr><td>propSeqNum</td><td>perAuth.authReqData.saleDetail.batchSequenceNum</td></tr>
<tr><td>batchID</td><td>res.authResDataNew.authResPayloadNew.capResPayload.batchID</td></tr>
<tr><td>seqNum</td><td>res.authResDataNew.authResPayloadNew.capResPayload.batchSequenceNum</td></tr>
<tr><td>brand</td><td>trans.brand</td></tr>
<tr><td>pBIN</td><td>trans.pBIN</td></tr>
<tr><td>rrpid</td><td>perAuth.authRevRRPID</td></tr>
<tr><td>batchStatusSeq</td><td>a SEQUENCE with one item:<br>res.authResDataNew.authResPayloadNew.authHeader.batchStatus</td></tr>
<tr><td>transAmt</td><td>perAuth.authNewAmt</td></tr>
<tr><td>transType</td><td>AuthRevReq</td></tr>
<tr><td>origBatchID</td><td>perAuth.batchID</td></tr>
</table>
Designate the value of batchData returned as batchData.</td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>14</td><td>If TransactionDetail is not stored in BatchData, continue with Step 15.<br><br>If propBatchID is batchID, delete the TransactionDetail record with an authRRPID field equal to perAuth.authRRPID and continue with Step 15.<br><br>Otherwise:<br><br>• Construct the following contents of TransactionDetail:<br><table><tr><td>transIDs</td><td>trans.transIDs</td></tr><tr><td>authRRPID</td><td>perAuth.authRRPID</td></tr><tr><td>brandID</td><td>trans.brand</td></tr><tr><td>batchSequenceNum</td><td>res.authResPayload.capResPayload.batchSequenceNum</td></tr><tr><td>transactionAmt</td><td>res.authResPayload.capResPayload.capAmt</td></tr><tr><td>transactionAmtType</td><td>credit</td></tr><tr><td>transExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr></table>• Append the result to batchData.transactionDetailSeq. Store the updated batchData in the batch database.</td></tr>
<tr><td>15</td><td>If res.authRevCode is approved:<br>• If req.authNewAmt is zero, continue with Step 22.<br>• Otherwise, continue with Step 16.<br><br>Otherwise, continue with Step 27.</td></tr>
</table>

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

| Step | Action |
|---|---|
| | Processing steps for an approved partial reversal |
| 16 | Construct *AuthStatus*:<br>*authDate* — ***perAuth*.authRevDate**<br>*authCode* — *success*<br>*authRatio* — ***res*.authNewAmt ÷ *trans*.order.purchAmt**<br>*currConv* — ***res*.authResPayloadNew.currConv** if present |
| 17 | Construct *Results*:<br>*acqCardMsg* — ***perAuth*.pResPayload.results.acqCardMsg**<br>*authStatus* — the result of Step 16 |
| 18 | Construct *PResPayload*:<br>*completionCode* — *authorizationPerformed*<br>*results* — the result of Step 17<br>*pRsExtensions* — any message extension(s) required to support additional business functions (optional) |
| 19 | Update the following contents of ***perAuth***:<br>*authAmt* — ***res*.authNewAmt**<br>*authResPayload* — ***res*.authResPayloadNew**<br>*authRetNum* — ***res*.authRevTags.authRetNum**<br>*capToken* — ***baggage*.capTokenNew** if specified; otherwise NULL<br>*pResPayload* — the result of Step 18 |

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

| Step | Action |
|---|---|
| 20 | Store in the transaction database:<br>*perAuth* → ***perAuth***<br>*pi* → ***baggage*.authTokenNew** if specified; otherwise, NULL |
| 21 | Stop processing. |
| | Processing steps for an approved full reversal |
| 22 | Construct *PResPayload*:<br>*completionCode* → *orderReceived*<br>*pRsExtensions* → any message extension(s) required to support additional business functions (optional) |
| 23 | Delete the following contents of ***perAuth***:<br>• *capToken* |
| 24 | Update the following contents of ***perAuth***:<br>*authAmt* → ***res*.authNewAmt**<br>*authResPayload* → ***res*.authResPayloadNew**<br>*authRetNum* → ***res*.authRevTags.authRetNum**<br>*pResPayload* → the result of Step 22 |
| 25 | Store in the transaction database:<br>*perAuth* → ***perAuth***<br>*pi* → ***perAuth*.pi** |
| 26 | Stop processing. |

*Continued on next page*

# Merchant Processes AuthRevRes, continued

**Process AuthRevRes** (continued)

| Step | Action |
|---|---|
| | Processing steps for failed reversals |
| 27 | Delete the following contents of ***perAuth***:<br>• *authNewAmt*<br>• *authRevDate*<br>• *authRevRRPID* |
| 28 | Store in the transaction database:<br>*perAuth* → ***perAuth*** |
| 29 | Complete processing based on ***res*.authRevCode**:<br>*expiredCard* → No further processing required.<br>*noReply* → Save the details of the request to submit a new **AuthRevReq** at a later time.<br>*piMismatch*, *authDataMissing*, *authDataMismatch*, *missingCapToken*, *invalidCapToken*, *alreadyCaptured* → Merchant manual intervention is required. Conditions that might have led to these responses include:<br>• incorrect settings specifying the requirements of the Payment Gateway, and/or<br>• corruption of the transaction database.<br>any other value → Merchant manual intervention is required.<br>Note: No change has been made to the original authorization and its status has not been changed. |

# Section 4
# Capture Request/Response Processing

## Overview

### Introduction

The capture message pair includes a request from a Merchant to a Payment Gateway and a response from the Payment Gateway to the Merchant.

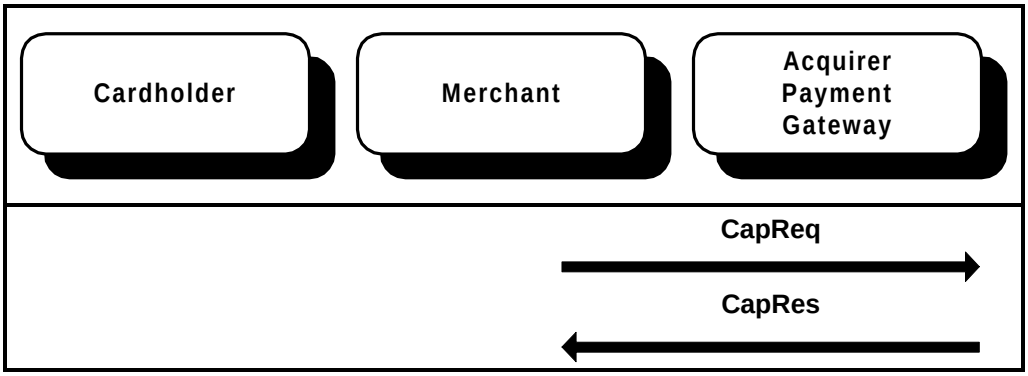


**Figure 8: CapReq/CapRes Message Pair**

### Purpose

This message pair provides the mechanism for completing the payment of moneys previously authorized ~~in one or more authorization transactions~~. ~~Amounts captured must be previously authorized using authorization messages.~~ The **CapReq** carries data from the Merchant necessary for the Payment Gateway to produce clearing request messages (for payment) that can be processed by the Acquirer or financial network for transmission to the Issuer. The **CapRes** returns the results of the attempted captures.

A single **CapReq** may contain multiple capture ~~tokens~~ items associated with distinct authorizations.

The Merchant shall not send a **CapReq** for a transaction that has already been successfully captured.

*Continued on next page*

# Overview, continued

### Variations

Capture may be accomplished by this message pair, although out-of-band methods of capture outside the scope of this protocol may also be used.

The amount captured may be restricted by payment brand or Acquirer rules such as:

- the total amount captured for a transaction may be required to be no more than (or within a certain percentage of) the amount indicated by the cardholder in the payment instructions; or
- the amount of a capture item may be required to be within a certain percentage of the corresponding authorization request.

See also "One capture per authorization" on page 465.

### Expired authorization

If an authorization has expired (according to rules which are outside the scope of SET), the Capture Request will fail. Instead, the Merchant may submit an Authorization Reversal Request (for the full amount authorized), followed by a new Authorization Request (which may or may not be approved).

### Capture amount vs. authorized amount

The capture amount can differ from the authorized amount, and may exceed it. For example, when the shipping amount is unknown at the time of authorization, it may be omitted or estimated, but when the capture is submitted, the exact amount of shipping will be applied.

Payment brand and acquirer rules determine permissible amounts. Generally speaking, the total amount for all shipments related to an order should be reasonably close to **PurchAmt**, the Purchase Amount, but SET does not provide any rules to enforce this.

- Ultimately, if the merchant has submitted an unreasonable capture request and the acquirer has permitted it to be processed, the cardholder can dispute the transaction.

# Merchant Prepares for CapReq

**Prepare for capture**

The Merchant application requires certain data to begin capture processing. The following processing sequence describe one method of obtaining that data.

Note: If this routine separates requests based on **BatchID** and multiple batches are open for a brand and BIN combination, this routine will need to incorporate the same logic for selecting a **BatchID** as is used in "Determine batch identification" on page 472.

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***perAuthSeq*** — a sequence of references to *PerAuth* entries in the transaction database |
| 2 | If the Payment Gateway requires *PANToken* to be included in the capture request, continue with Step 6. |
| 3 | Create an empty list of sequences of references to *PerAuth* entries in the transaction database and designate it as ***list***.<br>Each sequence in ***list*** will be uniquely identified by a combination of brand and BIN and will generate a separate **CapReq**. If the Payment Gateway only permits a single **BatchID** to appear in a capture request, the **BatchID** should also be used to identify each sequence. |

*Continued on next page*

# Merchant Prepares for CapReq, continued

**Prepare for capture** (continued)

| Step | Action |
|---|---|
| 4 | For each item in ***perAuthSeq***:<br>• Designate the item as ***perAuth***.<br>• If the authorization has been completely reversed, notify the operator and skip the item.<br>• If ***perAuth*.pResPayload.results.authStatus.authCode** is not *approved*, skip the item.<br>• If ***perAuth*.capCode** is *success*, skip the item.<br>• Retrieve the corresponding transaction record and designate it as ***trans***. If not found, abort processing.<br>• If ***trans*.brand** and ***trans*.pBIN** (and possibly **BatchID**) do not match the values assigned to an sequence in ***list***, add a new sequence to ***list*** (identified by ***trans*.brand** and ***trans*.pBIN** and possibly **BatchID**).<br>• If batch processing is used and there is no open batch corresponding to ***trans*.brand** and ***trans*.pBIN** (and possibly **BatchID**) and the batch must be open in order to invoke "Determine batch identification" on page 472, abort processing.<br>• Add ***perAuth*** to the corresponding sequence.<br>Notes:<br>• If the Payment Gateway limits the number of items in a capture request or the size of a capture request, a new sequence may need to be created to comply with those requirements.<br>• If the amount of the capture is different than the amount of the authorization, the field ***perAuth*.capAmt** must be defined. |
| 5 | For each sequence in ***list***:<br>• Designate as ***brand*** the value of the *BrandID* (without *Product*) that is associated with this sequence.<br>• Invoke "Create **CapReq**" on page 597 with the following input:<br>***perAuthSeq*** — the corresponding sequence in ***list***<br>***brand*** — ***brand***<br>Stop processing. |

*Continued on next page*

# Merchant Prepares for CapReq, continued

**Prepare for capture** (continued)

| Step | Action |
|---|---|
| | A separate **CapReq** will be generated for each item. |
| 6 | For each item in ***perAuthSeq***, perform Steps 7 through 9. |
| 7 | Designate the item as ***perAuth***. If ***perAuth*.capCode** is *success*, skip the item; otherwise, retrieve the corresponding transaction record and designate it as ***trans***. If not found, abort processing. |
| 8 | Designate ***trans*.brand** as ***brand***. |
| 9 | Invoke "Create **CapReq**" on page 597 with the following input: |

| | |
|---|---|
| ***perAuthSeq*** | a sequence with one item: a reference to ***perAuth*** |
| ***panRef*** | ***trans*.panRef** |
| ***brand*** | ***brand*** |

*Continued on next page*

## Merchant Prepares for CapReq, continued

**CapReqInfo**

For the purposes of this documentation, a logical record is defined containing data that applies to the capture request as a whole and is needed to process the capture response. The actual implementation of collecting and passing this data is at the discretion of the application developer. Processing steps included in "Create **CapReq**" describe one method of collecting the data.

| | |
|---|---|
| **capReqInfo** | **{ rrpid, perAuthSeq, brandID }** |
| **rrpid** | **RRPID** of the capture request/response pair |
| **perAuthSeq** | **{ perAuthRef + }** |
| **brand** | the **BrandID** (without *Product*) of all transactions in the Capture Request |
| **perAuthRef** | a reference to a *PerAuth* entry (and its corresponding transaction record) in the transaction database. |

**Table 62: CapReqInfo Data**

# Merchant Generates CapReq

### Create CapReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***perAuthSeq*** — a sequence of references to *PerAuth* entries in the transaction database<br>***panRef*** — a reference to a record in secure data storage (optional)<br>***brand*** — an instance of *BrandID* without *Product* |
| 2 | Create:<br>• an empty *CapItemSeq* and designate it as ***capItemSeq***;<br>• an empty *CapTokenSeq* and designate it as ***capTokenSeq*** |
| 3 | Generate a fresh statistically unique **RRPID** and designate it as ***rrpid***. |
| 4 | Create an instance of *GeneralizedTime*, populate it with the current date and time, and designate it as ***now***. |
| 5 | For each item in ***perAuthSeq***:<br>• Designate the item as ***perAuth***.<br>• Perform Steps 6 through 17. |
| | This processing is repeated for each set of input. |
| 6 | Retrieve the corresponding transaction record and designate it as ***trans***. |
| 7 | If batch processing is used and if the Merchant assigns **BatchID**, invoke "Determine batch identification" on page 472 with the following input:<br>***brand*** — ***brand***<br>***pBIN*** — ***trans*.pBIN**<br>***rrpid*** — ***rrpid***<br>Designate the value of ***batchID*** returned as ***batchID*** and the value of ***sequenceNum*** returned as ***sequenceNum***.<br>Note: If a single **BatchID** will apply to every item in the batch and the Merchant does not assign **BatchSequenceNum**, this invocation may precede Step 5. |

*Continued on next page*

# Merchant Generates CapReq, continued

**Create CapReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>8</td><td>Construct SaleDetail:
<table>
<tr><td>batchID</td><td>batchID</td></tr>
<tr><td>batchSequenceNum</td><td>sequenceNum</td></tr>
</table>
Populate other components of SaleDetail according to the type of transaction and brand policy.</td></tr>
<tr><td>9</td><td>Construct CapPayload:
<table>
<tr><td>capDate</td><td>now</td></tr>
<tr><td>capReqAmt</td><td>perAuth.capAmt if defined, otherwise, perAuth.authAmt</td></tr>
<tr><td>authReqItem</td><td>perAuth.authReqData.authReqItem (if required by the Payment Gateway and perAuth.capToken is not defined)</td></tr>
<tr><td>authResPayload</td><td>perAuth.authResPayload (if<br>• required by the Payment Gateway and perAuth.capToken is not defined; or<br>• perAuth.authCode is callIssuer)</td></tr>
<tr><td>saleDetail</td><td>the result of Step 8</td></tr>
<tr><td>cPayExtensions</td><td>any message extension(s) required to support additional business functions (optional)</td></tr>
</table></td></tr>
<tr><td>10</td><td>Construct CapItem:
<table>
<tr><td>transIDs</td><td>trans.transIDs</td></tr>
<tr><td>authRRPID</td><td>perAuth.authRRPID</td></tr>
<tr><td>capPayload</td><td>the result of Step 9</td></tr>
</table></td></tr>
<tr><td>11</td><td>Append the result of Step 9 to capItemSeq.</td></tr>
<tr><td>12</td><td>If perAuth.capToken exists, append it to capTokenSeq, otherwise, append a NULL to capTokenSeq.</td></tr>
</table>

*Continued on next page*

# Merchant Generates CapReq, continued

**Create CapReq** (continued)

| Step | Action |
|---|---|
| 13 | Update the following contents of ***perAuth***:<br>*capPayload* — the result of Step 9<br>*capRRPID* — ***rrpid*** |
| 14 | Store in the transaction database:<br>*perAuth* — ***perAuth*** |
| | The following processing is performed only once, after the data for each individual capture item has been created. |
| 15 | Designate ***trans*.pBIN** as ***pBIN***.<br>Note: Because all the transactions in a capture request have the same **pBIN**, it is sufficient to use the value stored in the last (or only) transaction record. |
| 16 | Construct *CapRRTags*:<br>*rrpid* — ***rrpid***<br>*merTermIDs* — from the Merchant profile<br>*date* — ***now*** |
| 17 | Construct *CapReqInfo* (see page 596):<br>*rrpid* — ***rrpid***<br>*perAuthSeq* — ***perAuthSeq***<br>*brand* — ***brand***<br>*panRef* — ***panRef*** |

*Continued on next page*

# Merchant Generates CapReq, continued

**Create CapReq** (continued)

| Step | Action |
|---|---|
| 18 | Recommended: Invoke "Create set of Thumbprints for request" on page 118 with the following input:<br>***brand*** = ***brand***<br>***bin*** = ***pBIN*** |
| 19 | Construct *CapReqData:*<br>*capRRTags* = the result of Step 16<br>*mThumbs* = the result of Step 18<br>*capItemSeq* = ***capItemSeq***<br>*cRqExtensions* = any message extension(s) required to support additional business functions (optional) |
| 20 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* includes *keyEncipherment* and<br>• *subject* matches ***trans.*peSubject**.<br>If found, designate the certificate as ***cert-PE***.<br>Otherwise, stop processing and display a message to the operator indicating that corrective action must be taken to obtain a current copy of the Payment Gateway certificate.<br>Notes:<br>• Because all the transactions in a capture request have the same **peSubject**, it is sufficient to use the value stored in the last (or only) transaction record.<br>• Under normal circumstances the certificate is retrieved every 24 hours using **PCertReq** and will be available in the trusted cache. |

*Continued on next page*

# Merchant Generates CapReq, continued

**Create CapReq** (continued)

| Step | Action |
|---|---|
| 21 | If ***panRef*** is specified, continue with Step 23.<br>Otherwise, invoke "Compose *EncB*" on page 198 with the following input:<br>***s*** — the Merchant's signature certificate<br>***r*** — ***cert-PE***<br>***t*** — the result of Step 19<br>***b*** — the result of Step 12<br>***type-t*** — *id-set-content-CapReqTBE*<br>***type-s*** — *id-set-content-CapReqTBS*<br>***type-b*** — *id-set-content-CapTokenSeq*<br>***certs*** — the new Merchant key encryption certificate for ***brandID***, if received since the last time a message was sent to this Payment Gateway |
| 22 | Append the result of Step 21 to the tag [0]. Continue with Step 26. |
| 23 | Construct the following contents of *PANToken* from the record in secure data storage identified by ***panRef***:<br>*pan* — PAN<br>*cardExpiry* — expiration date |
| 24 | Invoke "Compose *EncBX*" on page 203 with the following input:<br>***s*** — the Merchant's signature certificate<br>***r*** — ***cert-PE***<br>***t*** — the result of Step 19<br>***b*** — the result of Step 12<br>***p*** — the result of Step 23<br>***type-t*** — *id-set-content-CapReqTBEX*<br>***type-s*** — *id-set-content-CapReqTBSX*<br>***type-p*** — *id-set-content-PANToken*<br>***type-b*** — *id-set-content-CapTokenSeq*<br>***certs*** — the new Merchant key encryption certificate for ***brandID***, if received since the last time a message was sent to this Payment Gateway |

*Continued on next page*

# Merchant Generates CapReq, continued

**Create CapReq** (continued)

| Step | Action |
|---|---|
| 25 | Append the result of Step 24 to the tag [1]. |
| 26 | Store in the message database:<br>*capReqData* — the result of Step 19<br>*capReqInfo* — the result of Step 17 |
| 27 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** — ~~the Cardholder~~ the Payment Gateway<br>***msg*** — the result of either Step 22 or Step 25<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>***rrpid*** — ***rrpid***<br>***lid-C*** — if only a single set of input was received, ***trans*.transIDs.lid-C**<br>***lid-M*** — if only a single set of input was received, ***trans*.transIDs.lid-M**<br>***xID*** — if only a single set of input was received, ***trans*.transIDs.xID** |

*Continued on next page*

# Merchant Generates CapReq, continued

### CapReq data

| | |
|---|---|
| **CapReq** | **< EncB(M, P, CapReqData, CapTokenSeq), EncBX(M, P, CapReqData, CapTokenSeq, PANToken) >**<br>**CapTokenSeq** *is external "baggage".*<br>*If* **PANToken** *is included, it must correspond to a single* **CapItem** *and a single* **CapToken** *in* **CapTokenSeq**. |
| **CapReqData** | **{CapRRTags, [MThumbs], CapItemSeq, [CRqExtensions]}** |
| **CapTokenSeq** | **{[CapToken] +}**<br>*One or more* **CapTokens**, *in ordered one-to-one correspondence with* **CapItems** *in* **CapItemSeq**.<br>*Note: Any* **CapToken** *may be omitted; that is, may be* NULL. |
| **PANToken** | *See page 382* |
| **CapRRTags** | **RRTags,** *see page 395.*<br>*Fresh* **RRPID** *and* **Date.** |
| **MThumbs** | *Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.* |
| **CapItemSeq** | **{CapItem +}**<br>*One or more* **CapItem** *in an ordered array.* |
| **CRqExtensions** | *The data in an extension to the capture request shall ~~must~~ be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network, or the Issuer.*<br>*Note: The data in this extension applies to every item in the capture request; data related to a specific item should be placed in an extension to* **CapPayload**. |
| **CapToken** | *Copied from corresponding* **AuthRes** *(see page 538) or* **AuthRevRes** *(see page 582).* |
| **CapItem** | **{TransIDs, AuthRRPID, CapPayload}** |
| **TransIDs** | *Copied from corresponding* **AuthRes** *(see page 538) or* **AuthRevRes** *(see page 582).* |
| **AuthRRPID** | *The* **RRPID** *that appeared in the corresponding* **AuthReq** *(see page 506) or* **AuthRevReq** *(see page 568).* |
| **CapPayload** | *See page 604.* |

**Table 63: CapReq Data**

*Continued on next page*

# Merchant Generates CapReq, continued

### CapPayload data

| | |
|---|---|
| **CapPayload** | **{CapDate, CapReqAmt, [AuthReqItem], [AuthResPayload], [SaleDetail], [CPayExtensions]}** |
| **CapDate** | *Date of capture; this is the Transaction Date that will appear on the cardholder's statement.* |
| **CapReqAmt** | *Capture amount requested by Merchant, may differ from* **AuthAmt***; this is the Transaction Amount (before any currency conversion) that will appear on the cardholder's statement.* |
| **AuthReqItem** | *See "***AuthReq Data***" on page 506.*<br>*Required if the corresponding* **CapToken** *is not present or the Payment Gateway/Acquirer systems do not contain the relevant authorization request data.* |
| **AuthResPayload** | *See page 539.*<br>*Required if the corresponding* **CapToken** *is not present or the Payment Gateway/Acquirer systems do not contain the relevant authorization response data.* |
| **SaleDetail** | *See page 383.* |
| **CPayExtensions** | *The data in an extension to the capture request payload shall ~~must~~ be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network, or the Issuer.*<br>*Note: The data in this extension applies to an individual item in the capture request; data related to the entire capture request message should be placed in an extension to* **CapReqData***.* |

**Table 64: CapPayload Data**

# Payment Gateway Processes CapReq

### Process CapReq

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *EnvelopedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional)<br>This procedure uses the following internal variables:<br>***transExists*** — an instance of *BOOLEAN*<br>***perAuthExists*** — an instance of *BOOLEAN*<br>***capCode*** — an instance of *CapCode*<br>***storeLocally*** — an instance of *BOOLEAN* |
| 2 | Examine the tag at the beginning of ***msg***.<br>• If the tag is [0], continue with Step 3.<br>• Otherwise, continue with Step 4. |
| 3 | Invoke "Verify *EncB*" on page 199 with the following input:<br>***d*** — ***msg*** (without the leading tag [0])<br>***type-t*** — *id-set-content-CapReqTBE*<br>***type-s*** — *id-set-content-CapReqTBS*<br>***type-b*** — *id-set-content-CapTokenSeq*<br>Designate:<br>• the value of ***t*** returned as ***req***, and<br>• the value of ***b*** as ***capTokenSeq***.<br>Continue with Step 6. |
| 4 | Invoke "Verify *EncBX*" on page 205 with the following input:<br>***d*** — ***msg*** (without the leading tag [1])<br>***type-t*** — *id-set-content-CapReqTBEX*<br>***type-s*** — *id-set-content-CapReqTBSX*<br>***type-p*** — *id-set-content-PANToken*<br>***type-b*** — *id-set-content-CapTokenSeq*<br>Designate:<br>• the value of ***t*** returned as ***req***,<br>• the value of ***p*** returned as ***panToken***, and<br>• the value of ***b*** as ***capTokenSeq***. |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| 5 | Validate the following contents of ***req***:<br>*capRRTags.rrpid* = ***hdr*.rrpid**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 6 | If ***req*.capItemSeq** includes only one **capItem**, validate the following contents of ***req*.capItemSeq.capItem.transIDs**:<br>*lid-C* = ***hdr*.messageIDs.lid-C**<br>*lid-M* = ***hdr*.messageIDs.lid-M**<br>*xID* = ***hdr*.messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 7 | Create an empty *CapResItemSeq* and designate it as ***capResItemSeq***.<br>Create an empty sequence of *BatchID* and designate it as ***batchIDSeq***. |
| 8 | From the trusted cache, retrieve the certificate whose:<br>• *keyUsage* is *digitalSignature*,<br>• *issuer* matches ***msg*.signerInfos[1].issuerAndSerialNumber.issuer**, and<br>Designate the certificate as ***cert-MS***. |
| 9 | Designate ***cert-MS*.MerchantData.merAcquirerBIN** as ***pBIN***.<br>Designate ***cert-MS*.subject.organizationName** as ***brandID***. |
| 10 | For each **capItem** in ***req*.capItemSeq**:<br>• Designate the item as ***item***.<br>• Perform Steps 11 through 39. |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| | Processing for each **capItem** |
| 11 | Set ***capCode*** to *success*. |
| 12 | From the transaction database, retrieve the record for ***item*.transIDs.xid**. If not found:<br>~~• Set ***capCode*** ~~to *unknownXID* and continue with Step 34.~~<br>• Set ***transExists*** to FALSE;<br>• Set ***perAuthExists*** to FALSE;<br>• Continue with Step 16.<br>Otherwise:<br>• Designate it as ***trans***.<br>• Set ***transExists*** to TRUE. |
| 13 | Validate the following contents of ***item*.transIDs**:<br>*lid-C* — ***trans*.transIDs.lid-C**<br>*lid-M* — ***trans*.transIDs.lid-M**<br>If errors occur during validation, set ***capCode*** to *unknownLID* and continue with Step 34. |
| 14 | If the **perAuth** record for the authorization request that corresponds to ***item*.authRRPID** does not exist, set ***perAuthExists*** to FALSE.<br>Otherwise:<br>• retrieve the **perAuth** record and designate it as ***perAuth***; and<br>• set ***perAuthExists*** to TRUE. |
| 15 | If one of the following is TRUE:<br>• ***perAuthExists*** is TRUE and ***perAuth*.capCode** is *success*, or<br>• ***item*.authRRPID** appears on the list of captured **RRPID**s.<br>then set ***capCode*** to ~~*invalidCapToken*~~ *duplicateRequest* and continue with Step 34. |
| 16 | If the corresponding entry in ***capTokenSeq*** is present,<br>• Invoke "Process **CapToken**" on page 614 with the following input:<br>***capToken*** — the entry from ***capTokenSeq*** (including the leading tag)<br>• Designate the value of ***capCode*** returned as ***capCode*** and the value of ***capTokenData*** returned as ***capTokenData***. If ***capCode*** is not *success*, continue with Step 34. Otherwise, continue with Step 19. |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| 17 | If ~~capture without **CapToken** is not supported~~ a **CapToken** was returned in the most recent authorization response or authorization reversal response for ***item*.authRRPID**, set ***capCode*** to *capTokenMissing* and continue with Step 34. |
| 18 | If either ***item*.capPayload.authReqItem** or ***item*.capPayload.authResPayload** is not present, set ***capCode*** to *authDataMissing* and continue with Step 34. |
| 19 | If ***item*.capPayload.authReqItem** is present, validate that its contents match the data returned in the most recent authorization response or authorization reversal response for the ***item*.authRRPID**.<br>If errors occur during validation, set ***capCode*** to *invalidAuthData* and continue with Step 34. |
| 20 | If ***item*.capPayload.authResPayload** is present, validate that its contents match the data returned in the most recent authorization response or authorization reversal response for the ***item*.authRRPID** with the following exception:<br>• If ***item*.capPayload.authResPayload.authHeader.authcode** is *callIssuer*, then ***item*.capPayload.authResPayload.authHeader.responseData.authValCodes.approvalCode** may contain a value that was not returned in the authorization response.<br>If errors occur during validation, set ***capCode*** to *invalidAuthData* and continue with Step 34. |
| 21 | If ***item*.authRRPID** appears in the list of fully reversed **RRPID**s, set ***capCode*** to *invalidAuthData* and continue with Step 34. |
| 22 | If ***item*.capPayload.authResPayload.authHeader.responseData.authValCodes.approvalCode** is not present, set ***capCode*** to *invalidAuthData* and continue with Step 34. |
| 23 | Validate that the difference between ***item*.capPayload.capReqAmt** and the amount of the corresponding authorization request (or the remaining amount after the most recent authorization reversal) is within guidelines established by Acquirer or brand policy. If not, set ***capCode*** to *amountError* and continue with Step 34. |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| 24 | If ***item*.authRRPID** identifies an entry in the list of conditional **PI**s, move the conditional **PI** identified by ***item*.authRRPID** to the list of used **PI**s.<br>If an **authToken** identified by ***item*.authRRPID** appears in the list of conditional **authToken**s, delete it from that list (since the presence of the **ApprovalCode** means it is no longer conditional). |
| 25 | Validate the components of ***item*.capPayload.saleDetail** according to brand policy. If errors occur during validation, set ***capCode*** appropriately and continue with Step 34. |
| 26 | If ***perAuthExists*** is FALSE, construct the following contents of *PerAuth* from ***trans***, ***item*.capPayload.authReqItem**, ***item*.capPayload.authResPayload** and ***capTokenData***: |

| | |
|---|---|
| *authAmt* | a *CurrencyAmount* representing the amount of the corresponding authorization request (or the remaining amount after the most recent authorization reversal) |
| *authCode* | an *AuthCode* representing the result of the corresponding authorization request |
| *authReqItem* | an *AuthReqItem* representing the data of the corresponding authorization request |
| *authRRPID* | the **RRPID** of the corresponding authorization request |
| *responseData* | a *ResponseData* representing the results of the corresponding authorization response |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>27</td><td>If transExists is FALSE, store the corresponding values from item.capPayload.authReqItem, item.capPayload.authResPayload and capTokenData (or as otherwise noted) in the transaction database:
<table>
<tr><td>brand</td><td>brandID (without Product)</td></tr>
<tr><td>brandID</td><td>brandID</td></tr>
<tr><td>purchAmt</td><td>a CurrencyAmount representing the purchase amount of the transaction</td></tr>
<tr><td>pan</td><td>Either:<ul><li>panToken.pan (if present)</li><li>or from the value of panToken returned in Step 16</li></ul>Note: If not available, set capCode to authDataMissing and continue with Step 34.</td></tr>
<tr><td>cardExpiry</td><td>Either:<ul><li>panToken.cardExpiry (if present)</li><li>or from the value of panToken returned in Step 16</li></ul>Note: If not available, set capCode to authDataMissing and continue with Step 34.</td></tr>
<tr><td>transIDs</td><td>the TransIDs of the transaction</td></tr>
<tr><td>pBIN</td><td>pBIN</td></tr>
<tr><td>perAuth</td><td>the result of Step 26</td></tr>
</table>
Designate the resulting transaction record as trans.</td></tr>
<tr><td>28</td><td>If batch processing is not used, continue with Step 31.</td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| 29 | Invoke "Process batch identification" on page 487 with the following input:<br>***brand*** → ***trans*.brand**<br>***pBIN*** → ***pBIN***<br>***rrpid*** → ***req*.capRRTags.rrpid**<br>***mBatchID*** → ***item*.capPayload.saleDetail.batchID**<br>***transTypes*** → *CapReq*<br>If the value of ***capCode*** returned is not *success*:<br>• designate the value of ***capCode*** returned as ***capCode***, and<br>• continue with Step 34.<br>Otherwise, designate the value of ***batchID*** returned as ***batchID*** and the value of ***batchData*** returned as ***batchData***. |
| 30 | If ***batchID*** does not appear in ***batchIDSeq***, append ***batchID*** to ***batchIDSeq***. |
| 31 | If batches are not accumulated locally:<br>• Process capture via existing payment card financial network using transaction data from ***trans*** and ***perAuth***.<br>• Set ***capCode*** and ***sequenceNum*** (if provided) based on the results of the capture process.<br>• Continue with Step 33 |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>32</td><td>Invoke "Update batch (add item)" on page 493 with the following input:
<table>
<tr><td>trans</td><td>trans</td></tr>
<tr><td>perAuth</td><td>perAuth</td></tr>
<tr><td>rrpid</td><td>req.capRRTags.rrpid</td></tr>
<tr><td>batchData</td><td>batchData</td></tr>
<tr><td>sequenceNum</td><td>item.capPayload.saleDetail.<br>batchSequenceNum</td></tr>
<tr><td>transAmt</td><td>item.capPayload.capReqAmt</td></tr>
<tr><td>transType</td><td>CapReq</td></tr>
<tr><td>payload</td><td>item.capPayload</td></tr>
</table>
If the value of batchOK returned is FALSE, designate the value of capCode returned as capCode.<br><br>Otherwise, designate the value of sequenceNum returned as sequenceNum.</td></tr>
<tr><td>33</td><td>If capCode is success, add perAuth.authRRPID to the list of captured RRPIDs.</td></tr>
<tr><td>34</td><td>If capCode is success or if brand or acquirer policy requires the transaction record to be retained:<br><br>• set transExists to TRUE, and<br>• set perAuthExists to TRUE.<br><br>Otherwise, continue with Step 37.</td></tr>
<tr><td>35</td><td>Update the following contents of perAuth:
<table>
<tr><td>capCode</td><td>capCode</td></tr>
<tr><td>capDate</td><td>item.capPayload.capDate</td></tr>
<tr><td>capAmt</td><td>item.capPayload.capReqAmt</td></tr>
<tr><td>batchID</td><td>batchID</td></tr>
<tr><td>batchSequenceNum</td><td>sequenceNum</td></tr>
<tr><td>saleDetail</td><td>item.capPayload.saleDetail</td></tr>
</table></td></tr>
</table>

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapReq** (continued)

| Step | Action |
|---|---|
| 36 | Store in the transaction database:<br>*perAuth* — ***perAuth*** |
| 37 | Construct *CapResPayload:*<br>*capCode* — ***capCode***<br>*capAmt* — ***item*.capPayload.capReqAmt**<br>*batchID* — ***batchID***<br>*batchSequenceNum* — ***sequenceNum***<br>*cRsPayExtensions* — any message extension(s) required to support additional business functions (optional) |
| 38 | Construct *CapResItem*:<br>*transIDs* — ***item*.transIDs**<br>*authRRPID* — ***item*.authRRPID**<br>*capResPayload* — the result of Step 37 |
| 39 | Append the result of Step 38 to ***capResItemSeq***. |
| 40 | If ***transExists*** is FALSE, delete the transaction record. Otherwise, if ***perAuthExists*** is FALSE, delete the **perAuth** entry in the transaction record.<br>Note: These actions remove unnecessary records created as a side effect of processing invalid capture items. |
| | End of processing for each **capItem** |
| 41 | Invoke "Create **CapRes**" on page 616 with the following input:<br>***transIDs*** — ***item*.transIDs** (if a single set of input was received)<br>***itemSeq*** — ***capResItemSeq***<br>***batchIDSeq*** — ***batchIDSeq***<br>***req*** — ***req***<br>***brandID*** — ***brandID***<br>***pBIN*** — ***pBIN***<br>***merchantID*** — ***cert-MS*.MerchantData.merID** |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

### Process CapToken

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***capToken*** — an instance of *CapToken*<br>***authRRPID*** — an instance of *RRPID*<br>This procedure uses the following internal variables:<br>***capCode*** — an instance of *CapCode* |
| 2 | Set ***capCode*** to *success*. |
| 3 | Examine the tag at the beginning of ***capToken***.<br>• If the tag is [0], continue with Step 4.<br>• Otherwise, continue with Step 5. |
| 4 | Invoke "Verify *EncX*" on page 195 with the following input:<br>***d*** — ***capToken*** (without the leading tag [0])<br>***type-t*** — *id-set-content-CapTokenTBEX*<br>***type-s*** — *id-set-content-CapTokenData*<br>***type-p*** — *id-set-content-PANToken*<br>Designate:<br>• the value of ***t*** returned as ***capTokenData***, and<br>• the value of ***p*** returned as ***panToken***.<br>Continue with Step 6. |
| 5 | Invoke "Verify *Enc*" on page 187 with the following input:<br>***d*** — ***capToken*** (without the leading tag [1])<br>***type-t*** — *id-set-content-CapTokenTBE*<br>***type-s*** — *id-set-content-CapTokenData*<br>Designate the value of ***t*** returned as ***capTokenData***. |
| 6 | Verify that the entity identified by ***capToken*.signerInfos[1].IssuerAndSerialNumber** is the Payment Gateway. If not, set ***capCode*** to *invalidCapToken* and continue with Step 8. |

*Continued on next page*

# Payment Gateway Processes CapReq, continued

**Process CapToken** (continued)

| Step | Action |
|---|---|
| 7 | Validate the following contents of ***capTokenData***:<br>***authRRPID*** \| ***authRRPID***<br>If errors occur during validation, set ***capCode*** to *invalidCapToken*. |
| 8 | Return the following:<br>***capCode*** \| ***capCode***<br>***capTokenData*** \| ***capTokenData***<br>***panToken*** \| ***panToken*** |

*Continued on next page*

# Payment Gateway Generates CapRes

### Create CapRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***transIDs*** — an instance of *TransIDs* (optional)<br>***itemSeq*** — an instance of *CapResItemSeq*<br>***batchIDSeq*** — a sequence of *BatchID*<br>***req*** — an instance of *CapReqData*<br>***brandID*** — an instance of *BrandID*<br>***pBIN*** — an instance of *BIN*<br>***merchantID*** — an instance of *MerchantID* |
| 2 | Optional: Create an empty *BatchStatusSeq*; for each item in ***batchIDSeq***, optionally append the **BatchStatus** component of its **BatchData** record to the sequence. The status of other batches belonging to the Merchant may also be appended to the sequence.<br>Note: The mechanism to determine when batch status is to be returned as well as the mechanism to select batches for which information is to be returned is at the discretion of the Acquirer and the Payment Gateway vendor. |
| 3 | Retrieve the current Payment Gateway key encryption certificate for the brand identified by ***brandID*** and ***bin***. If not found, abort processing.<br>If ***req*.mThumbs** is absent or if ***req*.mThumbs** is present and does not include the thumbprint of the certificate, designate the certificate as ***cert-PE*** and its Thumbprint as ***peThumb***; otherwise, set ***cert-PE*** and ***peThumb*** to NULL. |
| 4 | Retrieve the **BrandCRLIdentifier** for the brand identified by ***brandID*** (without Product) and designate it as ***bci***; retrieve its Thumbprint and designate it as ***bciThumb***. If not found, abort processing.<br>If ***req*.mThumbs** is present and includes ***bciThumb***, set ***bci*** to NULL. |
| 5 | Construct *CapResData:*<br>*capRRTags* — ***req*.capRRTags**<br>*brandCRLIdentifier* — ***bci***<br>*peThumb* — ~~***GKThumb***~~ ***peThumb***<br>*batchStatusSeq* — the result of Step 2<br>*CapResItemSeq* — ***itemSeq***<br>*cRsExtensions* — any message extension(s) required to support additional business functions (optional) |

*Continued on next page*

# Payment Gateway Generates CapRes, continued

**Create CapRes** (continued)

| Step | Action |
|---|---|
| 6 | Invoke "Retrieve Merchant key encryption certificate" on page 537 with the following input:<br>***brandID*** = ***brandID***<br>***merchantID*** = ***merchantID*** |
| 7 | Invoke "Compose *Enc*" on page 186 with the following input:<br>***s*** = the Payment Gateway's signature certificate<br>***r*** = the result of Step 6<br>***t*** = the result of Step 5<br>***type-t*** = *id-set-content-CapResTBE*<br>***type-s*** = *id-set-content-CapResData*<br>***certs*** = ***cert-PE*** |
| 8 | Store in the message database:<br>*capResData* = the result of Step 5 |
| 9 | Invoke "Send Message" on page 109 with the following input:<br>***recip*** = the Merchant<br>***msg*** = the result of Step 7<br>***ext*** = any message extension(s) required to support additional business functions (optional)<br>***rrpid*** = ***req*.capRRTags.rrpid**<br>***lid-C*** = ***transIDs*.lid-C** (if specified)<br>***lid-M*** = ***transIDs*.lid-M** (if specified)<br>***xID*** = ***transIDs*.xID** (if specified) |

*Continued on next page*

# Payment Gateway Generates CapRes, continued

### CapRes data

| | |
|---|---|
| **CapRes** | **Enc(P, M, CapResData)** |
| **CapResData** | **{CapRRTags, [BrandCRLIdentifier], [PEThumb], [BatchStatusSeq], CapResItemSeq, [CRsExtensions]}** |
| **CapRRTags** | **RRTags** *(see page 395); copied from* **CapReq.** |
| **BrandCRLIdentifier** | *List of current CRLs for all CAs under a Brand CA . See page 347 in Part II.* |
| **PEThumb** | *Thumbprint of Payment Gateway certificate provided if* **CapReqData.MThumbs** *indicates Merchant needs one.* |
| **BatchStatusSeq** | **{BatchStatus +}** |
| **CapResItemSeq** | **{CapResItem +}**<br>*Order corresponds to* **CapReq.** |
| **CRsExtensions** | *The data in an extension to the capture response shall ~~must~~ be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network, or the Issuer.*<br>*Note: The data in this extension applies to every item in the capture response; data related to a specific item should be placed in an extension to* **CapResPayload**. |
| **BatchStatus** | *See page 396.* |
| **CapResItem** | **{TransIDs, AuthRRPID, CapResPayload}** |
| **TransIDs** | *Copied from corresponding* **CapReq**. |
| **AuthRRPID** | *The* **RRPID** *that appeared in the corresponding* **AuthReq** *or* **AuthRevReq***; copied from corresponding* **CapReq**. |
| **CapResPayload** | *See page 619.* |

**Table 65: CapRes Data**

*Continued on next page*

# Payment Gateway Generates CapRes, continued

### CapResPayload data

| | |
|---|---|
| **CapResPayload** | **{CapCode, CapAmt, [BatchID], [BatchSequenceNum], [CRsPayExtensions]}** |
| **CapCode** | *Enumerated code indicating status of capture. See page 620.* |
| **CapAmt** | *Copied from corresponding* **CapReq.** |
| **BatchID** | *Identification of the settlement batch for Merchant-Acquirer accounting; copied from corresponding* **CapReq**. |
| **BatchSequenceNum** | *The sequence number of this item within the batch; copied from corresponding* **CapReq.** |
| **CRsPayExtensions** | *The data in an extension to the capture response payload shall ~~must~~ be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network, or the Issuer.*<br>*Note: The data in this extension applies to an individual item in the capture response; data related to the entire capture response message should be placed in an extension to* **CapResData**. |

**Table 66: CapResPayload Data**

*Continued on next page*

# Payment Gateway Generates CapRes, continued

**CapCode**

The following values are defined for **CapCode**.

| | |
|---|---|
| success | *The capture item was successfully processed by the Payment Gateway.* |
| unspecifiedFailure | *The reason for the failure does not appear elsewhere in this list.* |
| duplicateRequest | *A capture request has already been processed for this transaction* (**XID** *and* **AuthRRPID**). |
| authExpired | *The authorization request was processed too long ago. The maximum time period is defined by brand or Acquirer rules.* |
| authDataMissing | *The authorization information was not present in the capture request.* |
| invalidAuthData | *The authorization information is not valid for this transaction.* |
| capTokenMissing | *The* **CapToken** *necessary to process this item was not present in the capture request.* |
| invalidCapToken | *The* **CapToken** *is not valid for this transaction.* |
| batchUnknown | *The batch for this item is unknown to the Payment Gateway.* |
| batchClosed | *The batch for this item has already been closed.* |
| unknownXID | *The* **XID** *is not recognized.* |
| unknownLID | **LID-C** *or* **LID-M** *is not recognized.* |

**Table 67: Enumerated Values for CapCode**

**Future values for CapCode**

The following conditions were identified after the ASN.1 for version 1.0 was completed. They are currently defined as constants mapping to *unspecifiedFailure*. In a future version of the ASN.1, these values will be added to the ENUMERATED **CapCode**. Application developers are encouraged to use these symbolic names in place of *unspecifiedFailure*.

| | |
|---|---|
| amountError | *The difference between the requested capture amount and the amount of the corresponding authorization request (or the remaining amount after the most recent authorization reversal) does not conform to guidelines established by Acquirer or brand policy.* |
| badSeqNum | *The Merchant provided a batch sequence number that has already been used.* |
| batchWrong | *The Merchant specified a* **batchID** *that is defined for a different brand and BIN combination.* |
| batchDataNeeded | *The Merchant must specify the* **batchID** *and* **batchSequenceNum**. |

**Table 68: Future Enumerated Values for CapCode**

# Merchant Processes CapRes

### Process CapRes

| Step | Action |
|---|---|
| 1 | Receive as input:<br>***hdr*** — an instance of *MessageHeader*<br>***msg*** — an instance of *EnvelopedData*<br>***ext*** — any message extension(s) required to support additional business functions (optional) |
| 2 | Invoke "Verify *Enc*" on page 187 with the following input:<br>***d*** = ***msg***<br>***type-t*** = *id-set-content-CapResTBE*<br>***type-s*** = *id-set-content-CapResData*<br>Designate the value of ***t*** returned as ***res***. |
| 3 | Validate the following contents of ***res***:<br>*capRRTags.rrpid* = ***hdr.*rrpid**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 4 | From the message database:<br>• retrieve the instance of *CapReqData* whose **rrpid** matches ***res*.capRRTags.rrpid** and designate it as ***req***;<br>• retrieve the instance of *CapReqInfo* whose **rrpid** matches ***res*.capRRTags.rrpid** and designate it as ***info***.<br>If either is not found, abort processing |
| 5 | Verify that the number of items in ***res*.capResItemSeq** is the same as the number of items in ***info*.perAuthSeq**. If not, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |

*Continued on next page*

# Merchant Processes CapRes, continued

**Process CapRes** (continued)

| Step | Action |
|---|---|
| 6 | If ***req*** includes only one item, validate the following contents of ***res*.capResItemSeq[1].capResItem.transIDs**:<br>*lid-C* = ***hdr*.messageIDs.lid-C**<br>*lid-M* = ***hdr*.messageIDs.lid-M**<br>*xID* = ***hdr*.messageIDs.xID**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input:<br>***errorCode*** = *wrapperMsgMismatch* |
| 7 | If ~~GKThumb~~ ***res*.peThumb** is present, verify that it matches the thumbprint of an existing Payment Gateway key encryption certificate in the trusted cache. If not:<br>• From the untrusted cache, retrieve the key encryption certificate whose Thumbprint matches ***res*.peThumb** and designate it as ***cert-PE***.<br>• Invoke "Verify certificate" on page 129 with the following input:<br>***cert*** = ***cert-PE*** |
| 8 | For each **capResItem** in ***res*.capResItemSeq**:<br>• Designate the item as ***item***.<br>• Designate the corresponding entry in ***info*.perAuthSeq** as ***perAuth***.<br>• Perform Steps 9 through 18. |
| | This processing is repeated for each set of input. |
| 9 | Retrieve the transaction record that corresponds to ***perAuth*** and designate it as ***trans***. If not found, abort processing. |

*Continued on next page*

# Merchant Processes CapRes, continued

**Process CapRes** (continued)

| Step | Action |
|---|---|
| 10 | Validate the following contents of ~~**CapRRTags**~~ ***item*.transIDs**:<br>*xid* — ***trans*.transIDs.xid**<br>*lid-C* — ***trans*.transIDs.lid-C**<br>*lid-M* — ***trans*.transIDs.lid-M**<br>If errors occur during validation, invoke "Create **Error** Message" on page 135 with the following input based on the field that failed:<br>***errorCode*** = *xid* → *unknownXID*; *lid-C* → *unknownLID*; *lid-M* → *unknownLID* |
| 11 | Invoke "Process batch information" on page 476 with the following input:<br>***propBatchID*** = ***perAuth*.capPayload.saleDetail.batchID**<br>***propSeqNum*** = ***perAuth*.capPayload.saleDetail.batchSequenceNum**<br>***batchID*** = ***item*.capResPayload.batchID**<br>***seqNum*** = ***item*.capResPayload.batchSequenceNum**<br>***brandID*** = ***trans*.brand**<br>***pBIN*** = ***trans*.pBIN**<br>***rrpid*** = ***hdr*.rrpid**<br>***transAmt*** = ***perAuth*.capPayload.capReqAmt**<br>***transType*** = *CapReq*<br>Designate the value of ***batchData*** returned as ***batchData***. |

*Continued on next page*

# Merchant Processes CapRes, continued

**Process CapRes** (continued)

| Step | Action |
|---|---|
| 12 | Optional:<br>• Construct the following contents of *TransactionDetail*:<br>*transIDs* → ***trans*.transIDs**<br>*authRRPID* → ***perAuth*.authRRPID**<br>*brandID* → ***trans*.brand**<br>*batchSequenceNum* → ***item*.capResPayload.batchSequenceNum**<br>*transactionAmt* → ***item*.capResPayload.capAmt**<br>*transactionAmtType* → *credit*<br>*transExtensions* → any message extension(s) required to support additional business functions (optional)<br>• Append the result to ***batchData*.transactionDetailSeq**. Store the updated ***batchData*** in the batch database. |
| 13 | Construct *CapStatus*:<br>*capDate* → ***perAuth*.capPayload.capDate**<br>*capCode* → ***item*.capResPayload.capCode**<br>*capRatio* → ***item*.capResPayload.capAmt ÷ *trans*.order.purchAmt** |
| 14 | Update the following components of ***perAuth*.pResPayload.results**:<br>*capStatus* → the result of Step 13 |
| 15 | If ***item*.capResPayload.capCode** is *success*, update the following components of ***perAuth*.pResPayload**:<br>*completionCode* → *capturePerformed* |
| 16 | Copy ***perAuth*.capPayload** to an instance of *CapPayload* and update the following contents:<br>*saleDetail.batchID* → ***item*.capResPayload.batchID**<br>*saleDetail.batchSequenceNum* → ***item*.capResPayload.batchSequenceNum** |

*Continued on next page*

# Merchant Processes CapRes, continued

**Process CapRes** (continued)

| Step | Action |
|---|---|
| 17 | ~~If ***capCode*** is *success,*~~ Update the following components of ***perAuth***:<br>*capAmt* → ***item*.capResPayload.capAmt**<br>*capCode* → ***item*.capResPayload.capCode**<br>*capPayload* → the result of Step 16<br>*capResPayload* → ***item*.capResPayload** |
| 18 | Store in the transaction database:<br>*perAuth* → the result of Step 17 |
| | End of processing for each set of input. |
| 19 | Delete from the message database the instance of *CapReqData* and the instance of *CapReqInfo* whose **rrpid** matches ***res*.capRRTags.rrpid**. |
| 20 | If ***res*.batchStatusSeq** is present, invoke "Process ***BatchStatus***" on page 479 with the following input:<br>***batchStatusSeq*** → ***res*.batchStatusSeq** |

# Stay tuned

**More to come**

We plan to release the processing steps for the following messages in about a week:

- Capture Reversal or Credit Data
- Capture Reversal Request/Response Processing
- Credit Request/Response
- Credit Reversal Request/Response Processing
- Payment Gateway Certificate Request/Response Processing
- Batch Administration Request/Response Processing