TRADE Working Group
INTERNET-DRAFT

Werner Hans
Hans-Bernhard.Beykirch
SIZ
Masaaki Hiroya
Yoshiaki Kawatsura
Hitachi

Expires: March 2001

September 2000

Payment API for v1.0 Internet Open Trading Protocol (IOTP)

<draft-ietf-trade-iotp-v1.0-papi-02.txt>

Status of this Memo

Abstract

The Internet Open Trading Protocol provides a data exchange format for trading purposes while integrating existing pure payment

protocols seamlessly. This motivates the multiple layered system

architecture which consists of at least some generic IOTP application core and multiple specific payment modules.

This document addresses the common interface between the IOTP application core and the payment modules, enabling the interoperability between these kinds of modules. Furthermore, such an interface provides the foundations for a plug-in- mechanism in actual implementations of IOTP application cores.

Such interfaces exist at the Consumers', the Merchants' and the Payment Handlers' installations connecting the IOTP application core and the payment software components/legacy systems.

Intended Readership

Software and hardware developers, development analysts, and users of payment protocols.

Copyright

Table of Contents

1. Introduction

Common network technologies are based on standardized and established Internet technologies. The Internet technologies provide mechanisms and tools for presentation, application development, network infrastructure, security, and basic data exchange.

Due to the presence of already installed trading roles' systems with their own interfaces (Internet shop, order management, payment, billing, and delivery management systems, or financial institute's legacy systems), IOTP has been limited to the common external interface over the Internet. However, some of these internal interfaces might be also standardized for better integration of IOTP aware components with of the existing infrastructure and its cost effective reuse.

The typical Payment Handlers, i.e., financial institutes or near-bank organizations, as well as Merchants require an IOTP aware application that easily fits into their existing financial infrastructure. The Payment Handler might even insist on the reuse of special in-house solutions for some subtasks of the IOTP aware application, e.g., reflecting their cryptography modules, gateway interfaces, or physical environment. Therefore, their IOTP aware implementation really requires such clear internal interfaces.

More important, Consumers demand modularization and clear internal interfaces: Their IOTP application aims at the support of multiple payment methods. Consumers prefer the flexible use of different seamless integrating payment methods within one trading application with nearly identical behavior and user interface. The existence of a well-defined interface enables payment software developers to bolt on their components to other developer's general IOTP Application Core.

Initially, this consideration leads to the two-level layered view of the IOTP software for each role, consisting of

o some generic IOTP system component, the so-called IOTP application core - providing IOTP based gateway services and generic business logic and

o the trading roles' specific backend systems implementing the specific trading transaction types' functionality.

In order to isolate the changes on the infrastructure, the IOTP trading application has been three-layered:

o the IOTP Application Core processes the generic parts of the IOTP transaction and holds the connection to the Internet,

- the Existing Legacy System or Existing Payment Software which

processes the actual transaction type, and particular payment transaction, and

o the IOTP Middleware or IOTP Payment Bridge which glues the other two possibly incompatible components. It brokers between the specific interface of the Existing Legacy System and the standardized interfaces of the IOTP Application Core.

As IOTP extends payment schemes to a trading scheme, primarily, this document focuses on payment modules, i.e. the interface between the IOTP Payment Bridge and the IOTP Application Core. It provides a standard method for exchanging payment protocol messages between the parties involved in a payment. But, it does not specify any interface for order or delivery processing.

Such a Payment Application Programmers Interface (API) must suit for a broad range of payment methods: (1) software based like Credit Card SET or CyberCoin, (2) chip card based like Mondex or GeldKarte, and (3) mimicries of typical and traditional payment methods like money transfer, direct debit, deposit, withdrawal, money exchange and value points. It should support both payments with explicit consumer acknowledge and automatic repeated payments, which have been consumer approved in advance.

The following discussion focuses on the Consumer's point of view and uses the associated terminology. When switching to Merchants' or Delivery Handlers' IOTP aware applications, the payment related components should be implicitly renamed by Existing Legacy Systems to the IOTP Middleware.

The next two sub-sections describe the general payment scenario and several assumptions about the coarsely sketched software components.

Chapter 2 illustrates the payment transaction progress and message flow of different kinds of transaction behavior. Chapters 3 to 4 provide the details of the API functions and Chapter 5 elaborates the call back interface.

1.1 General payment phases

The following table sketches the four logical steps of many payment schemes. The preceding agreements about the goods, payment method, purchase amount, or delivery rules are omitted.

Payment State	Party	Example Behavior
Mutual	Payment Handler	Generation of identification

Authentication request, solvency request, or

and Init-		some nonce
ialization	Consumer	Responses to the requests and generation of the own nonce
Authorization	Payment Handler	Generation of the authorization request (for consumer)
	Consumer	Agreement to payment (by reservation of the Consumer's e-money)
	Payment Handler	Acceptance or rejection of the agreement (consumer's authorization response), generation of the authorization request (for issuer/acquirer), and processing of its response
Capture		Generation of the capture request (for issuer/acquirer)
	Consumer	Is charged
	Payment Handler	Acceptance or rejection of the e-money, close of the payment transaction
Reversal		On rejection (online/delayed): generation of the reversal data
	Consumer	Receipt of the refund

However, some payment schemes

o limit themselves to one-sided authentication,
o perform offline authorization without any referral to any issuer/acquirer,
o apply capture processing in batch mode, or
o do not distinguish between authorization and capture,
o lack an inbound mechanism for reversals or implement a limited variant.

This model applies not only to payments at the typical points of sales but extends to refunds, deposits, withdrawals, electronic cheques, direct debits, and money transfers.

1.2 Assumptions

In outline, the IOTP Payment Bridge processes some input sequence of payment protocol messages being forwarded by the IOTP Application Core. It (1) disassembles the messages, (2) maps them onto the formats of the Existing Payment Software, (3) assembles its

responses, and (4) returns another sequence of payment protocol

messages that is mostly intended for transparent transmission by the IOTP Application Core to some IOTP aware remote party. Normally, this process continues between the two parties until the Payment Handler's Payment API signals the payment termination. Exceptionally, each system component may signal failures.

The relationship between the aforementioned components is illustrated in the following figure. These components might be related to each other in a flexible n-to-m-manner:

o One IOTP Application Core may manage multiple IOTP Payment Bridges and the latter might be shared between multiple IOTP Application Cores.
o Each Payment Bridge may manage multiple Existing Payment Software modules and the latter might be shared between multiple Payment Bridges.
o Each Existing Payment Software may manage multiple payment schemes (e.g. SET) and the latter might be supported by multiple Existing Payment Software modules.
o Each payment scheme may support multiple payment instruments (e.g. particular card) or methods (e.g. Visa via SET) and the latter might be shared by multiple Existing Payment Software Components.

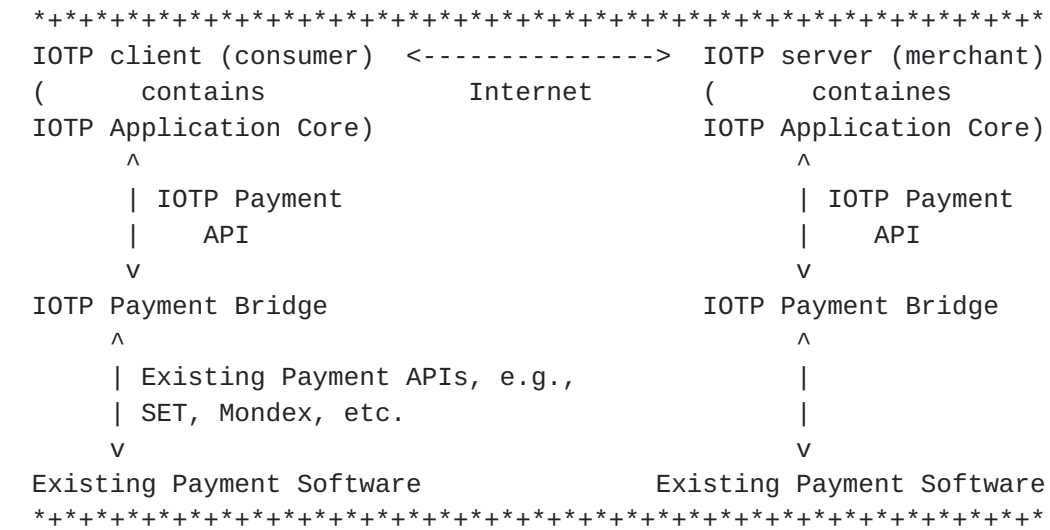


Figure 1: Relationship of the Components

The Payment API considers the following transaction types of Baseline IOTP [IOTP]:

o Baseline Purchase,
o Baseline Refund,
o Baseline Value Exchange,

- Baseline Withdrawal, and
- Baseline (Payment) Inquiry.

First, the authors' vision of the IOTP aware application's and its main components' capabilities are clarified: On the one hand, the Payment API should be quite powerful and flexible for sufficient connection of the generic and specific components. On the other hand, the Payment API should not be overloaded with nice-to-haves being unsupported by Existing Payment Software.

Despite the strong similarities on the processing of successful payments, failure resolution and inquiry capabilities differ extremely among different payment schemes. These aspects may even vary between different payment instrument using the same payment schemes. Eventually, the specific requirements of Consumers, Merchants and Payment Handlers add variance and complexity. Therefore, it is envisioned that the IOTP Application Core provides only very basic inquiry mechanisms while complex and payment scheme specific inquiries, failure analysis, and failure resolution are fully deferred to the actual Existing Payment Software - including the user interface.

The IOTP Application Core processes payments transparently, i.e., it forwards the wrapped payment scheme specific messages to the associated IOTP Payment Bridge/Existing Payment Software. The Existing Payment Software might even use these messages for inbound failure resolution. It reports only the final payment status to the IOTP Application Core or some intermediate - might be also final - status on abnormal interruption.

The IOTP Application Core implements the generic and payment scheme independent part of the IOTP transaction processing and provides the suitable user interface. Focusing on payment related tasks, it

o manages the registered IOTP Payment Bridges and provides a mechanism for their registration - the latter is omitted by this document.

o assumes that any IOTP Payment Bridge is a passive component, i.e., it strictly awaits input data and generates one response to each request,

o supports the payment negotiation (Consumer: selection of the actual payment instrument or method; Merchant: selection of the payment methods being offered to the Consumer) preceding the payment request,

o requests additional payment specific support from the Existing Payment Software via the selected and registered the IOTP Payment Bridge,

o initializes and terminates the Existing Payment Software via the IOTP Payment Bridge,

o inquires authentication data (for subsequent request or response) from the Existing Payment Software, specific authentication component - omitted in this document - or Consumer (by a suitable user interface),

o supervises the online transaction process and traces its progress,

o stores the transaction data being exchanged over the IOTP wire - payment scheme specific data is handled transparently,

o relates each payment transaction with multiple payment parameters (IOTP Transaction Identifier, Trading Protocol Options, Payment Instrument/Method, Offer Response, IOTP Payment Bridge, and Wallet Identifier, associated remote Parties). The relation might be lowered to the party's Payment Identifier, IOTP Payment Bridge, Wallet Identifier, and the remote parties when the actual payment transaction has been successfully started.

o implements a payment transaction progress indicator,

o enables the inquiry of pending and completed payment transactions,

o implements generic dialogs, e.g., brand selection, payment acknowledge, payment suspension / cancellation, receipt visualization, basic transaction inquiry, balance inquiry, or receipt validation,

o defers payment specific processing, supervision, validation, and error resolution to the Existing Payment Software. It is expected, that the Existing Payment Software tries to resolve many errors first by the extended exchange of Payment Exchange messages. The most significant and visible failures arise from sudden unavailability or lapses of the local or opposing payment component.

o supports the invocation of any Existing Payment Software in an interactive mode, which might be used (1) for the payment scheme specific post-processing of a (set of) payment transactions, (2) for the analysis of a payment instrument, (3) for the registration of a new payment instrument/scheme, or (4) re-configuration of a payment instrument/scheme.

o exports call back functions for use by the IOTP Payment Bridge or Existing Payment Software for progress indication.

In addition, the IOTP Application Core

o manages the IOTP message components and IOTP message blocks exchanged during the transaction which may be referenced and accessed during the processing of subsequent messages, e.g., for signature

verification. In particular, it stores named Packaged Content

elements exchanged during payments.

o manages several kinds of identifiers, i.e., transaction, message, component, and block identifiers,

o implements a message caching mechanism,

o detects time-outs at the protocol and API level reflecting the communication with both the IOTP aware remote party and the Payment API aware local periphery, e.g., chip card (reader) may raise time-outs.

However, the IOTP Payment Bridge and Existing Payment Software do not have to rely on all of these IOTP Application Core's capabilities. E.g., some Consumer's Existing Payment Software may refuse the disclosure of specific payment instruments at brand selection time and may delay this selection to the "Check Payment Possibility" invocation using its own user interface.

The IOTP Payment Bridge's capabilities do not only deal with actual payments between the Consumer and the Payment Handler but extend to the following:

o translation and (dis)assemblage of messages between the formats of the IOTP Payment API and those of the Existing Payment Software. Payment API requests and response are strictly 1-to-1 related.

o Consumer's payment instrument selection by the means of an unsecured/public export of the relationship of payment brands, payment protocols, and payment instruments (identifiers). Generally, this includes not just the brand (Mondex, GeldKarte, etc.) but also which specific instance of the instrument and currency to use (e.g. which specific Mondex card and which currency of all those available).

However, some Existing Payment Software may defer the selection of the payment instrument to the actual payment carrying-out or it may even lack any management of payment instruments. E.g., chip card based payment methods may offer - Point of Sale like - implicit selection of the payment instrument by simple insertion of the chip card into the chip card reader or it interrogates the inserted card and requests an acknowledge (or selection) of the detected payment instrument(s).

o payment progress checks, e.g., is there enough funds available to carry out the purchase, or enough funds left for the refund,

o IOTP Payment Receipt checks which might be performed over its Packaged Content or by other means.

o recoding of payment scheme specific receipts into a format which can be displayed to the user or printed,

o cancellation of payment, even though it is not complete,

o suspension and resumption of payment transactions. Two kinds of failures the Existing Payment Software might deal with are (1) the time-out of the network connection and (2) lack of funds. For resolution, the IOTP Application Core may try the suspension with a view to later possible resumption.

o recording the payment progress and status on a database. E.g., information about pending payments might be used to assist their continuation when the next payment protocol message is received.

o payment transaction status inquiry, so that the inquirer - IOTP Application Core or User - can determine the appropriate next step.

o balance inquiry or transaction history, e.g. consumers may interrogate their chip card based payment instrument or remotely administer some account in advance of a payment transaction acknowledge,

o inquiry on abnormal interrupted payment transactions, which might be used by the IOTP Application Core to resolve these pending transactions at startup (after power failure).

o payment progress indication. This could be used to inform the end user of details on what is happening with the payment.

o payment method specific authentication methods.

Existing Payment Software may not provide full support of these capabilities. E.g., some payment schemes may not support or even prevent the explicit transaction cancellation at arbitrary phases of the payment process. In this case, the IOTP Payment Bridge has to implement at least skeletons that signal such lack of support by the use of specific error codes (see below).

The Existing Payment Software's capabilities vary extremely. It

o supports payment scheme specific processing, supervision, validation, and error resolution. It is expected, that many errors are tried to be resolved first by the extended exchange of Payment Exchange messages.

o provides hints for out-of-band failure resolution on failed inbound resolution - inbound resolution is invisible to the IOTP Application Core.

o may implement arbitrary transaction data management and inquiry mechanisms ranging from no transaction recording, last transaction recording, chip card deferred transaction recording, simple transaction history to sophisticated persistent data management with flexible user inquiry capabilities. The latter is required by Payment Handlers for easy and cost effective failure resolution.

o implements the payment scheme specific dialog boxes.

Even the generic dialog boxes of the IOTP Application Core might be unsuitable: Particular (business or scheme) rules may require some dedicated appearance / structure / content or the dialog boxes, may prohibit the unsecured export of payment instruments, or may prescribe the pass phrase input under its own control.

2. Message Flow

The following lists all functions of the IOTP Payment API:

o Brand Compilation Related API Functions

"Find Accepted Payment Brand" identifies the accepted payment brands for any indicated currency amount.

"Find Accepted Payment Protocol" identifies the accepted payment protocols for any indicated currency amount (and brand) and returns payment scheme specific packaged content for brand selection purposes.

This function might be used in conjunction with the aforementioned function or called without any brand identifier.

"Get Payment Initialization Data" returns additional payment scheme specific packaged content for payment processing by the payment handler.

"Inquire Authentication Challenge" returns the payment scheme specific authentication challenge value.

"Check Authentication Response" verifies the returned payment scheme specific authentication response value.

"Change Process State" is used (here only) for abnormal termination. (cf. Payment Processing Related API Functions).

o Brand Selection Related API Functions

"Find Payment Instrument" identifies which instances of a payment

instrument of a particular payment brand are available for use in a payment.

"Check Payment Possibility" checks whether a specific payment instrument is able to perform a payment.

"Authenticate" forwards any payment scheme specific authentication data to the IOTP Payment Bridge for processing.

"Change Process State" is used (here only) for abnormal termination. (cf. Payment Processing Related API Functions).

o Payment Processing Related API Functions

"Start or Resume Payment Consumer/Payment Handler" initiate or resume a payment transaction. There exist specific API functions for the two trading roles Consumer and Payment Handler.

"Continue Process" forwards payment scheme specific data to the Existing Payment Software and returns more payment scheme specific data for transmission to the counter party.

"Change Process State" changes the current status of payment transactions. Typically, this call is used for successful/- less termination or suspension.

o General Inquiry API Functions

"Remove Payment Log" notifies the IOTP Payment Bridge that a particular entry has been removed from the Payment Log of the IOTP Application Core.

"Payment Instrument Inquiry" retrieves the properties of Payment Instruments.

"Inquire Pending Payment" reports any abnormal interrupted payment transaction known by the IOTP Payment Bridge.

Payment Processing Related Inquiry API Functions

"Check Payment Receipt" checks the consistency and validity of IOTP Payment Receipts, received from the Payment Handler or returned by "Inquire Process State" API calls. Typically, this function is called by the Consumer during the final processing of payment transactions. Nevertheless, this check might be advantageous both for Consumers and Payment Handlers on failure resolution.

"Expand Payment Receipt" expands the Packaged Content of IOTP Payment Receipts as well as payment scheme specific payment receipts into a

form which can be used for display or printing purposes.

"Inquire Process State" responds with the payment state and the IOTP Payment Receipt Component. Normally, this function is called by the Payment Handler for final processing of the payment transaction.

"Start Payment Inquiry" prepares the remote inquiry of the payment transaction status and responds with payment scheme specific data that might be needed by the Payment Handler for the Consumer initiated inquiry processing.

"Inquire Payment Status" is called by the Payment Handler on Consumer initiated inquiry requests. This function returns the payment scheme specific content of the Inquiry Response Block.

"Continue Process" and "Change Process State" (cf. Payment Processing Related API Calls)

o Other API Functions

"Manage Payment Software" enables the immediate activation of the Existing Payment Software. Further user input is under control of the Existing Payment Software.

"Call Back" provides a general interface for the visualization of transaction progress by the IOTP Application Core.

The following table shows which API functions must (+), should (#), or might (?) be implemented by which Trading Roles.

API function	Consumer	Payment Handler	Merchant
Find Accepted Payment Brand			+
Find Accepted Payment Protocol			#
Find Payment Instrument	+		
Get Payment Initialization Data			+
Check Payment Possibility	+		
Start Payment Consumer	+		
Start Payment Payment Handler		+	
Resume Payment Consumer	#		
Resume Payment Payment Handler		#	
Continue Process	+	+	
Inquire Process State	+	+	?
Change Process State	+	+	?
Check Payment Receipt	+	?	
Expand Payment Receipt	#	?	
Remove Payment Log	?	?	?

Inquire Authentication Challenge			?
Authenticate	+		
Check Authentication Response			?
Payment Instrument Inquiry	?		
Inquire Pending Payment	#	#	
Start Payment Inquiry	?		
Inquire Payment Status		?	
Manage Payment Software	#	?	?
Call Back	#		

Table 1: Requirements on API Functions by the Trading Roles

The next sections sketch the relationships and the dependencies between the API functions. They provide the informal description of the progress alternatives and depict the communication and synchronization between the general IOTP Application Core and the payment scheme specific modules.

2.1 Authentication Documentation Exchange

This section describes how the functions in this document are used together to process authentication.

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Authenticator   Inquire Authentication Challenge(Alg1*)   -> IPB
                Inq. Auth. Challenge Response(Alg1,Ch1)   <- IPB
                . . .
                Inquire Authentication Challenge(Algn*)   -> IPB
                Inq. Auth. Challenge Response(Algn,Chn)   <- IPB
                Create and transmit Authentication Request Block
Authenticatee   Authenticate(Alg1, Ch1)                   -> IPB
                AuthenticateResponse(...)                 <- IPB
                . . .
                Authenticate(Algm, Chm)                   -> IPB
                AuthenticateResponse(Res)                 <- IPB
                Create and transmit Authentication Response Block
Authenticator   Check Authentication Response(Algm,Chm,Res)->IPB
                Check Auth. Response()                     <-IPB
                Create and transmit Authentication Status Block
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
```

Figure 12 Authentication Message Flows

1. (Authenticator Process) None, one or multiple IOTP Payment Bridges (IPB) are requested for one or multiple authentication challenge values ("Inquire Authentication Challenge"). Each value is

encapsulated in an IOTP Authentication Request Component. In

addition, the IOTP Application Core may add payment scheme independent authentication methods. All of them form the final IOTP Authentication Request Block, which describes the set of authentication methods being supported by the authenticator and from which the authenticatee has to choose one method.

Note that the interface of the API function is limited to the response of exactly one algorithm per call. If the IOTP Application Core provides a choice of algorithms for input, this choice should be reduced successively by the returned algorithm ({Alg(i+1)*} is subset of {Algi*}).

During the registration of new Payment Instruments, the IOTP Payment Bridge notifies the IOTP Application Core about the supported authentication algorithms.

2. On presence of an IOTP Authentication Block within the received IOTP message, the Authenticatee's IOTP Application Core checks whether the IOTP transaction type in the current phase actually supports the authentication process.

For each provided Authentication Request Component, the IOTP Application Core analyzes the algorithms' names, the transaction context, and optionally user preferences in order to determine the system components which are capable to process the authentication request items. Such system components might be the IOTP Application Core itself or any of the registered IOTP Payment Bridges.

Subsequently, the IOTP Application Core requests the responses to the supplied challenges from the determined system components in any order. The authentication trials stop with the first successful response, which is included in the IOTP Authentication Response Block.

Alternatively, the IOTP Application might ask for a user selection. This might be appropriate, if two or more authentication algorithms are received that require explicit user interaction, like PIN or chip card insertion.

The Authenticatee's organizational data is requested by an IOTP Authentication Request Block without any content element. On failure, the authentication (sequence) might be retried, or the whole transaction might be suspended or cancelled.

3. (Authenticator Process) The IOTP Application Core checks the presence of the IOTP Authentication Response Component in the Authentication Response Block and forwards its content to the generator of the associated authentication challenge for verification ("Check Authentication Response").

On sole organizational data request, its presence is checked.

Any verification must succeed in order to proceed with the transaction.

2.2 Brand Compilation

The following shows how the API functions are used together so that the Merchant can (1) compile the Brand List Component, (2) generate the Payment Component, and (3) adjust the Order Component with payment scheme specific packaged content.

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Merchant      For each registered IOTP Payment Bridge
              |  Find Accepted Payment Brand()              -> IPB
              |  Find Accepted Payment Brand Response (B*) <- IPB
              |  Find Accepted Payment Protocol(B1)         -> IPB
              |  Find Accepted Payment Protocol Res.(P1*)   <- IPB
              |  . . .
              |  Find Accepted Payment Protocol(Bn)         -> IPB
              |  Find Accepted Payment Protocol Res.(Pn*)   <- IPB
              Create one Brand List Component, ideally sharing
                common Brand, Protocol Amount, Currency Amount,
                and Pay Protocol Elements
              Create Trading Protocol Options Block
              On brand independent transactions
              |  Create Brand Selection Component, implicitly
              |  Get Payment Initialization Data(B1,P1)   -> IPB
              |  Get Payment Initialization Data Res.()   <- IPB
              |  Optionally
              |  |  Inquire Process State()               -> IPB
              |  |  Inquire Process State Response(State) <- IPB
              |  Create Offer Response Block
              Transmit newly created Block(s)
Consumer      Consumer selects Brand (Bi)/Currency/Protocol (Pj)
                from those that will work and generates Brand
                Selection Component - at least logically
              On brand dependent transaction
              |  Transmit Brand Selection Component
Merchant      On brand dependent transaction
              |  Get Payment Initialization Data(Bi,Pj)   -> IPB
              |  Get Payment Initialization Data Res.()   <- IPB
              |  Optionally
              |  |  Inquire Process State()               -> IPB
              |  |  Inquire Process State Response(State) <- IPB
              |  Create Offer Response Block
```

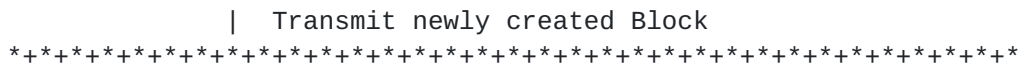

| Transmit newly created Block

Figure 3 Brand Compilation Message Flows

1. The Merchant's commerce server controls the shopping dialog with its own mechanisms until the Consumer checks out the shopping cart and indicates the payment intention. The notion shopping subsumes any non-IOTP based visit of the Merchant Trading Role's (which subsumes Financial Institutes) web site in order to negotiate the content of the IOTP Order Component. The subsequent processing switches to the IOTP based form by the activation of the Merchant's IOTP aware application.

2. The IOTP Application Core inquires the IOTP level trading parameters (Consumer's shopping identifier, payment direction, initial currency amounts, discount rates, Merchant's and Delivery Handler's Net Locations, Non-Payment Handler's Organisational Data, initial order information, ...).

3. The registered IOTP Payment Bridges are inquired by the IOTP Application Core about the accepted payment brands ("Find Accepted Payment Brand"). Their responses provide most of the attribute values for the compilation of the Brand List Component's Brand Elements. The IOTP Application Core might optionally match the returned payment brands with Merchant's general preferences.

The IOTP Application Core must provide any wallet identifiers, if they are required by the IOTP Payment Bridges which signal their need by specific error codes (see below). Any signaled error that could not immediately solved by the IOTP Application Core should be logged - this applies also to the subsequent API calls of this section. In this case, the IOTP Application Core creates an IOTP Error Block (hard error), transmits it to the Consumer, and terminates the current transaction.

4. The IOTP Application Core interrogates the IOTP Payment Bridges for each accepted payment brand about the supported payment protocols ("Find Accepted Payment Protocol"). These responses provide the remaining attribute values of the Brand Elements as well as all attribute values for the compilation of the Brand List Component's Protocol Amount and Pay Protocol Elements. Furthermore, the organisational data about the Payment Handler is returned. The IOTP Application Core might optionally match the returned payment brands with Merchant's general preferences.

Alternatively, the IOTP Application Core might skip the calls of "Find Accepted Payment Brands" (cf. Step 3) and issue the "Find Accepted Payment Protocol" call without any Brand given on the input parameter list. In this case, the IOTP Payment Bridge responds on the latter call with the whole set of payment schemes supported w.r.t.

the other input parameters.

5. The steps 3 and 4 are repeated during IOTP Value Exchange transactions - these steps are omitted in the previous figure.

6. The IOTP Application Core compiles the Brand List Component(s) and the IOTP Trading Protocol Options Block. It is recommended that "equal" items returned by IOTP Payment Bridge function calls are shared due to the extensive linking capabilities within the Brand List Component. However, the compilation must consider several aspects in order to prevent conflicts - sharing detection might be textual matching (after normalization):

- o Packaged Content Elements contained in the Brand List Component (and subsequently generated Payment and Order Components) might be payment scheme specific and might depend on each other.
- o Currently, IOTP lacks precise rules for the content of the Packaged Content Element. Therefore, transaction / brand / protocol / currency amount (in)dependent data might share the same Packaged Content Element or might spread across multiple Packaged Content Elements.
- o The Consumer's IOTP Application Core transparently passes the Packaged Content Elements to the IOTP Payment Bridges which might not be able to handle payment scheme data of other payment schemes, accurately.

The rules and mechanisms of how this could be accomplished are out of the scope of this document. Furthermore, this document does not define any further restriction to the IOTP specification.

7. The IOTP Application Core determines whether the IOTP message can be enriched with an Offer Response Block. This is valid under the following conditions:

- o All payment alternatives share the attribute values and Packaged Content Elements of the subsequently generated IOTP Payment and Order Components.
- o The subsequently generated data does not depend on any IOTP BrandSelInfo Elements that might be reported by the consumer within the TPO Selection Block in the brand dependent variant.

If both conditions are fulfilled, the IOTP Application Core might request the remaining payment scheme specific payment initialization data from the IOTP Payment Bridge ("Get Payment Initialization Data") and compile the IOTP Offer Response Block.

Optionally, the IOTP Application Core might request the current process state from the IOTP Payment Bridge and add the inferred order

status to the IOTP Offer Response Block. Alternatively, IOTP

Application might determine the order status on its own.

As in step 6, the rules and mechanisms of how this could be accomplished are out of the scope of this document.

8. The IOTP Application Core compiles the IOTP TPO Message including all compiled IOTP Blocks and transmits the message to the Consumer. The IOTP Application Core terminates if an IOTP Offer Response Block has been created.

9. The Consumer performs the Brand Selection Steps (cf. Section 2.3) and responds with a TPO Selection Block if no IOTP Offer Response Block has been received. Otherwise, the following step is skipped.

10. On brand dependent transactions, the IOTP Application Core requests the remaining payment scheme specific payment initialization data from the IOTP Payment Bridge ("Get Payment Initialization Data"), compiles the IOTP Offer Response Block, transmits it to the Consumer, and terminates. Like Step 7, the IOTP Application Core might access the current process state of the IOTP Payment Bridge for the compilation of the order status.

Any error during this process raises an IOTP Error Block.

2.3 Brand Selection

This section describes the steps that happen mainly after the Merchant's Brand Compilation (in a brand independent transaction). However, these steps might partially interlace the previous process (in a brand dependent transaction).

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Merchant      Merchant generates Brand List(s) containing
                Brands, Payment Protocols and Currency Amounts
              On brand independent transactions
              |  Merchant generates Offer Response Block
Consumer      Compile set(s) of Brands B/Protocols P
              for each set
              |  Find Payment Instrument(B, P, C)        -> IPB
              |  Find Payment Instrument Response (PI*)    <- IPB
              Consumer selects Brand/Currency/Payment Instrument
                from those that will work and generates Brand
                Selection Component
              For the Selection
              |  Get Payment Initialization Data(B,C,PI,P) -> IPB
              |  Get Payment Initialization Data Response()<- IPB
              On brand dependent transaction
```

```
                |  Generate and transmit TPO Selection Block
```

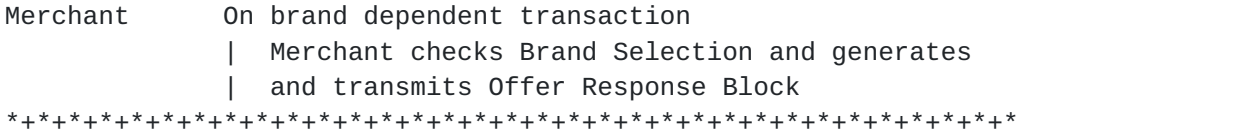


Figure 4 Brand Selection Message Flows

1. The Merchant's commerce server controls the shopping dialog with its own mechanisms until the Consumer checks out the shopping cart and indicates his payment intention. The subsequent processing switches to the IOTP based form by the activation of the Merchant's IOTP aware application.

2. The IOTP Application Core compiles the IOTP Trading Protocol Options Block which contains the IOTP Brand List Component(s) enumerating Merchant's accepted payment brands and payment protocols and initiates the Brand Selection process.

3. This first IOTP message activates the Consumer's IOTP aware application, e.g., the Web browser invokes a helper application (e.g., Java applet or external application). Its IOTP Application Core

o infers the accepted payment brands, payment protocols, payment direction, currencies, payment amounts, any descriptions etc., and their relationships from the IOTP message,

o determines the registered IOTP Payment Bridges,

o compiles one or multiple set of brand and protocol such that the join of all set describes exactly the set of payment alternatives being offered by the Merchant.

o inquires payment (protocol) support and the known payment instruments from each registered IOTP Payment Bridge for each compiled set ("Find Payment Instrument"). However, some IOTP Payment Bridges may refuse payment instrument distinction.

The payment protocol support may differ between payment instruments if the IOTP Payment Bridge supports payment instrument distinction.

These API calls are used to infer the payment alternatives at the startup of any payment transaction (without user unfriendly explicit user interaction).

The IOTP Application Core must provide wallet identifiers, if they are requested by the IOTP Payment Bridges which signal their need by specific error codes (see below).

It is recommended that the IOTP Application Core manages wallet

identifiers. But for security reasons, it should store pass phrases

in plain text only in runtime memory. Developers of IOTP Payment Bridges and payment software modules should provide a thin and fast implementation - without lengthy initialization processes- for this initial inquiry step.

4. The IOTP Application Core

verifies the Consumer's payment capabilities with the Merchant's accepted payment brands and currencies,

o displays the valid payment instruments and payment instrument independent payment brands (brand and protocol) together with their purchase parameters (payment direction, currency, amount), and

o requests the Consumer's choice or derives it automatically from any configured preferences. Any selection ties one IOTP Payment Bridge with the following payment transaction.

The handling and resolution of unavailable IOTP Payment Bridges during the inquiry in Step 3 is up to the IOTP Application Core. It may skip these IOTP Payment Bridges or may allow user supported resolution.

Furthermore, it may offer the registration of new payment instruments when the Consumer is requested for payment instrument selection.

5. The IOTP Application Core interrogates the fixed IOTP Payment Bridge whether the payment might complete with success ("Check Payment Possibility"). At this step, the IOTP Payment Bridge may issue several signals, e.g.,

- o payment can proceed immediately,
- o required peripheral inclusive some required physical payment instrument (chip card) is unavailable,
- o (non-IOTP) remote party (e.g. issuer, server wallet) is not available,
- o wallet identifier or pass phrase is required,
- o expired payment instrument (or certificate), insufficient funds, or
- o physical payment instrument unreadable.

In any erroneous case, the user should be notified and offered accurate alternatives. Most probably, the user might be recommended

- o to resolve the problem, e.g. to insert another payment instrument or to verify the periphery,
- o to proceed (assuming its success),
- o to cancel the whole transaction, or

o to suspend the transaction, e.g., initiating a nested

transaction for uploading an electronic purse.

If the payment software implements payment instrument selection on its own, it may request the Consumer's choice at this step.

If the check succeeds, it returns several IOTP Brand Selection Info Elements.

6. The Steps 2 to 5 are repeated and possibly interlaced for the selection of the second payment instrument during IOTP Value Exchange transactions - this is omitted in the figure above.

7. The IOTP Brand Selection Component is generated and enriched with the Brand Selection Info elements. This component is transmitted to the Merchant inside a TPO Selection Block if the received IOTP message lacks the IOTP Offer Response Block. The Merchant will then respond with an IOTP Offer Response Block (following the aforementioned compilation rules).

2.4 Successful Payment

An example of how the functions in this document are used together to effect a successful payment is illustrated in the Figure 5.

Technically, two payments happen during IOTP Value Exchange transactions.

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Consumer        Start Payment Consumer(Amount,[PS0]...)    -> IPB
                Start Payment Cons. Res.([PS1], CS=Cont.)  <- IPB
                Create and transmit Payment Request Block
Payment Handler Start Payment Pay. Handler(Amount, [PS1])  -> IPB
                Start Payment PH Response(PS2, CS=Cont.)   <- IPB
                Create and transmit Payment Exchange Block
Consumer        Continue Process(PS2)                      -> IPB
                Continue Process Response(PS3, CS=Cont.)   <- IPB

         ... CONTINUE SWAPPING PAYMENT EXCHANGES UNTIL ...

Payment Handler Continue Process Response([PSn], CS=End)   <- IPB
                Request any local payment receipt
                |  Inquire Process State()                 -> IPB
                |  Inquire Proc. State Resp.(State, [Rcp.])<- IPB
                Create and transmit Payment Response Block
                Terminate transaction, actively
                |  Change Process State(State)             -> IPB
```

```
                  |  Change PS Response(State=CompletedOK)   <- IPB
```

```
Consumer        On receipt of final payment scheme data
                |  Continue Process(PSn)                    -> IPB
                |  Continue Process Response(CS=End)        <- IPB
                Check Payment Receipt(Receipt)              -> IPB
                Check Payment Receipt Response()            <- IPB
                Request any local payment receipt
                |  Inquire Process State()                  -> IPB
                |  Inquire Proc. State Resp.(State, [Rcp.])<- IPB
                Terminate transaction, actively
                |  Change Process State(State)              -> IPB
                |  Change PS Response(State=CompletedOk)    <- IPB
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
```

Figure 5 Example Payment Message Flows

1. After Brand Selection and receipt of the IOTP Offer Response Block, the Consumer switches from communicating with the Merchant to communicating with the Payment Handler.

This might be a milestone requiring the renewed Consumer's agreement about the payment transaction's continuation. Particularly, this is a good moment for payment suspension (and even cancellation), which will be most probably supported by all payment schemes. Simply, because the genuine payment legacy systems have not been involved in the current transaction.

Such an agreement might be explicit per transaction or automatic based on configured preferences, e.g., early acknowledgements for specific payment limits.

It is assumed, that the transaction proceeds with minimal user (Consumer and Payment Handler) interaction and that its progess is controlled by the IOTP Application Core and IOTP Payment Bridge.

2. In order to open the actual payment transaction, the IOTP Application Core issues the "Start Payment Consumer" request towards the IOTP Payment Bridge. This request carries the whole initialization data of the payment transaction being referred to by the IOTP Payment Bridge for subsequent consistency checks:

- o payment brand and its description from the selected Brand Element of the IOTP Brand List Component,
- o payment instrument from preceding inquiry step,
- o further payment parameters (currency, amount, direction, expiration) from the selected Currency Amount element, Brand List Component, and Payment Component of the IOTP Offer Response Block,
- o payment protocol from the selected IOTP Pay Protocol Element,
- o order details contained in the IOTP Order Component which might be payment scheme specific,

o payment scheme specific data inclusive payment protocol descriptions from the IOTP Protocol Amount Element, and IOTP Pay

Protocol Element, and
o payment scheme specific data inclusive payment protocol descriptions, in which the name attribute includes the prefix as "Payment:" from the Trading Role Data Component.

Generally, the called API function redoes most checks of the "Check Payment Possibility" call due to lack of strong dependencies between both requests: There might be a significant delay between both API requests.

The called API function may return further payment scheme specific data being considered as payment specific initialization data for the Payment Handler's IOTP Payment Bridge.

If the fixed Existing Payment Software implements payment instrument selection on its own, it may request the Consumer's choice at this step.

The IOTP Payment Bridge reports lack of capability quite similar to the "Check Payment Possibility" request to the IOTP Application Core. The Consumer may decide to resolve the problem, to suspend, or to cancel the transaction, but this function call must succeed in order to proceed with the transaction.

Developers of payment modules may decide to omit payment instrument related checks like expiration date or refunds sufficiency, if they are part of the specific payment protocol.

If the IOTP Payment Bridge requests wallet identifiers or pass phrases anywhere during the payment process, they should be requested by this API function, too. It is recommended that the IOTP Application Core stores plain text pass phrases only in runtime memory.

Finally, the IOTP Application Core generates the IOTP Payment Request Block, inserts any returned payment scheme data, and submits it to the Payment Handler's system.

3. The Payment Handler's IOTP Application Core opens the payment transaction calling the "Start Payment Payment Handler" API function. The payment brand, its description, payment protocol, payment specific data, payment direction, currency and payment amount are determined quite similar to the Consumer's IOTP Application Core. Furthermore, the content of the IOTP Payment Scheme Component and the IOTP Brand Selection Info Elements are passed to this function.

On success, the Payment Handler's IOTP Payment Bridge responds with payment scheme specific data. On failures, this non- interactive

server application has to resolve any problems on its own or to give

up aborting the payment transaction. However, the Consumer may restart the whole payment transaction. Anyway, the payment log file should reflect any trials of payments.

Eventually, the Payment Handler informs the Consumer about the current IOTP Process State using the IOTP Payment Response or IOTP Error Block.

Note that the "Start Payment Payment Handler" call might return the Continuation Status "End" such that payment processing proceeds with Step 7.

4. The IOTP Application Core verifies the presence of the Payment Exchange Block in the IOTP message and passes the contained payment scheme specific data to the fixed IOTP Payment Bridge ("Continue Process") which returns the next IOTP Payment Scheme Component.

This Payment Scheme Component is encapsulated in an IOTP Payment Exchange Block and transmitted to the Payment Handler.

5. The Payment Handler's IOTP Application Core verifies the presence of the Payment Exchange Block and passes the contained payment scheme specific data to the fixed IOTP Payment Bridge ("Continue Process") which returns the next IOTP Payment Scheme Component for encapsulation and transmission to the Consumer.

6. The payment process continues with IOTP Payment Exchange Block exchanges, carrying the payment scheme specific data. Each party (1) submits the embedded payment scheme specific data transparently to the appropriate IOTP Payment Bridge calling the "Continue Process" API function, (2) wraps the returned payment scheme specific data into an IOTP Payment Exchange Block, and (3) transmits this block to the counter party.

However, the processing of the payment scheme specific data may fail for several reasons signaled by specific error codes which are transformed to IOTP Payment Response Blocks (generated by Payment Handler) or IOTP Error Blocks (both parties may generate them) and transmitted to the counter party.

7. Eventually, the Payment Handler's IOTP Payment Bridge recognizes the termination of the payment transaction and reports this by the continuation status "End" on the output parameter of "Continue Process" (or "Start Payment Payment Handler"). Then, the IOTP Application Core issues the "Inquire Process State" API call and verifies whether an IOTP Payment Receipt Component has been returned. The IOTP Application Core wraps the payment receipt, the status response, and the optional payment scheme specific data in an IOTP Payment Response Block and transmits this block to the Consumer.

However, any of these API calls may fail or any response might be incomplete (e.g., lack of payment receipt). Then, the Consumer has to be notified about the failed processing by an IOTP Error Block.

Finally, the Payment Handler terminates the payment transaction with the "Change Process State" API call without awaiting any further response from the Consumer. Further failures are not reported to the Consumer.

Note that it might be possible that the Consumer's IOTP Payment Bridge has returned the previous payment scheme specific data with the continuation status "End". Even in the absence of this knowledge - this status is not exchanged between the Consumer and the Payment Handler - the Payment Handler must not supply any further payment scheme specific data. Such data will be rejected by the Consumer's IOTP Payment Bridge.

8. The Consumer passes the optional payment scheme specific data and the payment receipt to the fixed IOTP Payment Bridge by "Continue Process" and "Check Payment Receipt" API calls.

Afterwards, the IOTP Application Core issues the "Inquire Process State" API call and verifies whether extensions to the payment receipt have been returned.

Finally, the transaction is terminated by calling the "Change Process State" API function which verifies and synchronizes the reported payment status with the local one and signals any inconsistencies. Any Inconsistency and returned status text should be displayed to the Consumer.

At this point, the payment transaction has already been closed by the Payment Handler. Therefore, any failure has to be resolved locally or out-of-band.

2.5 Payment Inquiry

In Baseline IOTP, Payment inquiries are initiated by the Consumer in order to verify the current payment progress and process state at the remote Payment Handler.

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Consumer        Start Payment Inquiry()                    -> IPB
                Start Payment Inquiry Response([PS1])      <- IPB
                Create and transmit Inquiry Request Trading Block
Payment Handler Inquire Payment Status([PS1])              -> IPB
                Inquire Payment Status Res.(State, [PS2])  -> IPB
```

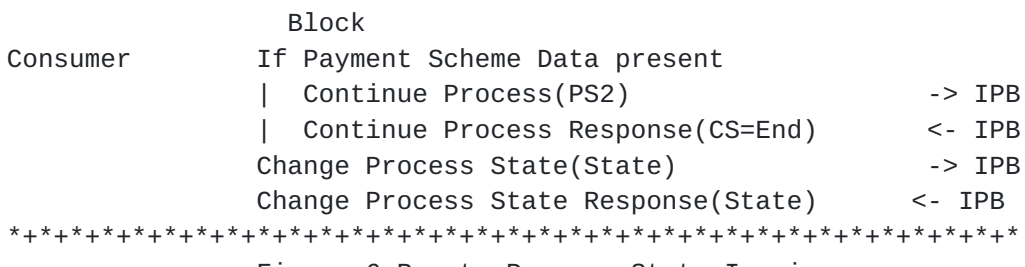


Figure 6 Remote Process State Inquiry

1. The Consumer might initiate a payment inquiry once the payment transaction has been opened by the IOTP Application Core, i.e., at any time after the initial submission of the IOTP Payment Request Block. The IOTP Application Core requests any additional specific payment scheme data from the IOTP Payment Bridge which has been fixed during brand selection (cf. Section 2.3) using the "Start Payment Inquiry" API request.

Erroneous API responses should be reported to the Consumer and valid alternatives (typically retry and cancellation) should be presented by the IOTP Application Core.

This request might perform the complete initialization, e.g. availability check of periphery or pass phrase supplement, and the IOTP Payment Bridge reports lack of capability quite similar to the "Check Payment Possibility" request to the IOTP Application Core.

If the IOTP Payment Bridge requests wallet identifiers or pass phrases anywhere during the payment process, they should be requested by this API function, too. It is recommended that the IOTP Application Core stores plain text pass phrases only in runtime memory.

The IOTP Application Core encapsulates any Payment Scheme Component in an IOTP Inquiry Request Block and submits the block to the Payment Handler.

2. The Payment Handler analyses the IOTP Inquire Request Block, maps the Transaction Identifier to payment related attributes (brand, consumer and payment identifiers), determines the appropriate IOTP Payment Bridge, and forwards the request to the this IOTP Payment Bridge ("Inquire Payment Status"). The IOTP Application Core transforms the response to an IOTP Inquiry Response Block and transmits it to the Consumer.

3. On receipt of the respective IOTP Inquiry Response Block the Consumer's IOTP Application Core submits any encapsulated payment scheme specific data to the IOTP Payment Bridge for verification ("Continue Process").

4. The IOTP Application Core passes the reported payment status

(except textual descriptions) to the IOTP Payment Bridge ("Change Process State") for verification purposes and payment status change. The IOTP Payment Bridge reports any inconsistencies as well as the final payment status to the IOTP Application Core.

Any additional information that might be of interest for the Consumer has to be displayed by the IOTP Payment Bridge or Existing Payment Software on their own.

2.6 Abnormal Transaction Processing

2.6.1 Failures and Cancellations

The IOTP specification distinguishes between several classes of failures:

- Business and technical errors
- Error depth on transport, message and block level
- Transient errors, warnings, and hard errors.

Any IOTP Payment API has to deal with the receipt of failure notifications by and failure responses. This proposal has borrowed the basic mechanisms for error reporting between the IOTP Application Core and the IOTP Payment Bridge from the genuine protocol: Business errors are reported by Status Components within IOTP Response Blocks while technical errors are signaled by Error Components within IOTP Error Blocks.

Cancellations are mimicked as specific business errors which might be initiated by each trading party.

Preferring slim interfaces, this IOTP Payment API introduces one additional Error Code value for business error indication - errors can be raised on every API call. On receipt of this value, the IOTP Application Core has to infer further details by the issuance of the API function call "Inquire Process State".

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Any Party       Issue some API request                    -> IPB
                Error Response(Error Code)                <- IPB
                On "Business Error" response
                |  Inquire Process State()                -> IPB
                |  Inquire P.S. Resp.(State, Receipt)     <- IPB
                Analyze local process state and try to resolve
                   with optional user interaction
```

If Process State Change needed

```
                |  Change Process State (State)            -> IPB
                |  Change Process State Response(State)   <- IPB
                If counter party's notification required
                |  Create Error or Cancel Block (, add to next
                |  message, ) and transmit it to counter party
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
                  Figure 7 Error Response from IPB
```

The specific Completion Codes "ConsCancelled", "MerchCancelled", and "PaymCancelled" - returned by "Inquire Process State" - determine that the IOTP Cancel Block has to be created instead of an IOTP Error Block.

The rules for determining the required behavior of the IOTP Application Core are given in the IOTP specification.

Note that any payment (intermediate) termination, i.e., failures, cancellations, and even success's is always reported to the IOTP Payment Bridge by the API function "Change Process State". This API function does both status changes and consistency checking / synchronization. Any suspicion of inconsistency should be reported by the IOTP Payment Bridge for display by the IOTP Application Core.

```
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
Any Party       Error Block or Cancel Block Received
                If Change Process State required
                |  Change Process State (State)            -> IPB
                |  Change Process State Response(State)   <- IPB
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
           Figure 8 Error Notification from counter party
```

Not every failure might be visible at the IOTP layer, e.g., the processing of payment transactions might temporarily be hampered by intermediate failures at the payment scheme or protocol transport layer which might be resolved by the genuine components.

However, final failures or cancellations have to be reported at the IOTP layer. E.g., communication time-outs and heavily faulty communication channels may disable the transaction.

Any system component may implement time-out recognition and use the aforementioned API mechanisms for the notification of process state changes. But, time-outs may happens while communicating with both the counter party and local system components, like chip card readers or IOTP Payment Bridges. Anyway, the Consumer's IOTP Application Core should notify the Consumer about the resolution alternatives, i.e., retry, suspension, and cancellation.

2.6.2 Resumption

Payment transaction resumption may apply at different steps of a payment transaction:

- The Consumer's and Payment Handler's view of the transaction might not be synchronized: Due to different time-out values the payment transaction may not have been suspended by the counter party.

  Any "Resume Payment ..." API function responds with an Error Code on non suspended payment transaction that signals a business error. Afterwards the IOTP Application Core has to issue the "Inquire Process State" API call for further analysis of the process state.

- One IOTP message sent by one party might not be processed successfully or even received by the counter party.

  This needs to be handled by the genuine payment scheme. It is expected that the IOTP Application Core will not recognize anything.

- IOTP does not provide any specific signal for payment resumption. On receipt of every IOTP Payment Exchange Block, the IOTP Application Core has to decide whether this Block belongs to a pending transaction or to a suspended transaction that should be resumed. The IOTP Application Core might call the "Inquire Process State" API function to update any lack of knowledge.

  Any "Resume Payment" API function responds with an Error Code on non suspended payment transaction that signals a business error. Similar, the "Continue Process" API function should report business errors on non pending payment transactions.

- The payment transaction may not have been created at the Payment Handler (early suspension and failed data transmission). In that case, the IOTP Application Core should respond with a business error that signals the repetition of the payment transaction (by the Consumer).

  Any "Resume Payment", "Continue Process" or "Inquire Process State" API function should return with an Error Code "AttValIllegal" on non existent payment transaction whereby the further Error Attribute "Names" denote the payment identifier.

- The IOTP Application Core should always request fresh payment scheme specific data on resumption - for synchronization purposes

with the Existing Payment Software. Old data in the cache that has

not been send to the counter party should not be accessed.

If the Consumer does not reconnect within an acceptable amount of time, the Payment Handler's system may perform local failure resolution in order to close the transaction and to retain resources for other transactions ("Change Process State"). If the Consumer reconnect afterwards, an IOTP Payment Response or IOTP Error Block could be generated.

2.7 IOTP Wallet Initialization

At startup or on explicit user request the IOTP Application Core should check its IOTP Payment Bridges' internal status by searching for pending payment transactions.

1. The IOTP Application Core interrogates the registered IOTP Payment Bridges about pending payment transactions. The IOTP Application Core may store indicators for pending transactions and use them for driving any subsequent inquiry ("Inquire Pending Payment").

2. If one or more IOTP Payment Bridges report the presence of pending transactions, the IOTP Application Core may try to suspend ("Change Process State") or resume (only Consumer: "Resume Payment Consumer") the pending transactions (on user request).

The IOTP Payment Bridge may deny the processing of any new payment transactions until the pending transactions have been processed. Such denials are signaled by the error code "Business Error".

2.8 Payment Software Management

The IOTP Application Core provides only a simple and generic interface for the registration of new payment methods / instruments ("Manage Payment Software"). It receives the initial user request and defers the actual registration to the corresponding IOTP Payment Bridge.

The IOTP Application Core may also activate the Existing Payment Software for further payment instrument and wallet administration.

3. Mutuality

The Payment API is formalized using the Extensible Markup Language (XML). It defines wrapper elements for both the input parameters and the API function's response. In particular, the response wrapper provides common locations for Error Codes and Error Descriptions.

It is anticipated that this description reflects the logical structure of the API parameter and might be used to derive implementation language specific API definitions.

XML definition:

```
<!ELEMENT IotpPaymentApiRequest (
  FindAcceptedPaymentBrand |
  FindAcceptedPaymentProtocol |
  GetPaymentInitializationData |
  FindPaymentInstrument |
  CheckPaymentPossiblity |
  StartPaymentConsumer |
  StartPaymentPaymentHandler |
  ResumePaymentConsumer |
  ResumePaymentPaymentHandler |
  ContinueProcess |
  InquireProcessState |
  ChangeProcessState |
  InquireAuthChallenge |
  Authenticate |
  CheckAuthResponse |
  CheckPaymentReceipt |
  ExpandPaymentReceipt |
  RemovePaymentLog |
  PaymentInstrumentInquiry |
  InquirePendingPayment |
  ManagePaymentSoftware |
  StartPaymentInquiry |
  InquirePaymentStatus |
  CallBack )>

<!ATTLIST IotpPaymentApi
  xml:lang          NMTOKEN   #IMPLIED
  ContentSoftwareID CDATA     #IMPLIED
  xmlns             CDATA     #FIXED
                "http://www.iotp.org/2000/08/PaymentAPI" >

<!ELEMENT IotpPaymentApiResponse (ErrorResponse?, (
  FindAcceptedPaymentBrandResponse |
  FindAcceptedPaymentProtocolResponse |
```

```
     GetPaymentInitializationDataResponse |
     FindPaymentInstrumentResponse |
```

```
    CheckPaymentPossiblityResponse |
    StartPaymentConsumerResponse |
    StartPaymentPaymentHandlerResponse |
    ResumePaymentConsumerResponse |
    ResumePaymentPaymentHandlerResponse |
    ContinueProcessResponse |
    InquireProcessStateResponse |
    ChangeProcessStateResponse |
    InquireAuthChallengeResponse |
    AuthenticateResponse |
    CheckAuthResponseResponse |
    CheckPaymentReceiptResponse |
    ExpandPaymentReceiptResponse |
    RemovePaymentLogResponse |
    PaymentInstrumentInquiryResponse |
    InquirePendingPaymentResponse |
    ManagePaymentSoftwareResponse |
    StartPaymentInquiryResponse |
    InquirePaymentStatusResponse |
    CallBackResponse )?)>

  <!ATTLIST IotpPaymentApiResponse
    xml:lang          NMTOKEN #IMPLIED
    ContentSoftwareID CDATA   #IMPLIED
    xmlns             CDATA   #FIXED
               "http://www.iotp.org/2000/08/PaymentAPI" >


  <!ELEMENT ErrorResponse (ErrorLocation+,PaySchemePackagedContent*) >
  <!ATTLIST ErrorResponse
    xml:lang      NMTOKEN   #IMPLIED
    ErrorCode     NMTOKEN   #REQUIRED
    ErrorDesc     CDATA     #REQUIRED
    Severity(Warning |
      TransientError |
             HardError)     #REQUIRED
    MinRetrySecs  CDATA     #IMPLIED
    SwVendorErrorRef CDATA  #IMPLIED >
```

Most of the attribute items are intended for immediate insertion in the IOTP Error Block. The attribute values of the Error Location elements attribute have to enriched and transformed into Error Location Elements of the Error Component (cf. IOTP Specification).

Attributes (cf. IOTP Specification):

xml:lang Defines the language used by attributes or child elements within this component, unless

overridden by an xml:lang attribute on a child element.

ContentSoftwareId	Contains information which identifies the ssoftware that generated the content of the element. Its purpose is to help resolve interoperability problems that might occur as a result of incompatibilities between messages produced by different software. It is a single text string in the language defined by "xml:lang". It must contain, as a minimum problems that might occur as a result of o the name of the software manufacturer, o the name of the software, o the version of the software, and o the build of the software.
ErrorCode	Contains an error code which indicates the nature of the error in the message in error. Valid values for the Error Code are given in the following section. This mnemonic enables the automatic failure resolution of the IOTP Application Core which analyzes the error code value in order to determine the continuation alternatives.
ErrorDesc	Contains a description of the error in the language defined by xml:lang. The content of this attribute is defined by the vendor/developer of the software that generated the Error Response Element. It is intended for user display and provides detailed explanations about the failure and its (out-of-band) resolution alternatives.
Severity	Indicates the severity of the error. Valid values are: o Warning. This indicates that although there is a message in error the IOTP Transaction can still continue. o TransientError. This indicates that the error in the message in error may be recovered if the message in error that is referred to by the "Names" attribute is resent. o HardError. This indicates that there is an unrecoverable error in the message in error

and the IOTP Transaction must stop.

MinRetrySecs	This attribute should be present if "Severity" is set to "TransientError". It is the minimum number of whole seconds which the IOTP aware application which received the message reporting the error should wait before re-sending the message in error identified by the "ErrorLocation" attribute. If Severity is not set to "TransientError" then the value of this attribute is ignored.
SwVendorErrorRef	This attribute is a reference whose value is set by the vendor/developer of the software that generated the Error Element. It should contain data that enables the vendor to identify the precise location in their software and the set of circumstances that caused the software to generate a message reporting the error.
Content:	
ErrorLocation	This identifies, where possible, the element and attribute in the message in error that caused the Error Element to be generated. If the "Severity" of the error is not "TransientError", more that one "ErrorLocation" may be specified as appropriate depending on the nature of the error and at the discretion of the vendor/developer of the IOTP Payment Bridge. Its definition coincides with the IOTP specification whereby the attributes "IotpMsgRef", "BlkRef" and "CompRef" are left blank, intentionally.
PaySchemePackagedContent	cf. Table 5

3.1 Error Codes

The following table lists the valid values for the ErrorCode attribute of the Error Response Element. The first sentence of the

error description contains the default text that can be used to

describe the error when displayed or otherwise reported. Individual implementations may translate this into alternative languages at their discretion. However, not every error code may apply to every API call. An Error Code must not be more than 14 characters long.

The Error Codes have been taken from the IOTP Specification and extended by some additional codes which are highlighted by a preceding asterisk.

Generally, if the corrupt values have been user supplied, the IOTP Application Core might prompt for their correction. If the renewal fails or if the IOTP Application Core skips any renewals and some notification has to be send to the counter-party, the error code is encapsulated within an IOTP Error Block.

However, the IOTP server application reports business errors - visible at the IOTP layer - in the Status Component of the respective Response Block.

The IOTP Application Core may add the attributes (and values) within the ErrorLocation elements being omitted by the IOTP Payment Bridge.

The following table mentions any modification from this general processing for particular error values. Furthermore, it contains hints for developers of IOTP Application Core software components about the processing of error codes. Conversely, developers of IOTP Payment Bridges get impressions about the expected behavior of the IOTP Application Core.

The IOTP Payment API assumes that the IOTP Application Core implements the dialog boxes needed for error resolution. But it does not assume, that the IOTP Payment Bridge actually relies on them. Instead, the IOTP Payment Bridge may try resolution on its own, may implement specific dialog boxes, and may signal only final failures.

Note: This abstract document assumes that the API parameters are exchanged XML encoded. Therefore, several error values might disappear in lower level language specific derivations.

Error Value	Error Description
Reserved	Reserved. This error is reserved by the vendor/developer of the software. Contact the vendor/developer of the software for more information (see the SoftwareId attribute of the Message Id element in the Transaction Reference Block [IOTP]).
XmlNotWellFrmd	XML not well formed. The XML document is not

well formed. See [XML] for the meaning of

	"well formed".
XmlNotValid	XML not valid. The XML document is well formed but the document is not valid. See [XML] for the meaning of "valid". Specifically: o the XML document does not comply with the constraints defined in the IOTP document type declaration, and o the XML document does not comply with the constraints defined in the document type declaration of any additional [XML-NS] that are declared. The Names attribute might refer some attributes and elements of the input parameter list.
(*)ElNotValid	Element not valid. Invalid element in terms of prescribed syntactical characteristics. The ElementRef attributes of ErrorLocation elements might refer to the corresponding elements (if they have ID attributes). The IOTP Application Core has to replace the error code with "XmlNotValid" before transmission to the counterparty.
ElUnexpected	Unexpected element. Although the XML document is well formed and valid, an element is present that is not expected in the particular context according to the rules and constraints contained in this specification. The ElementRef attributes of ErrorLocation elements might refer to the corresponding elements (if they have ID attributes).
ElNotSupp	Element not supported. Although the document is well formed and valid, an element is present that o is consistent with the rules and constraints contained in this specification, but o is not supported by the IOTP Aware

Application which is processing the IOTP

	Message. The ElementRef attributes of ErrorLocation elements might refer to the corresponding elements (if they have ID attributes).
ElMissing	Element missing. Although the document is well formed and valid, an element is missing that should have been present if the rules and constraints contained in this specification are followed. The ElementRef attributes of ErrorLocation elements might refer to the corresponding elements (if they have ID attributes).
ElContIllegal	Element content illegal. Although the document is well formed and valid, the element contains values which do not conform the rules and constraints contained in this specification. The ElementRef attributes of ErrorLocation elements might refer to the corresponding element (if they have ID attributes). The IOTP Application Core has to replace the Error Code with "ElNotSupp" before transmission to the counter party, if the ErrorLocation elements refer to non PackagedContent element.
EncapProtErr	Encapsulated protocol error. Although the document is well formed and valid, the Packaged Content of an element contains data from an encapsulated protocol which contains errors. The ElementRef attributes of ErrorLocation elements might refer to the corresponding element (if they have ID attributes).
AttUnexpected	Unexpected attribute. Although the XML document is well formed and valid, the presence of the attribute is not expected in the particular context according to the rules and constraints contained in this specification.

The AttName attributes of ErrorLocation

elements might refer to the corresponding attribute tags.

(*)AttNotValid — Attribute not valid. Invalid attribute value in terms of prescribed syntactical characteristics.

The AttName attributes of ErrorLocation elements might refer to the corresponding attribute tags.

The IOTP Application Core has to replace the error code with "XmlNotValid" before transmission to the counter party.

AttNotSupp — Attribute not supported. Although the XML document is well formed and valid, and the presence of the attribute in an element is consistent with the rules and constraints contained in this specification, it is not supported by the IOTP Aware Application which is processing the IOTP Message.

AttMissing — Attribute missing. Although the document is well formed and valid, an attribute is missing that should have been present if the rules and constraints contained in this specification are followed.

The AttName attributes of ErrorLocation elements might refer to the corresponding attribute tags.

If the attribute is required by the IOTP Document Type Declaration (#REQUIRED) the hints for non valid attributes should be adopted, otherwise these for illegal attribute values.

AttValIllegal — Attribute value illegal. The attribute contains a value which does not conform to the rules and constraints contained in this specification.

The AttName attributes of ErrorLocation elements might refer to the corresponding attribute tags - valid values are:

BrandId: illegal/unknown Brand Identifier -

If the brand is not recognized/known by any

IOTP Payment Bridge, the IOTP Application Core may offer the registration of a new Payment Instrument.

PaymentInstrumentId: illegal/unknown Payment Instrument Identifier - This indicates a serious communication problem if the attribute value has been reported by the same "wallet" on a previous inquiry requests. The IOTP Application Core has to replace the error code with "UnknownError" before transmission to the counter party.

WalletId: illegal/unknown Wallet Identifier - It is assumed that the wallet identifier is checked before the pass phrase. On invalid wallet identifiers, the IOTP Application Core may open the dialog in order to request the correct wallet identifier. In addition, any pass phrase may be supplied by the user. The dialog should indicate the respective payment brand(s). The IOTP Application Core has to replace the error code with "UnknownError" before transmission to the counter party.

Passphrase: illegal/unknown Pass Phrase - The IOTP Application Core may open the dialog in order to request the correct pass phrase. If the pass phrase is wallet identifier specific the dialog should display the wallet identifier. The IOTP Application Core has to replace the error code with "TransportError" before transmission to the counter party.

Action: illegal / unknown / unsupported Action

PropertyTypeList: lists contains illegal / unknown / unsupported Property Types - The IOTP Application Core tries only the local resolution but does never transmit any IOTP Error Block to the counter party.

CurrCode: illegal/unknown/unsupported Currency Code

CurrCodeType: illegal/unknown/unsupported

Currency Code Type

Amount: illegal/unknown/unsupported Payment Amount

PayDirection: illegal/unknown/unsupported Payment Direction

ProtocolId: illegal/unknown/unsupported Protocol Identifier

OkFrom: illegal/unknown/unsupported OkFrom Timestamp

OkTo: illegal/unknown/unsupported OkTo Timestamp

ConsumerPayId: illegal/unknown Consumer Payment Identifier

PaymentHandlerPayId: illegal/unknown Payment Handler Payment Identifier

PayId: illegal/unknown Payment Identifier

AttValNotRecog	Attribute Value Not Recognized. The attribute contains a value which the IOTP Aware Application generating the message reporting the error could not recognize. The AttName attributes of ErrorLocation elements might refer to the corresponding attribute tags
MsgTooLarge	Message too large. The message is too large to be processed by the IOTP Payment Bridge (or IOTP Application Core).
ElTooLarge	Element too large. The element is too large to be processed by the IOTP Payment Bridge (or IOTP Application Core). The ElementRef attributes of ErrorLocation elements might might refer to the corresponding elements.
ValueTooSmall	Value too small or early. The value of all or part of an element content or an attribute, although valid, is too small.

	The ErrorLocation elements might refer to the corresponding attribute tags or elements.
ValueTooLarge	Value too large or in the future. The value of all or part of an element content or an attribute, although valid, is too large. The ErrorLocation elements might refer to the corresponding attribute tags or elements.
ElInconsistent	Element Inconsistent. Although the document is well formed and valid, according to the rules and constraints contained in this specification: o the content of an element is inconsistent with the content of other elements or their attributes, or o the value of an attribute is inconsistent with the value of one or more other attributes. The Error Description may contain further explanations. The ErrorLocation elements might refer to the corresponding attribute tags or elements that are inconsistent.
TransportError	Transport Error. This error code is used to indicate that there is a problem with the transport mechanism that is preventing the message from being received. It is typically associated with a "Transient Error". The connection to some periphery or the counter party could not be established, is erroneous, or has been lost. The Error Description may contain further narrative explanations, e.g., "chip card does not respond", "remote account manager unreachable", "Internet connection to xyz lost", "no Internet connection available", "no modem connected", or "serial port to modem used by another application". This text should be shown to the end user. If

timeout has occurred at the Consumer this

	text should be shown and the Consumer may decide how to proceed - alternatives are retry, payment transaction suspension, and cancellation.
MsgBeingProc	Message Being Processed. This error code is only used with a Severity of Transient Error. It indicates that the previous message, which may be an exchange message or a request message, is being processed and, if no response is received by the time indicated by the "MinRetrySecs" attribute, then the original message should be resent.
SystemBusy	System Busy. This error code is only used with a Severity of Transient Error. It indicates that the IOTP Payment Bridge or Existing Payment Software that received the API request is currently too busy to handle it. If no response is received by the time indicated by the "MinRetrySecs" attribute, then the original message should be resent. The Error Description may provide further explanations, e.g., "wallet / chip card reader is unavailable or locked by another payment transaction", "payment gateway is overloaded", "unknown chip card reader", or "unrecognized chip card inserted, change chip card". The Consumer's IOTP Application Core may visualize the error description and ask the Consumer about the continuation - alternatives are retry, payment transaction suspension, and cancellation.
UnknownError	Unknown Error. Indicates that the transaction cannot complete for some reason that is not covered explicitly by any of the other errors. The Error description attribute should be used to indicate the nature of the problem. The ErrorLocation elements might refer to the corresponding attribute tags or elements that are inconsistent.

(*)SyntaxError	Syntax Error. An (unknown) syntax error has occurred.

	The ErrorLocation elements might refer to the corresponding attribute tags or elements that are inconsistent. The IOTP Application Core has to replace the error code with "XmlNotValid" or "UnknownError" before transmission to the counter party.
(*)ReqRefused	Request refused. The API request is (currently) refused by the IOTP Payment Bridge. The error description may provide further explanations, e.g., "wallet / chip card reader is unavailable or locked by another payment transaction", "payment gateway is overloaded", "unknown chip card reader", or "unrecognized chip card inserted, change chip card". The Consumer's IOTP Application Core may visualize the error description and ask the Consumer about the continuation - alternatives are retry, payment transaction suspension, and cancellation. Denials due to invalid Process States should be signaled by "BusinessError". Typically, this kind of error is not passed to the counter party's IOTP Application Core. Otherwise, it maps to "TransportError" or "UnknownError".
(*)ReqNotSupp	Request not supported. The API function(ality) has not been implemented in the IOTP Payment Bridge. Typically, this kind of error is not passed to the counter party's IOTP Application Core. Otherwise, it maps to "TransportError" or "UnknownError".
(*)BusError	Business Error. The API request has been rejected because some payment transaction has an illegal payment status. Particularly, this error code is used to signal any raise of payment business layered failures. The ErrorLocation elements may refer to payment transactions using the party's Payment Identifier - it defaults to the

current transaction or might contain the
current payment transaction party's Payment

	Identifier - identified by the ElementRef attribute while the AttName attribute is fixed with "PayId".
	The IOTP Application Core must inquire the IOTP Payment Bridge about the actual Process State which actually encodes the business error ("Inquire Process State"). This error code must not be passed to the counter party's IOTP Application Core.

Table 2: Common Error Codes

The IOTP Payment Bridge may also use the error description in order to notify the Consumer about further necessary steps for failure resolution, e.g., "sorry, your payment transaction failed. Unfortunately, you have been charged, please contact your issuer."

3.2 Attributes and Elements

The following table explains the XML attributes in alphabetical order - any parenthesized number behind the attribute tag recommends the maximal length of the attribute value in characters:

Attribute	Description
Amount (11) AmountFrom(11) AmountTo (11)	Indicates the payment amount to be paid in whole and fractional units of the currency. For example $245.35 would be expressed "245.35". Note that values smaller than the smallest denomination are allowed. For example one tenth of a cent would be "0.001".
AuthenticationId	An identifier specified by the authenticator which, if returned by the organization that receives the authentication request, will enable the authenticator to identify which authentication is being referred to.
BrandId (128)	This contains a unique identifier for the brand (or promotional brand). It is used to match against a list of Payment Instruments which the Consumer holds to determine whether or not the Consumer can pay with the

Brand.

	Values of BrandId are managed under procedure being described in the IOTP protocol specification.
BrandLogoNetLocn	The net location which can be used to download the logo for the organization (cf. IOTP Specification). The content of this attribute must conform to [URL].
BrandName	This contains the name of the brand, for example "MasterCard Credit". This is the description of the Brand which is displayed to the consumer in the Consumer's language defined by "xml:lang". For example it might be "American Airlines Advantage Visa". Note that this attribute is not used for matching against the payment instruments held by the Consumer.
BrandNarrative	This optional attribute is used by the Merchant to indicate some special conditions or benefit which would apply if the Consumer selected that brand. For example "5% discount", "free shipping and handling", "free breakage insurance for 1 year", "double air miles apply", etc.
CallBackFunction	A function which is called whenever there is a change of Process State or payment progress, e.g. for display updates. However, the IOTP Payment Bridge may use its own mechanisms and dialog boxes.
CallBackLanguageList	A list of language codes which contain, in order of preference, the languages in which the text passed to the Call Back function will be encoded.
CompletionCode (14)	Indicates how the process completed. It is required if ProcessState is set to "Failed" otherwise it is ignored. Valid values as well as recovery options are given in the IOTP specification. The IOTP Payment Bridge may also use the Status Description to notify the Consumer about further necessary steps in order to

resolve some kind of business failures,

e.g.,

- "sorry, your payment transaction failed. Unfortunately, you have been charged, please contact your issuer."
- "insufficient capacity left (on your card) for refund",
- "payment failed/chip card error/internal error, please contact your payment instrument's issuer"

ConsumerDesc	A narrative description of the Consumer.
ConsumerPayId (14)	An unique identifier specified by the Consumer that, if returned by the Payment Handler in another Payment Scheme Component or by other means, enables the Consumer to identify which payment is being referred to. This unique identifier is generated by the IOTP Application Core and submitted to the IOTP Payment Bridge on every API call. It may equal to the Payment Handler Payment Identifiers but need not necessarily be so. The uniqueness extends to multiple payment instruments, payment brands, payment protocols, wallet identifiers, and even multiple IOTP Payment Bridges.
ContStatus	During payment progress, this status value indicates whether the payment needs to be continued with further IOTP Payment Scheme Component exchanges with the remote party. "End" indicates that the reported payment scheme data is the last data to be exchanged with the counter party.
ContentSoftwareId	This contains information that identifies the software that generated the content of the element. Its purpose is to help resolve interoperability problems that might occur as a result of incompatibilities between messages produced by different software. It is a single text string in the language defined by xml:lang. It must contain, as a minimum: - the name of the software manufacturer,

- the name of the software,

- the version of the software, and
- the build of the software.

CurrCodeType (14) Indicates the domain of the CurrCode. This attribute is included so that the currency code may support nonstandard currencies such as frequent flyer point, trading stamps, etc. Its values may be

- ISO-4217-A, the default, indicates the currency code is the three-letter alphabetic code that conform to ISO 4217

- IOTP indicates that the values of CurrCode are managed under the procedure described in [IOTP].

CurrCode (14) A code which identifies the currency to be used in the payment. The domain of valid currency codes is defined by "CurrCodeType"

MerchantPayId (14) An private identifier specified by the Merchant which will enable the Merchant to identify which payment is being referred to. It is a pure private item and is never sent to any other party. It is provided by the IOTP Payment Bridge on payment preparation during brand compilation.

Cf. To "ConsumerPayId" for note about uniqueness.

MerchantOrgId (64) A local item that might refer to some specific shop in a multi shop environment. This item is optional and might enrich the Wallet Identifier which itself can be used for the same purpose.

Name Distinguishes between multiple occurrences of Packaged Content Elements at the same point in IOTP. For example:

```
<ABCD>
  <PackagedContent Name='FirstPiece'>
    snroasdfnas934k
  </PackagedContent>
  <PackagedContent Name='SecondPiece'>
    dvdsjnl5poidsdsflkjnw45
  </PackagedContent>
```

```
</ABCD>
```

	The "Name" attribute may be omitted, for example if there is only one Packaged Content element.
OkFrom (30) OkTo (30)	The date and time in UTC Format range indicated by the merchant in which the Payment Handler may accept the payment.
Passphrase (32)	Payment wallets may use pass phrase protection for transaction data and payment instruments' data. However, it is assumed that there exists a public and customizable payment instrument identifier such that these identifiers together with their relationship to payment brands, payment protocols, payment directions, and currency amounts can be inquired by the IOTP application without any pass phrase knowledge.
PayDirection	Indicates the direction in which the payment for which a Brand is being selected is to be made. Its values may be: o Debit: The sender of the Payment Request Block (e.g. the Consumer) to which this Brand List relates will make the payment to the Payment Handler, or o Credit: The sender of the Payment Request Block to which this Brand List relates will receive a payment from the Payment Handler.
PayId (14)	This attribute is introduced for API simplification: o The Consumer has to identify PayId and ConsumerPayId. o The Merchant has to identify PayId and MerchantPayId. o The Payment Handler has to identify PayId and Payment Handler Pay Id.
PayInstId	This contains the unique identifier used internally by the IOTP Payment Bridge/Existing Payment Software.

PayInstName	This contains the user-defined name of the payment instrument. There exist no (technical) constraints like uniqueness. The "xml:lang" attribute denotes the language encoding of its value.
PaymentHandlerDesc	A narrative description of the Payment Handler.
PaymentHandlerPayId (14)	An unique identifier specified by the Payment Handler that, if returned by the Consumer in another Payment Scheme Component or by other means, enables the Payment Handler to identify which payment is being referred to. It is required whenever it is known. Cf. To "ConsumerPayId" for note about uniqueness.
PaymentInstrumentId (32)	An identifier for a specific payment instrument, e.g. "credit card", "Mondex card for English Pounds". This identifier is fully customizable. It is assumed, that it does not contain confidential information or even an indication to it. The payment instrument identifier is unique within each payment brand. It is displayed to the Consumer during brand selection.
PayReceiptNameRefs (32)	Optionally contains element references to other elements (containing payment scheme specific data) that together make up the receipt. Note that each payment scheme defines in its supplement the elements that must be referenced The IOTP Application Core should save all the components referenced so that the payment receipt can be reconstructed when required.
PayReqNetLocn	The Net Location indicating where an unsecured Payment Request message should be sent if this protocol choice is used. The content of this attribute must conform to [URL] and depends on the Transport Mechanism.

PercentComplete (3) A number between 0 and 100 which indicates the progress of the payment transaction. The values range between 0 and 99 for pending and suspended transactions.

ProcessState Contains a Process State Code that indicates the current state of the process being carried out. Valid values are:

o NotYetStarted. The Payment Request Block has been received but processing of the Payment Request has not yet started

o InProgress. The payment transaction is pending. The processing of the (Payment) Request Block has started but it is not yet complete.

o (*)Suspended: The payment transaction has been suspended and can be resumed.

This process state is mapped to "InProgress", if it is passed to the counter party's IOTP Application Core.

o CompletedOk. The processing of the (Payment) Request Block and any following Payment Exchange Blocks has completed successfully.

o Failed. The payment processing has finally failed for a Business Error.

o ProcessError. This value is only used when the Status Component is being used in connection with an Inquiry Request Trading Block. It indicates there was a Technical Error in the Request Block which is being processed or some internal processing error. Each party's IOTP Payment Bridge uses this value in order to notify the IOTP Application Core about the presence of technical errors.

PropertyType (14) The property type defines codes used for interrogation of specific properties about a payment instrument. They are unique for each payment brand. The predefined property "all" is used on general inquiries. However, these property types are not used during normal

payment processing. E.g., they may apply to

payment brand specific transactions or out-of-band failure resolution.

PropertyDesc — The property description carries the respective human readable property (value)'s description.

PropertyValue — The actual property value intends automatic post processing.

ProtocolBrandId (64) — This is an identifier to be used with a particular payment protocol. For example, SET and EMV have their own well defined, yet different, values for the Brand identifier to be used with each protocol. The valid values of this attribute are defined in the supplement for the payment protocol identified by "ProtocolId" that describes how the payment protocol works with IOTP. Identifier maps to at most one Protocol Brand Identifier.

ProtocolId (64) — An identifier for a specific payment protocol and version, e.g. "SETv1.0", "ecash". Valid values are defined by supplements to the IOTP specification and they are unique within each payment brand.

ProtocolIds — A sequence of Protocol Identifiers

ProtocolName — A narrative description of the payment protocol and its version in the language identified by "xml:lang". For example "Secure Electronic Transaction Version 1.0". Its purpose is to help provide information on the payment protocol being used if problems arise.

SecPayReqNetLocn — The Net Location indicating where a secured Payment Request message should be sent if this protocol choice is used.

A secured payment involves the use of a secure channel such as [SSL]/[TLS] in order to communicate with the Payment Handler.

The content of this attribute must conform to [URL].

ReceiverOrgId	The Organization Identification which

		receives the payment bridge processing Trading Role Data PackagedContent.
StatusDesc	(256)	An optional textual description of the current process state in the language identified by "xml:lang" that should be displayed to the Consumer. The usage of this attribute is defined in the payment supplement for the payment method being used. Particularly, it provides hints for out-of-band failure resolution. Its length is limited to 256 characters.
StyleSheetNetLocn		This contains the net location to a style sheet with visualisation rules for XML encoded data.
TimeStamp	(30)	The date and time in UTC Format when the payment transaction has been started.
WalletId	(32)	Many existing payment wallet software are multiple wallet capable. The Wallet Identifier selects the actual wallet. It is assumed, that the wallet identifier is a public item, that might be stored by the IOTP Application Core.
xml:lang		Defines the language used by the Process State Description attribute (cf. IOTP Specification)

Table 3: Attributes

The following table explains the XML elements in alphabetical order:

Element	Description
Algorithm	This contains information which describes an Algorithm that may be used to generate the Authentication response. The algorithm that may be used is identified by the Name attribute (cf. IOTP Specification).
AuthReqPackagedContent	The Authentication Request Packaged Content originates from a Authentication

(Data/Response) Component's content

	whereby the outermost element tags are prefixed with "AuthReq". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification). It encapsulates the authentication challenge value. The content of this information is defined in the supplement for a payment protocol.
AuthResPackagedContent	The Authentication Response Packaged Content originates from a Authentication Response Component's content whereby the outermost element tags are prefixed with "AuthRes". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification). It encapsulates the authentication response value. The content of this information is defined in the supplement for a payment protocol.
BrandPackagedContent	Container for further payment brand description. Its content originates from a Brand Element content whose outermost element tags are prefixed with "Brand". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification).
BrandSelBrandInfoPackagedContent	This contains any additional data that may be required by a particular payment brand. It forms the content of the Brand Selection Brand Info Element.
BrandSelProtocolAmountInfoPackagedContent	This contains any additional data that may be required by a particular payment brand in the format. It forms the content of the Brand Selection Protocol Amount Info Element.
BrandSelCurrencyAmountInfoPackagedContent	This contains any additional data that payment brand and currency specific in the format. It forms the content of the Brand Selection Currency Amount Info Element.
MerchantData	Any merchant related data that might be

used by the IOTP Payment Bridge for
different purposes, e.g., it might

	contain access keys to some mall keys. Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification).
PackagedContent	Generic Container for non-IOTP data (cf. IOTP Specification).
PayProtocolPackagedContent	The Pay Protocol Packaged Content originates from a Pay Protocol Element's content whereby the outermost element tags are prefixed with "PayProtocol". It contains information about the protocol which is used by the payment protocol. The content of this information is defined in the supplement for a payment protocol.Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification).
PaySchemePackagedContent	The PayScheme Packaged Content originates from a Payment Scheme Component's content whereby the outermost element tags are prefixed with "PayScheme". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification). It carries the payment specific data. The content of this information is defined in the supplement for a payment protocol.
ProtocolAmountPackagedContent	The Protocol Amount Packaged Content originates from a Protocol Amount Element's content whereby the outermost element tags are prefixed with "Amount". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification). It contains information about the protocol which is used by the payment protocol. The content of this information is defined in the supplement for a payment protocol.
ProtocolBrandPackagedContent	The Protocol Brand Packaged Content originates from a Protocol Brand Element's content whereby the outermost element tags are prefixed with

"ProtocolBrand". Its declaration
coincides with the Packaged Content's

	declaration (cf. IOTP Specification). It contains information about the brand which might be used by the payment protocol. The content of this information is defined in the supplement for a payment protocol.
ResponsePackagedContent	Container for authentication response data. Its content originates from a Authentication Response Component's Packaged Content whose outermost element tags are prefixed with "Response". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification).
TradingRoleDataPackagedContent	The TradingRoleData Packaged Content originates from a TradingRoleData Component's content whereby the outermost element tags are prefixed with "TradingRoleData". Its declaration coincides with the Packaged Content's declaration (cf. IOTP Specification). It contains information from Merchant to Payment Handler via Consumer about the protocol which is used by the payment. The content of this information is defined in the supplement for a payment protocol. The Name attribute in this packaged contents must include prefix as "Payment:" to indicate that the payment bridge processes this, for example "Payment:SET-OD" The element's declaration coincides with the Packaged Content's declaration (cf. IOTP Specification).

Table 4: Elements

XML definition: <!ENTITY % AuthReqPackagedContent "PackagedContent"> <!ENTITY % AuthResPackagedContent "PackagedContent">

<!ENTITY % BrandPackagedContent "PackagedContent"> <!ENTITY % BrandSelInfoPackagedContent "PackagedContent"> <!ENTITY % BrandSelProtocolAmountPackagedContent "PackagedContent"> <!ENTITY % BrandSelCurrencyAmountPackagedContent

"PackagedContent">

```
<!ENTITY % ProtocolAmountPackagedContent
                                        "PackagedContent"> <!ENTITY %
PayProtocolPackagedContent   "PackagedContent">

<!ENTITY % TradingRoleDataPackagedContent "PackagedContent">

<!ENTITY % MerchantData "PackagedContent">

<!ENTITY % PaySchemePackagedContent     "PackagedContent">
```

3.3 Process States

The IOTP Payment API supports six different attribute values that encode the transaction status from the IOTP's point of view, i.e. the appropriate point of view at the interface between the IOTP Application Core and IOTP Payment Bridge. This point of view does not completely mimic the more detailed view on the actual payment by the genuine Existing Payment Software or IOTP Payment Bridge.

The following three tables distinguish between the Merchant's, Consumer's, and Payment Handlers' environment. They extend the aforementioned explanations towards the mapping between IOTP process states and the internal payment scheme related states of the Existing Payment Software/IOTP Payment Bridge.

3.3.1 Merchant

The Merchant's point of view of payment is limited to the local payment initiation being interlaced with order processing because IOTP assigns the actual payment processing to the Payment Handler.

ProcessState	Description
NotYetStarted	The Payment Transaction exists within the IOTP Application Core, i.e., the Merchant's shop has already signaled to the IOTP Application Core that an IOTP transaction has been initiated by the Consumer. However, neither any API call has been issued to the IOTP Payment Bridge nor the IOTP Order Request has been created.
InProgress	The IOTP Application changes the process state to this value when it issues the

first API call to the Payment Bridge

during Brand List compilation.

This value indicates that the Payment Bridge might have some knowledge about the expected payment or might have performed some preparatory tasks (even with the Payment Handler out-of-band to IOTP).

However, this value does not indicate that some IOTP Order Request has been created and transmitted to the Consumer.

Suspended — The IOTP transaction has been suspended before the order request block has been transmitted to the Consumer.

Implicitly, the payment is also deferred.

CompletedOk — The IOTP Order Request has been successfully created and transmitted to the Consumer. Actually, this process state indicates only that the order processing has been finished.

But it contains no indication about the status of the genuine payment, which is accepted by the Payment Handler.

However, successful order processing signals the IOTP Application Core that a payment with some specific parameters is expected within the near future. And this signal might be used by the Existing Payment Software for similar purposes. This attribute might be interpreted as successful preparation of the payment system.

Particularly, it is expected that the Existing Payment Software maps this IOTP status value to some other internal value, e.g. "NotYetStarted", that is more accurate from its point of view.

As IOTP provides no communication channel between the Merchant and Payment Handler, any change of payment process state will be initiated out-of-band to IOTP, e.g. by

electronic statements of account or

	payment scheme specific mechanisms.
Failed	The IOTP transaction, i.e. order processing, has failed for some (business) reason and it is known that no payment will occur. This indication might be used to clear all data about this transaction within the Existing Payment Bridge (by "RemovePaymentLog" or "ChangeProcessState") or to reverse any preparation (with the Payment Handler out-of-band to IOTP). However, the ideal point of view of IOTP suspects that the genuine payment transaction has been neither started nor initiated.
ProcessError	The IOTP transaction, i.e. order processing, has failed for some (technical) reason and it is known that no payment will occur. This indication might be used to clear all data about this transaction within the Existing Payment Bridge (by "RemovePaymentLog" or "ChangeProcessState") or to reverse any preparation (with the Payment Handler out-of-band to IOTP). However, the ideal point of view of IOTP suspects that the genuine payment transaction has been neither started nor initiated.

Table 5: Merchant

3.3.2 Consumer

The Consumer's IOTP Application Core restricts its point of view to the payment transaction. It is assumed that the IOTP Payment Bridge handles the preceding brand selection process in a stateless manner.

NotYetStarted	This encodes the initial process state of any IOTP payment transaction. This value

is set during brand selection but it will

	not change during the whole brand selection process, normally.
InProgress	With the issuance of the Start Payment Consumer API call, the IOTP Application Core changes the process state to this value.
Suspended	The payment transaction has been suspended. Suspension may occur anywhere during brand selection (with the Merchant) or payment processing (with the Payment Handler). On resumption, the IOTP Application Core and the IOTP Payment Bridge have to use other internal data to decide whether brand selection or actual payment processing needs to be continued, i.e., whether the process state needs to be reset to "NotYetStarted" or "InProgress". Note that the Payment API assumes stateless brand selection by the IOTP Payment Bridge. Typically, any suspension during brand selection requires the repetition of the whole process. Hereby, the IOTP Application Core might to consider any already negotiated conditions in a brand depended purchase (brand, protocol).
CompletedOk	The successful payment has been acknowledged by the Payment Handler, i.e. the successful IOTP Payment Response has been received. Implicitly, this implies successful order processing.
Failed	The IOTP transaction, i.e. order or payment processing, has failed for some (business) reason. In either case it is known that the payment will not succeed.
ProcessError	The IOTP transaction, i.e. order or payment processing, has failed for some (technical) reason. However, the local process state might be

different from that of Payment Handler.

Table 6: Consumer

3.3.3 Payment Handler

The Payment Handler is responsible for the actual payment processing. New payment transactions are reported by the Consumer with the transmission of new IOTP Payment Request Blocks. IOTP Payment Exchange Block are send by the Consumer for payment transaction continuation and resumption.

NotYetStarted	This encodes the initial process state of any payment transaction. Typically, this value will last for a short amount of time.
InProgress	The IOTP Application Core changes the process state changes to "InProgress" when the Payment Handler starts with the actual processing of the IOTP Payment Request Block. Note that this does not assume that the "StartPaymentPaymentHandler" API function has been called.
Suspended	The payment transaction has been suspended.
CompletedOk	The payment has been processed, successfully, i.e. the IOTP Payment Response Block was created and transmitted to the Consumer.
Failed	The payment transaction, has finally failed for some (business) reason. Note that this value encodes the payment state reported by the IOTP Payment Bridge on "InquireProcessState". It does neither reflect whether the payment receipt has been inquired nor whether the IOTP Payment Response Block has been created and submitted to the Consumer.
ProcessError	The payment transaction, has finally failed for some (technical) reason.

Note that this value encodes the payment

state reported by the IOTP Payment
Bridge. It does not reflect whether some
IOTP Error Block has been created and
submitted to the Consumer.

Table 7: Consumer

4. Payment API Calls

4.1 Brand Compilation Related API Calls

4.1.1 Find Accepted Payment Brand

This API finction determines the payment brands being accepted by the Payment Handler on behalf of the Merchant.

Input Parameters

- Payment Direction - provided by the IOTP Application Core
- Currency Code and Currency - provided by the IOTP Application Core
- Payment Amount - provided by the IOTP Application Core
- Merchant Payment Identifier - Merchant's unique private reference to the payment transaction
- Merchant Organisation Identifier - used for distinction between multiple merchants that share the some IOTP merchant system
- Wallet Identifier - managed by the IOTP Application Core
- Merchant Data - specific data used by the IOTP Payment Bridge which is managed in the IOTP Application Core.

XML definition:

```
<!ELEMENT FindAcceptedPaymentBrand (MerchantData*) >
<!ATTLIST FindAcceptedPaymentBrand
  PayDirection  (Debit|Credit)  #REQUIRED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  MerchantPayId  CDATA  #REQUIRED
  MerchantOrgId  CDATA  #IMPLIED
  WalletID  CDATA  #IMPLIED >
```

Output Parameters

- Payment Brand Identifier - for insertion in the Brand List

Component's Brand Element
- Payment Brand Name and language annotation - for insertion in the Brand List Component's Brand Element
- Payment Brand Logo Net Location - for insertion in the Brand List Component's Brand Element
- Payment Brand Narrative Description - for insertion in the Brand List Component's Brand Element
- (Brand) Packaged Content - further payment brand description for insertion in the Brand List Component's Brand Element

The Existing Payment Software returns an empty list of brand items, if it does not support any payment brand/payment protocol combination for the given payment parameters.

XML definition:

```
<!ELEMENT FindAcceptedPaymentBrandResponse (BrandItem*) >
<!ELEMENT BrandItem (BrandPackagedContent*) >
<!ATTLIST BrandItem
  BrandId  CDATA  #REQUIRED
  xml:lang  NMTOKEN  #IMPLIED
  BrandName  CDATA  #REQUIRED
  BrandLogoNetLocn  CDATA  #REQUIRED
  BrandNarrative  CDATA  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.1.2 Find Accepted Payment Protocol

This API function determines the instances of payment protocols (and optionally the payment brands) being accepted by the Payment Handler on behalf of the Merchant. The function might be called in two variants:

- With the Brand Identifier set on the input parameter list: The function responds with the payment protocols that fits to the submitted brand.

- Without any Brand Identifier - that allows the omission of the "Find Accepted Payment Brand" API call (cf. Section 4.1.1): This function responds with both the supported brand identifiers and the payment protocols being related by the Brand Elements.

Input Parameters

- Brand Identifier - returned by "Find Accepted Payment Brand"

- Payment Direction
- Currency Code and Currency
- Payment Amount
- Merchant Payment Identifier - Merchant's unique private reference to the payment transaction
- Merchant Organisation Identifier - used for distinction between multiple merchants that share the some IOTP merchant system
- Wallet Identifier - managed by the IOTP Application Core
- (Brand) Packaged Content - further payment brand description; returned by "Find Accepted Payment Brand"; this elements are only provided iff the Brand Identifier is set
- Merchant Data - specific data used by the IOTP Payment Bridge which is managed in the IOTP Application Core.

XML definition:

```
<!ELEMENT FindAcceptedPaymentProtocol (BrandPackagedContent*,
MerchantData?) >
<!ATTLIST FindAcceptedPaymentProtocol
  BrandId  CDATA  #IMPLIED
  PayDirection  (Debit|Credit)  #REQUIRED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  MerchantPayId  CDATA  #REQUIRED
  MerchantOrgId  CDATA  #IMPLIED
  WalletID  CDATA  #IMPLIED >
```

Output Parameters

- Payment Protocol Identifier - for insertion in the Brand List Component's Pay Protocol Element
- Protocol Brand Identifier - for insertion in the Protocol Brand Element of the Brand List Component's Brand Element
- Payment Protocol Name and language annotation- for insertion in the Brand List Component's Pay Protocol Element
- Payment Request Net Location - for insertion in the Brand List Component's Pay Protocol Element
- Secured Payment Request Net Location - for insertion in the Brand List Component's Pay Protocol Element
- Brand Item List (cf. Section 4.1.1) - there must be at least one element if no brand identifier has been provided on the input parameter list.
- (Protocol Amount) Packaged Content - for insertion in the Brand List Component's Protocol Amount Element
- (Pay Protocol) Packaged Content - for insertion in the Brand List Component's Pay Protocol Element
- Currency Amount element - quite similar to the definition in

[IOTP], that contain
- refined Currency Code and Currency - for insertion in the

Brand List Component's Currency Amount Element
- refined Payment Amount - for insertion in the Brand List Component's Currency Amount Element

- Brand - there must be at least one element in each Protocol Item if no brand identifier has been provided on the input parameter list.

XML definition:

```
<!ELEMENT FindAcceptedPaymentProtocolResponse (ProtocolItem+,
BrandItem*) >
<!ELEMENT ProtocolItem (ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*
CurrencyAmount+, Brand*,ProtocolBrand*)>
<!ATTLIST ProtocolItem
  ProtocolId  CDATA  #REQUIRED
  ProtocolBrandId  CDATA  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  ProtocolName  CDATA  #REQUIRED
  PayReqNetLocn  CDATA  #IMPLIED
  SecPayReqNetLocn  CDATA  #IMPLIED >

<!ELEMENT Brand EMPTY >
<!ATTLIST Brand
  BrandId  CDATA  #REQUIRED >

<!ELEMENT CurrencyAmount EMPTY >
<!ATTLIST CurrencyAmount
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #IMPLIED
  Amount  CDATA  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.1.3 Get Payment Initialization Data

This API function provides the remaining initialization data being required by the Consumer's or Payment Handler's Existing Payment Software. This function might be called both for "brand dependent" and "brand independent" transaction types. In ether case, this function is called with one particular brand.

Input Parameters

- Brand Identifier - returned by "Find Accepted Payment Brand"
- Merchant Payment Identifier - Merchant's unique private

reference to the payment transaction

- Payment Direction
- Currency Code and Currency - from the Brand List Component's Currency Amount Element
- Payment Amount - from the Brand List Component's Currency Amount Element
- Payment Protocol Identifier - from the Brand List Component's Pay Protocol Element
- Protocol Brand Identifier - from the Protocol Brand Element which relates to the selected Brand Element, if any

- (TradingRoleData) Receiver Organization Identifier

- OkFrom, OkTo - identical to the entries of the Order Component

Merchant Payment Identifier

- Merchant Organisation Identifier - used for distinction between multiple merchants that share the some IOTP merchant system
- Wallet Identifier and/or Pass Phrase

Protocol Brand Element

- (Brand) Packaged Content - further payment brand description, from the Brand List Component's Brand Element
- (Protocol Amount) Packaged Content - further payment protocol description, from the Brand List Component's Protocol Amount Element
- (Pay Protocol) Packaged Content - further payment protocol description, from the Brand List Component's Pay Protocol Element

- (Protocol Brand) Packaged Content - further brand information, from the Protocol Brand Element of the Brand List Component which relates to the selected Brand Element, if any

- (Order) Packaged Content - further order description, from the Order Element
- three Brand Selection Info Packaged Content elements - copied from the Brand Selection Component on brand dependent purchases
- Brand - additional data about the payment brand
- Protocol Amount - additional data about the payment protocol
- Currency Amount - additional payment brand and currency specific data
- Merchant Data - specific data used by the IOTP Payment Bridge which is managed in the IOTP Application Core.

XML definition:

```
<!ELEMENT GetPaymentInitializationData (ProtocolBrand?
```

BrandPackagedContent*

```
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*,
OrderPackagedContent*,
BrandSelBrandInfoPackagedContent*,
BrandSelProtocolAmountInfoPackagedContent*,
BrandSelCurrencyAmountInfoPackagedContent*,
MerchantData*) >
<!ATTLIST GetPaymentInitializationData
  BrandId  CDATA  #REQUIRED
  MerchantPayId  CDATA  #REQUIRED
  PayDirection  (Debit|Credit)  #REQUIRED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  ProtocolId  CDATA  #REQUIRED
  OkFrom  CDATA  #REQUIRED
  OkTo  CDATA  #REQUIRED
  ReceiverOrgId  CDATA  #IMPLIED
  MerchantOrgId  CDATA  #IMPLIED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- OkFrom, OkTo - for insertion in the Payment Component
- (TradingRoleData) Packaged Content - further payment protocol description; the Name Attribute of the packaged Content element must include "Payment:" as the prefix, for example "Payment:SET-OD".
- (Order) Packaged Content - defaults to the supplied order packaged content if omitted.

XML definition:

```
<!ELEMENT GetPaymentInitializationDataResponse
(OrderPackagedContent*,
TradingRoleDataPackagedContent*) >
<!ATTLIST GetPaymentInitializationDataResponse
  OkFrom  CDATA  #IMPLIED
  OkTo  CDATA  #IMPLIED>
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.1.4 Inquire Authentication Challenge

This API function inquires any payment protocol specific

authentication challenge value from the IOTP Payment Bridge. In

Baseline IOTP this API function is called by the Merchant (or Financial Institution). The IOTP Application Core may propose a choice of algorithms to the IOTP Payment Bridge. However, the IOTP Payment Bridge may ignore the proposal and select some other algorithm.

The inquiry is assumed to be stateless. E.g., the IOTP Application Core may check the returned algorithm and stop transaction processing without notifying the IOTP Payment Bridge.

The IOTP Application Core may issue several API calls to the IOTP Payment Bridge to build up the IOTP Authentication Request Block. Any subsequently submitted choice of algorithms should be reduced by the accepted algorithms from earlier API responses.

The IOTP Payment Bridge responds with the Business Error Code if it does not provide any (more) authentication algorithms and challenges.

Input Parameters

- Authentication Identifier - the authenticator may provide its payment identifier, i.e., Payment Handler or Merchant Payment Identifier.
- Wallet Identifier and/or Pass Phrase
- set of pre-selected algorithms for authentication

XML definition:

```
<!ELEMENT InquireAuthChallenge (Algorithm*) >
<!ATTLIST InquireAuthChallenge
  AuthenticationId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- list of Authentication Challenge Packaged Contents - for insertion into the IOTP Authentication Request Component
- Algorithm Element - for insertion into the IOTP Authentication Request Component

XML definition:

```
<!ELEMENT InquireAuthChallengeResponse (AuthReqPackagedContent*,
Algorithm) >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.1.5 Authenticate

The Consumer's IOTP Application Core defers payment protocol specific authentication processing and the current challenge value to the active IOTP Payment Bridge. Alternative authentication algorithms might be tried sequentially or offered to the user for selection.

Note that the IOTP Application Core has to consider both the current context and the algorithm in order to determine the responsible IOTP Payment Bridge.

Failed authentication is reported by the Business Error Code which might trigger the inquiry of the details ("Inquire Process State"). Final failures might be encoded by the process state "Failed".

Input Parameters

- Authentication Identifier
- Wallet Identifier and/or Pass Phrase
- Authentication Challenge Packaged Content - copied from the IOTP Authentication Request Component
- Algorithm Element - copied from the IOTP Authentication Request Component

XML definition:

```
<!ELEMENT Authenticate (Algorithm, AuthReqPackagedContent*) >
<!ATTLIST Authenticate
  AuthenticationId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Authentication Response Packaged Content - for insertion into the IOTP Authentication Response Component

XML definition:

```
<!ELEMENT AuthenticateResponse (AuthResPackagedContent*) >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.1.6 Check Authentication Response

This API function verifies the Consumer's payment protocol specific

authentication response. In Baseline IOTP this API function is called

by the Merchant (or the Financial Institution). It is called only if the counter party has responded with an IOTP Authentication Response Component within the Authentication Response Block. Of course, the IOTP Application Core traces the need of such an response.

Due to the authentication's statelessness, all parameters (algorithm, challenge and response) are submitted to the IOTP Payment Bridge. Authentication failure is reported by a Process State different from "CompletedOK".

Input Parameters

- Authentication Identifier
- Wallet Identifier and/or Pass Phrase
- Authentication Challenge Packaged Content - generated by previous "Inquire Authentication Challenge" API call
- Algorithm Element
- Authentication Response Packaged Content - copied from the Authentication Response Component

XML definition:

```
<!ELEMENT CheckAuthResponse (Algorithm, AuthReqPackagedContent*,
AuthResPackagedContent*) >
<!ATTLIST CheckAuthResponse
  AuthenticationId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Current Process (Authentication) State
- Completion Code
- Status Description and its language annotation

XML definition:

```
<!ELEMENT CheckAuthResponseResponse EMPTY >
<!ATTLIST CheckAuthResponseResponse
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
   CompletedOk |
   Failed |
   ProcessError)#REQUIRED
  CompletionCode  NMTOKEN  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  StatusDesc  CDATA  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3

introduces the Error Codes.

4.2 Brand Selection Related API Calls

4.2.1 Find Payment Instrument

This API function determines which instances of a Payment Brand, e.g., two Mondex cards, are present. The same physical card may even represent multiple payment instruments.

The IOTP Application Core supplies possible payment brand and payment protocol to the IOTP Payment Bridge that has to be considered when the IOTP Payment Bridge searches for appropriate payment instruments. This set represents the (sub)set of payment alternatives being supported by the Merchant. If the IOTP Application Cote has multiple possible payment brand/protocol, it can call this function in turn.

The Existing Payment Software responds with PayInstrument Elements with empty PayInstId attributes if it does not distinguish between different payment instruments for the particular payment alternatives.

Note that the Payment API assumes that the values of the attributes BrandId, ProtocolId, ProtocolBrandId and the currency amount suffice for the determination of the appropriate Packaged Content Element that will be transmitted to the Payment Handler later on.

Input Parameters

- Brand Identifier - copied from the Brand List Component's Brand Element
- Payment Protocol Identifier and associated Protocol Brand Identifier
- Payment Direction - copied from the Brand List Component
- Currency Code and Currency - copied from the Currency Amount Element
- Payment Amount - copied from the Currency Amount Element
- Consumer Payment Identifier - Consumer's unique reference to the current payment transaction
- Wallet Identifier - managed by the IOTP Application Core
- (Brand) Packaged Content - further payment brand description; copied from the Brand List Component's Brand Element
- (Protocol Brand) Element - further information; copied from the Protocol Brand Element of the Brand List Component which

relates to the Consumer selected Brand Element, if any.

- (Protocol Amount) Packaged Content - further payment protocol description, copied from the Brand List Component's Protocol Amount Element
- Element (Protocol) Packaged Content - further payment protocol description, copied from the Brand List Component's Pay Protocol Element

XML definition:

```
<!ELEMENT FindPaymentInstrument (BrandPackagedContent*,
  ProtocolBrand?,
  PayProtocolPackagedContent*,
  ProtocolAmountPackagedContent*) >
<!ATTLIST FindPaymentInstrument
  BrandId  CDATA  #REQUIRED
  ProtocolId  CDATA  #REQUIRED
  PayDirection  (Debit|Credit)  #REQUIRED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  ConsumerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED >
```

Output Parameters

- The known Payment Instrument Identifiers, these are internal values
- The user-defined names of the payment instrument and their language encoding

The Existing Payment Software responds with an empty list of identifiers, either if it does not distinguish between different payment instruments or if there are no registered payment instruments available despite brand support for at least one (unspecified) payment protocol. In the latter case, the IOTP Payment Bridge has to request the registration of a suitable payment instrument at a subsequent step of the payment process.

XML definition:

```
<!ELEMENT FindPaymentInstrumentResponse (PayInstrument*) >
<!ELEMENT PayInstrument EMPTY >
<!ATTLIST PayInstrument
  Id  CDATA  #REQUIRED
  xml:lang  NMTOKEN  #IMPLIED
  PayInstName  CDATA  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.2.2 Check Payment Possibility

This API function checks whether a payment (both debit and credit) can go ahead. It can be used, for example, to check

- if there are sufficient funds available in a particular currency for an electronic cash payment brand,
- whether there is sufficient value space left on the payment instrument for payment refund,
- whether required system resources are available and properly configured, e.g., serial ports or baud rate,
- whether environment requirements are fulfilled, e.g., chip card reader presence or Internet connection.

If the payment method bases on external components, e.g., magnetic stripe or chip cards, and the check accesses the medium, the existing payment method should not mutually exclusive lock system resources, e.g., serial port or modem, that may also be required by other Existing Payment Software, e.g., multiple payment software components may share the same card reader. If this happens for API internal request processing, the function has to unlock these components prior to return. Otherwise, the payment may not proceed if the Consumer cancels immediately and decides to use another payment instrument. In this event the previous IOTP Payment Bridge is not notified about the change.

This function call happens immediately after the Consumer's payment instrument selection. For example, if the payment instrument is a chip card, that is not inserted in the chip card reader, the Consumer may be prompted for its insertion. However, the Consumer should be able to hit some 'skip' button, if the payment check is part of the actual payment protocol, too. Finally, the IOTP Payment Bridge may provide only a subset of these capabilities or may even directly generate a successful response without any checks.

Input Parameters

- Brand Identifier - user selection
- Payment Instrument Identifier - user selection
- Currency Code and Currency Code Type - copied from the selected Currency Amount Element
- Payment Amount - copied from the selected Currency Amount Element
- Payment Direction - copied from the selected Trading Protocol Option Block
- Protocol Identifier - copied from the selected Pay Protocol Element
- Protocol Brand Identifier - copied from the selected Protocol

Brand Element of the Brand List Component which relates to the selected Brand Element, if any

- Consumer Payment Identifier - Consumer's unique reference to the current payment transaction
- Wallet Identifier and/or Pass Phrase
- (Brand) Packaged Content - copied from the selected Brand Element
- (Protocol Amount) Packaged Content - copied from the selected Protocol Amount Element
- (Protocol) Packaged Content - copied from the selected Pay Protocol Element
- (Protocol Brand) Packaged Content - copied from the selected Protocol Brand Element of the Brand List Component which relates to the selected Brand Element, if any

XML definition:

```
<!ELEMENT CheckPaymentPossibility (BrandPackagedContent*,
ProtocolBrand?
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*>
<!ATTLIST CheckPaymentPossibility
  BrandId  CDATA  #REQUIRED
  PaymentInstrumentId  CDATA  #IMPLIED
  PayDirection  (Debit|Credit)  #REQUIRED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  ProtocolId  CDATA  #REQUIRED
  ConsumerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- three Brand Selection Info Packaged Content elements - for insertion into the Brand Selection component
- Brand - additional data about the payment brand
- Protocol Amount - additional data about the payment protocol
- Currency Amount - additional payment brand and currency specific data

XML definition:

```
<!ELEMENT CheckPaymentPossibilityResponse
(BrandSelBrandInfoPackagedContent*,
BrandSelProtocolAmountInfoPackagedContent*,
BrandSelCurrencyAmountInfoPackagedContent*) >
<!ATTLIST CheckPaymentPossibilityResponse >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3 Payment Transaction Related API calls

These Payment API calls may be made either by the Consumer's or Payment Handler's IOTP Application Core.

4.3.1 Start Payment Consumer

This API function initiates the genuine payment transaction at the Consumer side. The IOTP Payment Bridge and the Existing Payment Software perform all necessary initialization and preparation for payment transaction processing. This includes the reservation of external periphery. E.g., 1) the Consumer's chip card reader needs to be protected against access from other software components, 2) the insertion of the chip card may be requested, 3) the Internet connection may be re-established, or 4) the Payment Handler may open a mutual exclusive session to the security hardware.

The IOTP Payment Bridge monitors the payment progress and stores the current payment states such that resumption - even after power failures - remains possible. Note that the IOTP Application Core supplies only a subset of the following input parameter to the associated resumption API function and refers to the payment transaction through the party's payment identifier.

Input Parameters

- Brand Identifier - copied from the selected Brand Element
- Payment Instrument Identifier - the user selection
- Currency Code and Currency - copied from the selected Currency Amount Element
- Payment Amount - copied from the selected Currency Amount Element
- Payment Direction - copied from the Brand List Component
- Protocol Identifier - copied from the selected Payment Protocol Element
- Protocol Brand Element - further information; copied from the Protocol Brand Element of the Brand List Component which relates to the selected Brand Element, if any
- OkFrom, OkTo - copied from the Payment Component
- Consumer Payment Identifier - Consumer's unique reference to the current payment transaction
- Wallet Identifier and/or Pass Phrase
- Call Back Function - used for end user notification/logging purposes
- Call Back Language List. This list is required if the Call Back Function is set
- (Brand) Packaged Content - further payment brand description;

copied from the selected Brand Element's content

- (Protocol Amount) Packaged Content - further payment protocol description; copied from the selected Protocol Amount Element's content
- (Payment Protocol) Packaged Content - further payment protocol description; copied from the selected Pay Protocol Element's content
- (Order) Packaged Content - further order description, copied from the Order Component

XML definition:

```
<!ELEMENT StartPaymentConsumer (BrandPackagedContent*,
ProtocolBrand?
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*,
OrderPackagedContent*) >
<!ATTLIST StartPaymentConsumer
  BrandId  CDATA  #REQUIRED
  PaymentInstrumentId  CDATA  #IMPLIED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  PayDirection  (Debit|Credit)  #REQUIRED
  ProtocolId  CDATA  #REQUIRED
  ProtocolBrandId  CDATA  #IMPLIED
  OkFrom  CDATA  #REQUIRED
  OkTo  CDATA  #REQUIRED
  ConsumerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED
  CallBackFunction  CDATA  #IMPLIED
  CallBackLanguageList  NMTOKENS  #IMPLIED >
```

Output Parameters

- Continuation Status
- (Payment Scheme) Packaged Content - for insertion into the Payment Scheme Component of the IOTP Payment Request Block

The IOTP Application Core is allowed to reissue this request several times on failed analyses of the response.

XML definition:

```
<!ELEMENT StartPaymentConsumerResponse
(PaySchemePackagedContent*) >
<!ATTLIST StartPaymentConsumerResponse
  ContStatus  (End|Continue)  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3.2 Start Payment Payment Handler

This API function initializes the Consumer initiated payment transaction at the Payment Handler's side. Similar to the Consumer's system, the IOTP Payment Bridge and the Existing Payment Software perform all necessary initialization and preparation for payment transaction processing.

Input Parameters

- Brand Identifier - copied from the Consumer selected Brand Element
- Consumer Payment Identifier - copied from the Payment Scheme Component
- Currency Code and Currency - copied from the Consumer selected Currency Amount Element
- Payment Amount - copied from the Consumer selected Currency Amount Element
- Payment Direction - copied from the Brand List Component
- Protocol Identifier - copied from the Consumer selected Payment Protocol Element
- Protocol Brand Identifier - copied from the Brand Protocol Element of the Brand List Component which relates to the Consumer selected Brand Element, if any
- OkFrom, OkTo - copied from the Payment Component
- Payment Handler Payment Identifier - Payment Handler's unique reference to the current payment transaction
- Merchant Organisation Identifier - copied from the Merchant's Organisation Element
- Wallet Identifier - renaming to till identifier neglected - and/or Pass Phrase
- Call Back Function - used for end user notification/logging purposes
- Call Back Language List. This list is required if the call back function is set
- (Brand) Packaged Content - further payment brand description; copied from the Consumer selected Brand Element's content
- (Protocol Brand) Packaged Content - further information; copied from the Protocol Brand Element of the Brand List Component which relates to the Consumer selected Brand Element, if any.
- (Protocol Amount) Packaged Content - further payment protocol description; copied from the Consumer selected Protocol Amount Element's content

- (Protocol) Packaged Content - further payment protocol description; copied from the Consumer selected Pay Protocol

Element's content
- (TradingRoleData) Packaged Content - further payment protocol description; the Name Attribute of the packaged contents must include "Payment:" as the prefix, for example "Payment:SET-OD".
- Three Brand Selection Info Packaged Content Elements - copied from the Brand Selection Component
- Brand - additional data about the payment brand
- Protocol Amount - additional data about the payment protocol
- Currency Amount - additional payment brand and currency specific data
- (Payment Scheme) Packaged Content.

XML definition:

```
<!ELEMENT StartPaymentPaymentHandler (BrandPackagedContent*,
ProtocolBrand?,
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*,
BrandSelBrandInfoPackagedContent*,
BrandSelProtocolAmountInfoPackagedContent*,
BrandSelCurrencyAmountInfoPackagedContent*,
TradingRoleDataPackagedContent*,
PaySchemePackagedContent*) >
<!ATTLIST StartPaymentPaymentHandler
  BrandId  CDATA  #REQUIRED
  ConsumerPayId  CDATA  #IMPLIED
  CurrCodeType  NMTOKEN  'ISO4217-A'
  CurrCode  CDATA  #REQUIRED
  Amount  CDATA  #REQUIRED
  PayDirection  (Debit|Credit)  #REQUIRED
  ProtocolId  CDATA  #REQUIRED
  OkFrom  CDATA  #REQUIRED
  OkTo  CDATA  #REQUIRED
  PaymentHandlerPayId  CDATA  #REQUIRED
  MerchantOrgId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED
  CallBackFunction  CDATA  #IMPLIED
  CallBackLanguageList  NMTOKENS  #IMPLIED >
```

Output Parameters

- Continuation Status
- (Payment Scheme) Packaged Content - for insertion into the Payment Scheme Component of the IOTP Payment Exchange Block

The response message must contain payment schema data if the continuation status signals "Continue". The IOTP Application Core is

allowed to reissue this request several times on failed analyses of the response.

XML definition:

```
<!ELEMENT StartPaymentPaymentHandlerResponse
(PaySchemePackagedContent*) >
<!ATTLIST StartPaymentPaymentHandlerResponse
  ContStatus  (End|Continue)  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3.3 Resume Payment Consumer

This API function resumes a previously suspended payment at the Consumer side. Resumption includes the internal inquiry of the payment transaction data, e.g., payment amount, protocol identifier, and the whole initialization as it has been applied on the "Start Payment Consumer" API request.

It is up to the IOTP Application Core to decide whether an IOTP Payment equest Block or a IOTP Payment Exchange Block needs to be generated. One indicator might be the receipt of a previous IOTP Payment Exchange Block from the Payment Handler, e.g., the knowledge of the Payment Handler Payment Identifier.

Input Parameters

- Consumer Payment Identifier
- Wallet Identifier and/or Pass Phrase
- Call Back Function - used for end user notification/logging purposes

XML definition:

```
<!ELEMENT ResumePaymentConsumer EMPTY >
<!ATTLIST ResumePaymentConsumer
  ConsumerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED
  CallBackFunction  CDATA  #IMPLIED
  CallBackLanguageList  NMTOKENS  #IMPLIED >
```

Output Parameters

- Continuation Status
- (Payment Scheme) Packaged Content - for insertion in the Payment Scheme Component of the next IOTP message (Payment Exchange or Request Block).

The IOTP Application Core is allowed to reissue this request several times on failed analyses of the response. However, the IOTP Payment Bridge might reject the resumption request by using the "AttNotSupp" Error Code "naming" the Consumer Payment Identifier attribute. Then the Consumer has to apply normal error processing to the current (sub-)transaction and to issue a new Payment Request Block to the Payment Handler.

XML definition:

```
<!ELEMENT ResumePaymentConsumerResponse
(PaySchemePackagedContent*) >
<!ATTLIST ResumePaymentConsumerResponse
  ContStatus  (End|Continue)  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3.4 Resume Payment Payment Handler

This API function resumes a payment at the Payment Handler side.

Input Parameters

- Payment Handler Payment Identifier
- Wallet Identifier - renaming to till identifier neglected - and Pass Phrase
- Call Back Function - used for end user notification/logging purposes
- Call Back Language List. This list is required if the Call Back Function is set
- (Payment Scheme) Packaged Content - copied from the Payment Scheme Component of the received IOTP message (Payment Exchange or Request Block).

XML definition:

```
<!ELEMENT ResumePaymentPaymentHandler
(PaySchemePackagedContent*) >
<!ATTLIST ResumePaymentPaymentHandler
  PaymentHandlerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED
  CallBackFunction  CDATA  #IMPLIED
  CallBackLanguageList  NMTOKENS  #IMPLIED >
```

Output Parameters

- Continuation Status
- (Payment Scheme) Packaged Content - for insertion in the Payment Scheme Component of the next Payment Exchange Block.

The response message contains payment schema specific data if the continuation status signals "Continue". The IOTP Application Core is allowed to reissue this request several times on failed analyses of the response.

XML definition:

```
<!ELEMENT ResumePaymentPaymentHandlerResponse
(PaySchemePackagedContent*) >
<!ATTLIST ResumePaymentPaymentHandlerResponse
  ContStatus  (End|Continue)  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3.5 Continue Process

This API function passes one specific IOTP Payment Scheme Component, i.e., the encapsulated Packaged Content elements, received from the counter party (e.g. Consumer) to the IOTP Payment Bridge and responds with the next IOTP Payment Scheme Component for submission to the counter party.

Input Parameters

- Payty's Payment Idetifier
- Process (Transaction) Type which distinguishes between Payments and Inquiries.
- Wallet Identifier and/or Pass Phrase
- (Payment Scheme) Packaged Content - copied from the Payment Scheme Component of the received Payment Exchange Block or from the Error Block.

Each party should set the payment identifier with the local identifier (Consumer: ConsumerPayId; Merchant: MerchantPayId; Payment Handler: PaymentHandlerPayId).

XML definition:

```
<!ELEMENT ContinueProcess (PaySchemePackagedContent+) >
<!ATTLIST ContinueProcess
  PayId  CDATA  #REQUIRED
  ProcessType  (Payment | Inquiry) 'Payment'
```

```
     WalletID  CDATA  #IMPLIED
```

```
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Continuation Status
- (Payment Scheme) Packaged Content - for insertion in the Payment Scheme Component of the next Payment Exchange Block or final Payment Response Block

The response message contains payment schema data if the continuation status signals "Continue". The IOTP Payment Bridge must signal "End", if the payment scheme data was received within an IOTP Error Block containing an Error Component with severity HardError.

XML definition:

```
<!ELEMENT ContinueProcessResponse (PaySchemePackagedContent*) >
<!ATTLIST ContinueProcessResponse
  ContStatus  (End|Continue)  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.3.6 Change Process State

The IOTP Application Core changes the current payment status by this request. The IOTP Payment Bridge may be notified about business level normal termination, cancellation, suspension, and processing errors. Notification happens by requesting the intended process state.

The IOTP Payment Bridge processes the status change and reports the result.

The IOTP Application Core has to analyze any returned process status in order to check whether the IOTP Payment Bridge has agreed to or declined the status switch. E.g., the submitted Process State "CompleteOk" may lead to the Payment Status "Failed" if the payment transaction has already failed.

Transaction Suspension is notified by the newly introduced Process State "Suspended". The other attribute values have been taken from the IOTP specification.

This API function might be called by the Consumer, Merchant, or Payment Handler for each payment transaction anytime after the issuance of "FindPaymentInstrument" to the IOTP Payment Bridge by the Consumer, the issuance of "FindAcceptedPaymentBrand" by the Merchant,

or the issuance of "StartPaymentPaymentHandler" by the Payment

Handler.

The Process States "CompletedOk", "Failed", and "ProcessError" are final in the sense that they can not be changed on subsequent calls. However, the API function should not return with an error code if such an incompatible call has been issued. Instead it should report the old unchanged Process State.

Unknown payment transactions are reported by the Error Code "AttValInvalid" pointing to the PayId attribute.

Input Parameters

- Party's Payment Identifier
- intended Payment Status
- intended Completion Code
- Process (Transaction) Type which distinguishes between Payments and Inquiries.
- Wallet Identifier and/or Pass Phrase

XML definition:

```
<!ELEMENT ChangeProcessState EMPTY >
<!ATTLIST ChangeProcessState
  PayId  CDATA  #REQUIRED
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
   CompletedOk |
   Failed |
   ProcessError)  #REQUIRED
  CompletionCode  NMTOKEN  #IMPLIED
  ProcessType  (Payment | Inquiry) 'Payment'
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Process State and Percent Complete
- Completion Code
- Status Description and its language annotation

XML definition:

```
<!ELEMENT ChangeProcessStateResponse EMPTY >
<!ATTLIST ChangeProcessStateResponse
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
```

```
      CompletedOk |
```

```
    Failed |
    ProcessError)  #REQUIRED
  PercentComplete  CDATA  #IMPLIED
  CompletionCode  NMTOKEN  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  StatusDesc  CDATA  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.4 General Inquiry API Calls

The following calls are not necessarily assigned to a payment transaction and may be issued at any time. There are no dependencies on any other calls.

4.4.1 Remove Payment Log

The IOTP Application Core notifies the IOTP Payment Bridge and/or the corresponding Existing Payment Software via IOTP Payment Bridge that any record in the Payment Log file, that deals with the listed payment transaction, might be removed.

Input Parameters

- Party's Payment Identifier
- Wallet Identifier and/or Pass Phrase

XML definition:

```
<!ELEMENT RemovePaymentLog EMPTY >
<!ATTLIST RemovePaymentLog
  PayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

XML definition:

```
<!ELEMENT RemovePaymentLogResponse EMPTY >
<!ATTLIST RemovePaymentLogResponse >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.4.2 Payment Instrument Inquiry

This API function retrieves the properties of the Payment Instrument. The Payment Instrument Identifier could be omitted if this identifier is derived by other means, e.g., by analysis of the currently inserted chip card. If the Payment instrument could not uniquely determined, the IOTP Payment Bridge may provide suitable dialogs for user input.

E.g., this API function might be used during problem resolution with the Customer Care Provider of the issuer of the payment instrument, in order to inquire payment instrument specific values.

Input parameters

- Brand Identifier
- Payment Instrument Identifier
- Protocol Identifier
- Wallet Identifier and/or Pass Phrase
- Property Type List - sequence of values whose language is identified by xml:lang
- (Brand) PackagedContent Content - further payment brand description
- Protocol Brand Content - further payment brand information
- (Protocol Amount) PackagedContent Content - further payment protocol description
- (Pay Protocol) PackagedContent Content - further payment protocol description

The codes in the property type list are of two types:

- generic codes which apply to all payment methods but might be unavailable
- Payment Brand specific codes.

Generic codes for the Property Type List are:

Property Type	Meaning
Balance	Current balance
Limit	Maximum balance
PaymentLimit	Maximum payment transaction limit
Expiration	Expiration date
Identifier	Issuer assigned identifier of the payment instrument. Usually, it does not match with the API's payment instrument identifier.
LogEntries	Number of stored payment transaction entries. The entries are numbered from 0 (the most recent) to some non-negative value for the oldest entry.

PayAmountn	Payment Amount of the n-th recorded payment

	transaction, n may negative
PayPartyn	Remote party of the n-th payment recorded transaction, n may negative
PayTimen	Time of the n-th payment recorded transaction, n may negative

XML definition:

```
<!ELEMENT PaymentInstrumentInquiry (BrandPackagedContent*,
ProtocolBrand?,
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*) >
<!ATTLIST PaymentInstrumentInquiry
  BrandId  CDATA  #REQUIRED
  PaymentInstrumentId  CDATA  #IMPLIED
  ProtocolId  CDATA  #REQUIRED
  PropertyTypeList  NMTOKENS  #REQUIRED
  xml:lang  NMTOKEN  #IMPLIED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output parameters

- a list of zero or more unavailable property values whose language are identified by xml:lang.
- a list of zero or more sets of "Properties Types", "Property Values" and "Property Descriptions"

XML definition:

```
<!ELEMENT PaymentInstrumentInquiryResponse
(PaymentInstrumentProperty*) >
<!ATTLIST PaymentInstrumentInquiryResponse
  xml:lang  NMTOKEN  #REQUIRED
  UnavailablePropertyList NMTOKENS  #IMPLIED >

<!ELEMENT PaymentInstrumentProperty EMPTY >
<!ATTLIST PaymentInstrumentProperty
  PropertyType  NMTOKEN  #REQUIRED
  PropertyValue  CDATA  #REQUIRED
  PropertyDesc  CDATA  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.4.3 Inquire Pending Payment

This API function reports the party's payment identifiers of any pending payment transactions that the IOTP Payment Bridge/Existing Payment Software recommends to complete or suspend prior to the processing of new payment transactions. It does not respond further transaction details. These have to be inquired by "Inquire Process State".

Note that the IOTP Payment Bridge has to respond without the supplement of any pass phrase if there exist no pending payment transaction. But if there are some pending payment transactions, the IOTP Payment Bridge may refuse the immediate response and may instead request the appropriate pass phase from the IOTP Application Core.

Input Parameters

o Wallet Identifier and/or PassPhrase

XML definition:

```
<!ELEMENT InquirePendingPayment EMPTY >
<!ATTLIST InquirePendingPayment
  WalletId  CDATA  #IMPLIED
  PassPhrase  CDATA  #IMPLIED >
```

Output Parameters

o Party's Payment Identifier

XML definition:

```
<!ELEMENT InquirePendingPaymentResponse (PaymentId*) >

<!ELEMENT PaymentId EMPTY >
<!ATTLIST PaymentId
  PayId  CDATA  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.5 Payment Related Inquiry API Calls

4.5.1 Check Payment Receipt

This function is used by the Consumer and might be used by the Payment Handler to check the consistency, validity, and integrity of IOTP payment receipts whereby any receipt might consist of Packaged Content Elements

- from the IOTP Payment Receipt Component - provided by the Payment Handler's "Inquire Process State" API call shortly before payment completion,
- from Payment Scheme Components being exchanged during the actual payment, or
- being returned by the Consumer's "Inquire Process State" API call shortly before payment completion

The IOTP Application Core has to check the PayReceiptNameRefs attribute of the IOTP Payment Receipt Component and to supply exactly the Packaged Content Elements being referred to.

Failed verification is returned with a business error.

Note that this Payment API assumes that any payment receipt builds upon a subset of elements w.r.t. [IOTP]. Furthermore, the Packaged Content Element have to be distinguishable by their Name attribute.

Input Parameters

- Party's Payment Identifier
- Wallet Identifier and/or Pass Phrase
- All Packaged Content Elements that build the payment receipt

XML definition:

```
<!ELEMENT CheckPaymentReceipt (PackagedContent*) >
<!ATTLIST CheckPaymentReceipt
  PayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

XML definition:

```
<!ELEMENT CheckPaymentReceiptResponse EMPTY >
<!ATTLIST CheckPaymentReceiptResponse >
```

Tables 4 and 5 explain the attributes and elements; Table 3

introduces the Error Codes.

4.5.2 Expand Payment Receipt

This API function expands any IOTP payment receipt into a form which may be used for display or printing purposes. "Check Payment Receipt" should be used first if there is any question of the payment receipt containing errors.

There apply the same conventions to the input parameter as for "Check Payment Receipt" (cf. Section 4.5.1).

Input Parameters

- Party's Payment Identifier
- Wallet Identifier and/or Pass Phrase
- All Packaged Content Elements that build the payment receipt

XML definition:

```
<!ELEMENT ExpandPaymentReceipt (PackagedContent*) >
<!ATTLIST ExpandPaymentReceipt
  PayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Brand Identifier
- Protocol specific Brand Identifier
- Payment Instrument Identifier
- Currency Code and Currency Code Type
- Payment Amount
- Payment Direction
- Time Stamp - issuance of the receipt
- Protocol Identifier
- Protocol specific Transaction Identifier - this is an internal reference number which identifies the payment
- Consumer Description, Payment Handler Description, and a language annotation
- Style Sheet Net Location
- Payment Property List. A list of type/value/description triples which contains additional information about the payment which is not covered by any of the other output parameters; property descriptions have to consider the language annotation.

The Style Sheet Net Location refers to a Style Sheet (e.g. [XSLT])

that contains visualization information about the reported XML

encoded data.

XML definition:

```
<!ELEMENT ExpandPaymentReceiptResponse (PaymentProperty*) >
<!ATTLIST ExpandPaymentReceiptResponse
  BrandId  CDATA  #IMPLIED
  PaymentInstrumentId  CDATA  #IMPLIED
  Amount  CDATA  #IMPLIED
  CurrCodeType  NMTOKEN  #IMPLIED
  CurrCode  CDATA  #IMPLIED
  PayDirection  (Debit|Credit)  #IMPLIED
  TimeStamp  CDATA  #IMPLIED
  ProtocolId  CDATA  #IMPLIED
  ProtocolBrandId  CDATA  #IMPLIED
  ProtocolTransId  CDATA  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  ConsumerDesc  CDATA  #IMPLIED
  PaymentHandlerDesc  CDATA  #IMPLIED
  StyleSheetNetLocn  CDATA  #IMPLIED>

<!ELEMENT PaymentProperty EMPTY >
<!ATTLIST PaymentProperty
  PropertyType  NMTOKEN  #REQUIRED
  PropertyValue  CDATA  #REQUIRED
  PropertyDesc  CDATA  #REQUIRED >
```

The Existing Payment Software should return as many attributes as possible from the supplied IOTP Payment Receipt. The payment supplement define that attribute value for the payment properties.

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.5.3 Inquire Process State

This API function returns the current payment state and optionally further Packaged Content Elements that form the payment receipt. Called by the Payment Handler, the IOTP Payment Bridge might respond with data intended for inclusion in the IOTP Payment Receipt Component's Packaged Content. When the Consumer calls this function shortly before payment completion, it may respond with further items of the payment receipt. Such items might be created by a chip card.

Input Parameters

- Party's Payment Identifier

- Wallet Identifier and/or Pass Phrase

XML definition:

```
<!ELEMENT InquireProcessState EMPTY >
<!ATTLIST InquireProcessState
  PayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Current Process State and Percent Complete
- Completion Code
- Status Description and its language annotation
- Payment Receipt Name References to all Packaged Content Elements that build the payment receipt (cf. Section 4.5.1), even if they have not been created so far (Consumer's share)
- Any Packaged Content Element being available that form the payment receipt

The IOTP provides a linking capability to the payment receipt delivery. Instead of encapsulating the whole payment specific data into the packaged content of the payment receipt, other Payment Scheme Components' Packaged Content might be referred to.

XML definition:

```
<!ELEMENT InquireProcessStateResponse
(PackagedContent*) >
<!ATTLIST InquireProcessStateResponse
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
   CompletedOk |
   Failed |
   ProcessError)  #REQUIRED
  PercentComplete  CDATA  #IMPLIED
  CompletionCode  NMTOKEN  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  StatusDesc  CDATA  #IMPLIED
  PayReceiptNameRefs  NMTOKENS  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.5.4 Start Payment Inquiry

This API function responds any additional payment scheme specific data that is needed by the Payment Handler for Consumer initiated payment transaction inquiry processing. Probably, the IOTP Payment Bridge (or the corresponding Existing Payment Software) has to determine the payment related items that were provided with the "Start Payment Consumer" API function call.

Input Parameters

- Consumer Payment Identifier
- Wallet Identifier and/or Pass Phrase

XML definition:

```
<!ELEMENT StartPaymentInquiry EMPTY >
<!ATTLIST StartPaymentInquiry
  ConsumerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- (Payment Scheme) Packaged Content - intended for insertion in the Payment Scheme Component of the Inquiry Request Block

XML definition:

```
<!ELEMENT StartPaymentInquiryResponse
(PaySchemePackagedContent*) >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.5.5 Inquire Payment Status

The Payment Handler calls this API function for Consumer initiated inquiry processing. It differs from the previous "Inquire Process State" API function by the optional supplement of payment scheme specific data. The response may encapsulate further details about the payment transaction.

Input Parameters

- Payment Handler Payment Identifier
- Wallet Identifier and/or Pass Phrase

- (Payment Scheme) Packaged Content - copied from the Inquiry

Request Block's Payment Scheme Component

XML definition:

```
<!ELEMENT InquirePaymentStatus (PaySchemePackagedContent*) >
<!ATTLIST InquirePaymentStatus
  PaymentHandlerPayId  CDATA  #REQUIRED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- Current Process State
- Completion Code
- Status Description and its language annotation
- (Payment Scheme) Packaged Content - intended for insertion in the Payment Scheme Component of the Inquiry Response Block

XML definition:

```
<!ELEMENT InquirePaymentStatusResponse
(PaySchemePackagedContent*) >
<!ATTLIST InquirePaymentStatusResponse
  PaymentHandlerPayId  CDATA  #REQUIRED
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
   CompletedOk |
   Failed |
   ProcessError)  #REQUIRED
  CompletionCode  NMTOKEN  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  StatusDesc  CDATA  #IMPLIED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

4.6 Other API Calls

4.6.1 Manage Payment Software

The following API function notifies the IOTP Payment Bridge about the intended registration, modification, or deletion of a payment instrument. The actual processing is up to the IOTP Payment Bridge.

This API request may also be used to activate the IOTP Payment Bridge

(and the corresponding Existing Payment Software) for general administration purposes.

Input Parameters

- Brand Identifier
- Protocol Identifier
- Any action code:
- New - add new payment method / instrument
- Update - change the payment method's / instrument's data
- Delete - delete a payment method / instrument
- Wallet Identifier and/or Pass Phrase
- (Brand) Packaged Content - further payment brand description
- (Pay Protocol) Packaged Content - further payment protocol description
- (Protocol Amount) Packaged Content - further payment protocol description

If the Action attribute is set, the Brand and Protocol Identifier have to be set, too. The IOTP Payment Bridge has to provide the required user dialogs and selection mechanisms. E.g., updates and deletions may require the selection of the payment instrument. A new wallet might be silently generated on the supplement of a new Wallet Identifier or after an additional end user acknowledge. The IOTP Application Core should not provide any pass phrases for new wallets. Instead, the IOTP Payment Bridge has to request and verify them which may return their value to the IOTP Application Core in plain text. In addition, the IOTP Payment Bridge returns the supported authentication algorithms when a new brand and protocol pair has been registered.

If the "Action" attribute is omitted, the IOTP Payment Bridge which is responsible for the Existing Payment Software pops up in a general interactive mode.

XML definition:

```
<!ELEMENT ManagePaymentSoftware (BrandPackagedContent*,
ProtocolAmountPackagedContent*,
PayProtocolPackagedContent*) >
<!ATTLIST ManagePaymentSoftware
  BrandId  CDATA  #IMPLIED
  ProtocolId  CDATA  #IMPLIED
  Action  (New |
   Update |
   Delete)  #IMPLIED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED >
```

Output Parameters

- An action code:
- New - added new wallet
- Update - changed wallet's configuration
- Delete - removed a wallet
- Wallet Identifier and/or Pass Phrase

The IOTP Payment Bridge does not return any information about the set of registered payment instruments because these data items are dynamically inferred during the brand selection process at the beginning of each IOTP transaction. However, the IOTP Application Core has to be notified about new wallets and should be notified about updated and removed wallet (identifier)s". Alternatively, removed wallets can be implicitly detected during the next brand selection phase. Updated wallets do no affect the processing of the IOTP Application Core. The IOTP Payment Bridge should only support the addition of at most one wallet because it is not able to report multiple additions at once back to the IOTP Application Core.

XML definition:

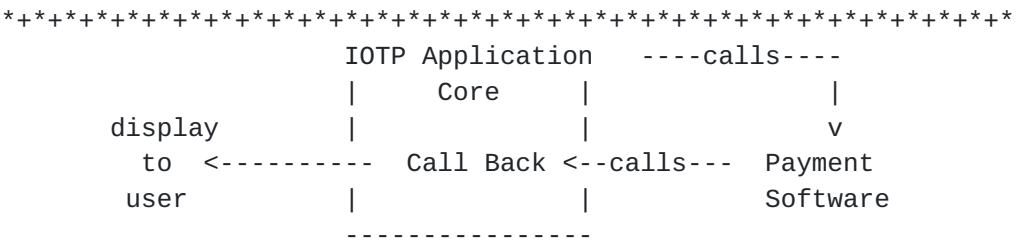
```
<!ELEMENT ManagePaymentSoftwareResponse EMPTY >
<!ATTLIST ManagePaymentSoftwareResponse
  Action  (New |
   Update |
   Delete)  #IMPLIED
  WalletID  CDATA  #IMPLIED
  Passphrase  CDATA  #IMPLIED
  AuthNames  NMTOKENS  #REQUIRED >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

5. Call Back Function

This API function, called by the IOTP Payment Bridge, is used to provide information for Consumer or Payment Handler notification about the progress of the payment transaction.

Its use is illustrated in the diagram below.



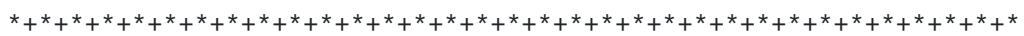


Figure 9 Call Back Function

Whenever this function is called, the content of the status description should be made available to the user. For example on a status bar, a pop up window, etc.

A reference to the Call Back function is passed as an input parameter to the "Start Payment X" and "Resume Payment X" API function. Afterwards, this function might be called whenever the status changes or progress needs to be reported.

Input Parameters

- the software identifier of the caller
- Party's Payment Identifier
- Process State and Percent Complete
- Completion Code
- Status Description and its language annotation, text which provides information about the progress of the call. It should be displayed or made available to, for example, the Consumer.

Examples of Status Description could be:

- "Paying 12.30 USD to XYZ Inc"
- "Payment completed"
- "Payment aborted"

The valid languages are announced in the Call Back Language List attribute in "Start Payment X" and "Resume Payment X" API function calls.

XML definition:

```
<!ELEMENT CallBack EMPTY >
<!ATTLIST CallBack
  ContentSoftwareID  CDATA  #IMPLIED
  PayId CDATA #REQUIRED
  ProcessState  (NotYetStarted |
   InProgress |
   Suspended |
   CompletedOk |
   Failed |
   ProcessError)  #IMPLIED
  PercentComplete  CDATA  #IMPLIED
  CompletionCode  NMTOKEN  #IMPLIED
  xml:lang  NMTOKEN  #IMPLIED
  StatusDesc  CDATA  #IMPLIED >
```

Output Parameters

XML definition:

```
<!ELEMENT CallBackResponse EMPTY >
<!ATTLIST CallBackResponse <!-- see below --> >
```

Tables 4 and 5 explain the attributes and elements; Table 3 introduces the Error Codes.

Basically, the call back function accepts all input arguments or rejects the whole request. It may even accept malformed requests.

Some payment schemes may support or require that the Consumer might be able to cancel the payment at any time. The Call Back function can be used to facilitate this by returning the cancellation request on the next call (using the Business Error Code and Completion Code "ConsCancelled").

Vice versa the Payment Handler's Application Core might use the similar mechanism to signal its IOTP Payment Bridges any exceptional need for a fast shutdown. These IOTP Payment Bridges may initiate the appropriate steps for terminating/canceling all pending payment transactions.

Note that the "Change Process State" API function provides the second mechanism for such kind of notification. Therefore, the IOTP Payment Bridge or Existing Payment Software may ignore the details of the "Call Back" response.

6. Security Consideration

[T.B.D.]

See also security consideration section of [IOTP].

Full Copyright Statement

References

[IOTP] - Internet Open Trading Protocol Specification, Version 1.0, April 2000, See RFC2801.

[IOTPBOOK] - D. Burdett, D.E. Eastlake III, and M. Goncalves, Internet Open Trading Protocol, McGraw-Hill, 2000

[HTML] - Hyper Text Mark Up Language. The Hypertext Markup Language (HTML) is a simple markup language used to create hypertext documents that are platform independent. See RFC 1866 and the World Wide Web (W3C) consortium web site at: http://www.w3.org/MarkUp/

[HTTP] - Hyper Text Transfer Protocol versions 1.0 and 1.1. See RFC 1945: Hypertext Transfer Protocol - HTTP/1.0. T. Berners-Lee, R. Fielding & H. Frystyk. May 1996. and RFC 2068: Hypertext Transfer Protocol - HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. January 1997.

[ISO4217] - ISO 4217: Codes for the Representation of Currencies.

Available from ANSI or ISO.

[MIME] - Multipurpose Internet Mail Extensions. See RFC822, RFC2045, RFC2046, RFC2047, RFC2048 and RFC2049.

[URL] - Berners-Lee, T., Masinter, L. and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[SET] - SET Secure Electronic Transaction(TM) , Version 1.0, May 31, 1997
Book 1: Business Description
Book 2: Programmer's Guide
Book 3: Formal Protocol Definition
Download from: <http://www.setco.org>.

[SET/IOTP] - Yoshiaki Kawatsura "SET Supplement for IOTP" (currently draft-ietf-trade-iotp-v1.0-set-01.txt)

[UTC] - Universal Time Coordinated. A method of defining time absolutely relative to Greenwich Mean Time (GMT). Typically of the form: "CCYY-MM- DDTHH:MM:SS.sssZ+n" where the "+n" defines the number of hours from GMT. See ISO DIS8601.

[XML] - Extensible Mark Up Language. A W3C recommendation. See http://www.w3.org/TR/1998/REC-xml-19980210

[XSLT] - Extensible Style Language Transformations 1.0, November 1999, See http://www.w3.org/TR/xslt

[XML-NS] - Namespaces in XML Recommendation. T. Bray, D. Hollander, A. Layman. Janaury 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114

Author's Address

Hans-Bernhard Beykirch and Werner Hans
IT Development & Coordination Center for the German Savings Banks Organization (SIZ)
Konigswinterer Strase 553
53227 Bonn
Germany
E-mail: Hans-Bernhard.Beykirch@siz.de, Werner.Hans@siz.de

Masaaki Hiroya and Yoshiaki Kawatsura
Hitachi, Ltd.
890 Kashimada Saiwai-ku Kawasaki-shi
Kanagawa, Japan 212-8567

E-mail: hiroya@sdl.hitachi.co.jp, kawatura@bisd.hitachi.co.jp

Expiration and File Name

This draft expires March 2001.

Its file name is draft-ietf-trade-iotp-v1.0-papi-02.txt.