WIPO
OMPI

PCT/EF/PFC 02/003

STATUS AT: December 20, 2002
SITUATION AU : 20 décembre 2002

WORLD INTELLECTUAL PROPERTY ORGANIZATION
ORGANISATION MONDIALE DE LA PROPRIÉTÉ INTELLECTUELLE
GENEVA/GENÈVE

ADMINISTRATIVE INSTRUCTIONS UNDER THE PATENT COOPERATION TREATY (PCT)

STANDARD FOR THE ELECTRONIC FILING AND PROCESSING OF INTERNATIONAL APPLICATIONS

INSTRUCTIONS ADMINISTRATIVES DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

NORME CONCERNANT LE DÉPÔT ET LE TRAITEMENT ÉLECTRONIQUES DES DEMANDES INTERNATIONALES

PROPOSAL FOR CHANGE FILE
DOSSIER RELATIF À LA PROPOSITION DE MODIFICATION

SUBJECT: / **SUJET :**	Transmission (Annex F, section 4) / Transmission (Annexe F, section 4)	**PROPOSED BY:** / **PROPOSÉE PAR :**	JP
HANDLING: / **TRAITEMENT :**	Annual cycle / Cycle annuel	**PROPOSED DATE OF ENTRY INTO FORCE:** / **DATE PROPOSÉE D'ENTRÉE EN VIGUEUR :**	03.03

ANNEX/ ANNEXE	**CONTENT/CONTENU**	**ORIGIN/ ORIGINE**	**DATE**
1	Proposal/Proposition	JP	28.02.02
2	Circular C. PCT 830/Circulaire C. PCT 830	IB	04.04.02
3	Comments/Commentaires	AU	09.05.02
4	Comments/Commentaires	JP	15.05.02
5	Comments/Commentaires	US	15.05.02
6	Comments/Commentaires	KR	17.05.02
7	Revised proposal/Proposition révisée	JP	31.05.02
8	Comments/Commentaires	EP, JP, US	01.07.02
9	Comments/Commentaires	JP	20.12.02

[Annex 1 follows/
L'annexe 1 suit]

NEXT ACTION:	Promulgation	**BY:**	March 2003
PROCHAINE ACTION :	Promulgation	**POUR LE :**	Mars 2003

PROPOSAL BY THE JAPAN PATENT OFFICE

5 TRANSMISSION

The IA package can be transmitted over secure or non-secure channels depending on the package type. This section includes the protocol to be followed as well as the package/transmission combinations that are permitted in the Applicant-Office (international phase), Office-Office, and designated Office communication sectors. While additional sectors are referred to in this standard (see section 2.3), permissible transmission/package combinations can be categorized in the three sectors listed above.

5.1 The E-filing interoperability protocol

This section describes both the transmission layer protocol between the clients and the server as well as providing a definition of the behavior required of both the client and the server.

The protocol is designed to support HTTP communication over an SSL Tunnel for all PKI based E-filing solutions and includes the following capabilities:

(a) Enables large applications to be transmitted via multiple HTTP post actions to address reliability and integrity issues
(b) Efficient error detection and correction
(c) Enables offices to control optimal transaction size

Note that this is an evolving protocol, with production systems in development at a number of IP Offices, and further revisions are foreseen.

5.1.1 Principles

The following principles have been adopted for the interoperability protocol:

(a) All communications between client and server is in the form of HTTP post actions initiated by the client
(b) All post requests and resulting responses use the same transaction management header followed by an optional data block
(c) All transmissions use the Division mechanism to divide large blobs of data into manageable chunks with a protocol that allows for retries and pacing

5.1.2 Application Layer Protocol for application

At the highest level for application, there are five events that the protocol requires a client and server to support. These events are:

(a) Begin Transaction
(b) Send Package Header
(c) Send Package Data
(d) Get Receipt
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package data contains the information for submitting application. It is a WASP consisting of various types of files.

(iii) The receipt is an acknowledgment of the submission. The content of this receipt (XML data plus an optional human readable certificate in PDF or TIFF), which is signed by the receiving Office, is defined in Annex F Appendix I. The date of receipt will be determined according to the usual principles applicable to the filing of applications on paper, including filing by electronic means (such as by facsimile transmission), that is, based on the date prevailing at the location of the Office at the time when the complete transmission of the application has been received.

5.1.2.1 Use of the SSL Tunnel for application

These events are all performed within an SSL tunnel that is established before issuing the Begin Transaction event. The SSL tunnel is built using both client and server authentication. The SSL tunnel may be stopped at the end of the transaction or, if a batch of transmissions is foreseen, the SSL tunnel can be left open and only stopped when all transmissions are complete. The SSL tunnel uses the SSL protocol version 3.0.

5.1.2.2 Application level Events for application

Start SSL session (See Figure 5)

Step 0: Begin Transaction

Client action:

Get transaction Information.

Server response:

Return values in the *transaction_id* and *max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in the submission of an application.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Send Package Header

Client action:

Send package header

Server response:

a) OK
b) Error (Abort, go back to step 0)
c) Package already received; go to step 3 to get the receipt.

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance expired), the Application Response Code (ARC) will remain OK, but the server will capture the error and provide a message on the receipt.

Step 2: Send Package Data

Client action:

Send package data

Server response:

a) OK
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package data, the server must verify the signature of the WASP and compare the message digest of the unsigned package against the message digest provided in the Package Header in Step 1 of the transaction before returning the ARC to the client. If both conditions are met, the server should return an ARC indicating OK. If the hash values in package header and the WAD of the package data do not match, the ARC value should be set to FFF7. If the signature is invalid (for instance expired), the ARC will remain OK, but the server will capture the error and provide a message on the receipt.

Step 3: Request Receipt

Client action:

Send request

Server response:

a) OK (Receipt object included in response)
b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK
b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires each individual transaction to be acknowledged by an individual receipt.

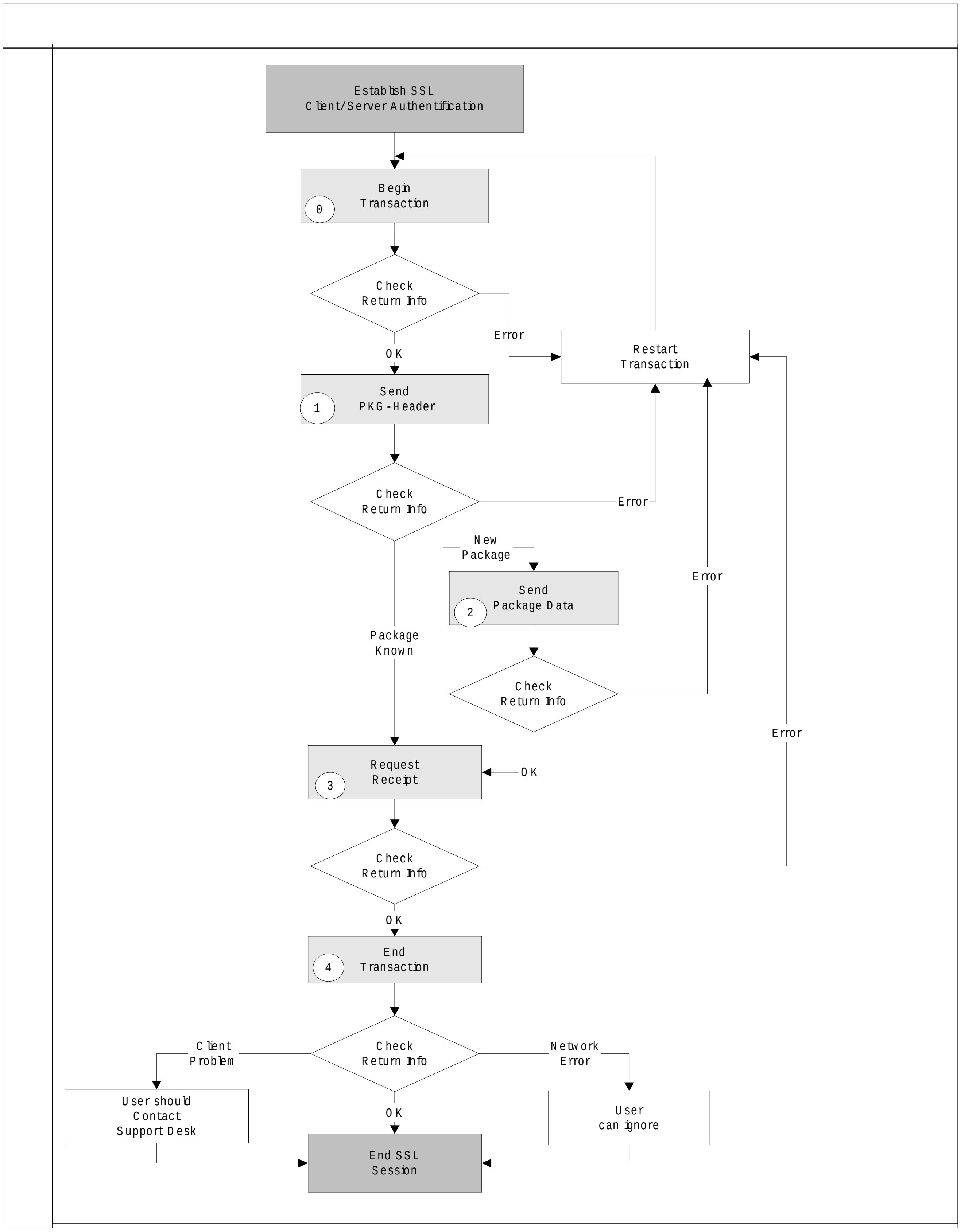


Figure 5 - Application level protocol behavior ***for application***

5.1.3 Application Layer Protocol for notification

At the highest level for notification, there are five events that the protocol requires a client and server to support. These events are:

(a) Begin Transaction
(b) Get Package Header for notification
(c) Get Package Data for notification
(d) Send Receipt Check Notice for notification
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package header contains summary information (such as a dispatch-number and the number of notifications to be sent) on the notification to be notified. It is a WASP containing the package header in XML format.
(iii) The package data contains the notification for dispatch.

5.1.1.1 Use of the SSL Tunnel for notification

Refer to Section 5.1.2.1, "Use of the SSL Tunnel for application."

5.1.1.2 Application level Events for notification

Start SSL session (See Figure 6)

Step 0: Begin Transaction
Client action:
Get transaction Information.
Server response:
Return values in the *transaction_id and max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in sending a notification.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Get Package Header for notification
Client action:
Send package header Request notification sending.
Server response:
a) OK The response contains the package header, which contains notification summary information.
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance, due to a signature verification error or validation data expiration), the application response code (ARC) value is set to FFF6.

If the number of sendable notifications in package header of Server response is "0(zero)" (there is no sendable notifications), then go to Step 4.

Step 2: Get Package Data for notification

Client action:

Send package-data request.

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 3: Send Receipt Check Notice for notification

Client action:

Send Receipt Check Notice for notification

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK

b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires that, for each transaction, the client acknowledge the reception by sending Receipt Check Notice to the server.

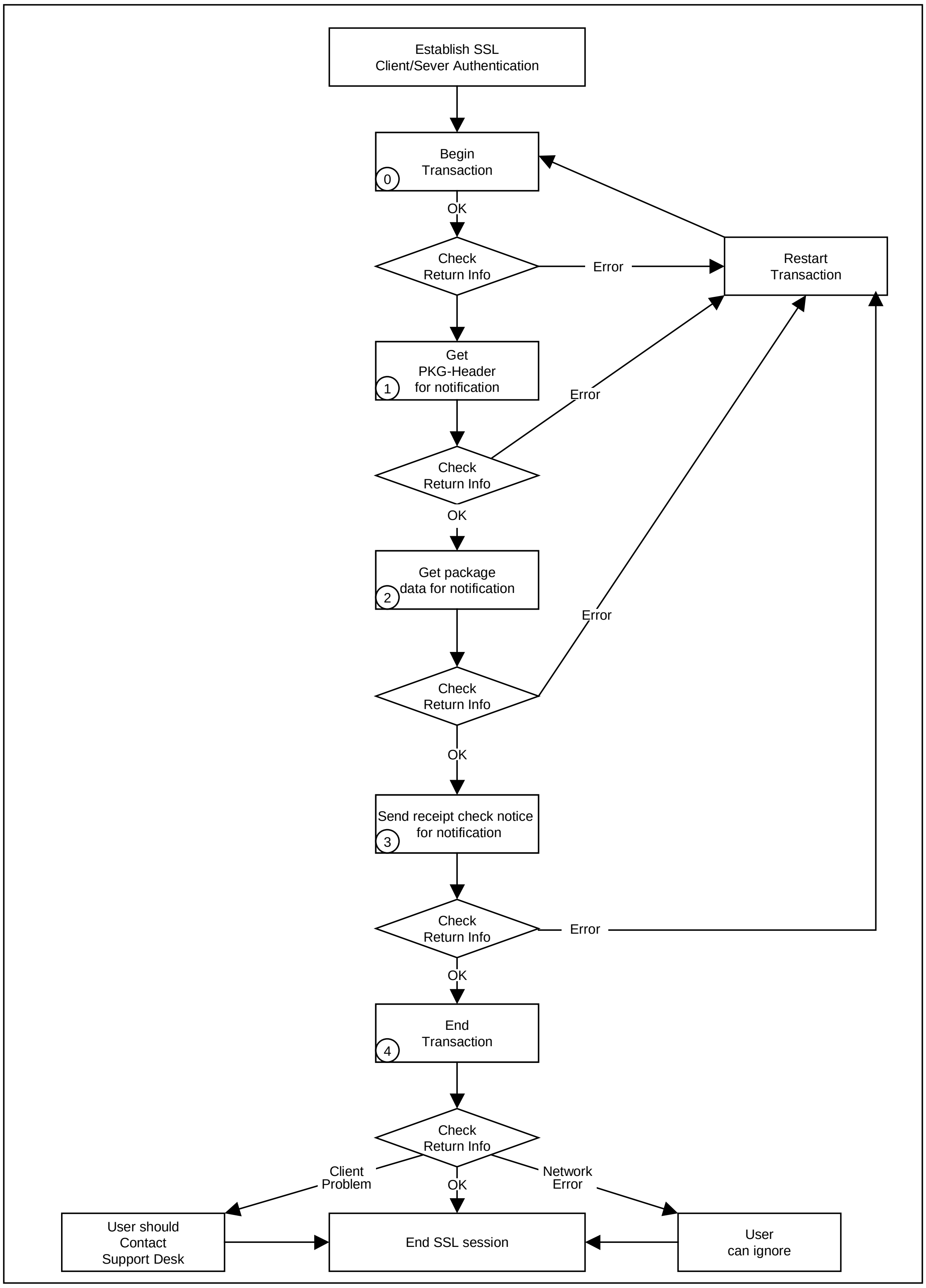


Figure 6 - Application level protocol behavior for notification

5.1.4 Application Layer Protocol for Dispatch list

At the highest level for dispatch list, there are five events that the protocol requires a client and server to support. These events are:
Each office retains the right to choose, based on its free decision, whether to use or not to use the Application Layer Protocol for dispatch list.

(a) Begin Transaction
(b) Get Package Header for dispatch list
(c) Get Package Data for dispatch list
(d) Send Receipt Check Notice for dispatch list
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package header contains the information for sending a dispatch list. It is a WASP containing the package header in XML format.
(iii) The package data contains a dispatch list, which contains a dispatch number.

5.1.4.1 Use of the SSL Tunnel for Dispatch list

Refer to Section 5.1.2.1, "Use of the SSL Tunnel for application."

5.1.1.2 Application level Events for Dispatch list

Start SSL session (See Figure 7)

Step 0: Begin Transaction
Client action:
Get transaction Information.
Server response:
Return values in the *transaction_id and max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in sending a dispatch list.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Get Package Header for dispatch list
Client action:
Send package header Request dispatch-list sending.
Server response:
a) OK The response contains the package header, which contains the information for sending a dispatch list.
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance, due to a signature verification error or validation data expiration), the application response code (ARC) value is set to FFF6.

If the number of sendable dispatch-lists in package header of Server response is "0(zero)" (there is no sendable dispatch-lists), then go to Step 4.

Step 2: Get Package Data for dispatch list

Client action:

Send package-data request.

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 3: Send Receipt Check Notice for dispatch list

Client action:

Send Receipt Check Notice for dispatchlist

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK

b) Error (Client can ignore this response)

Close SSL session

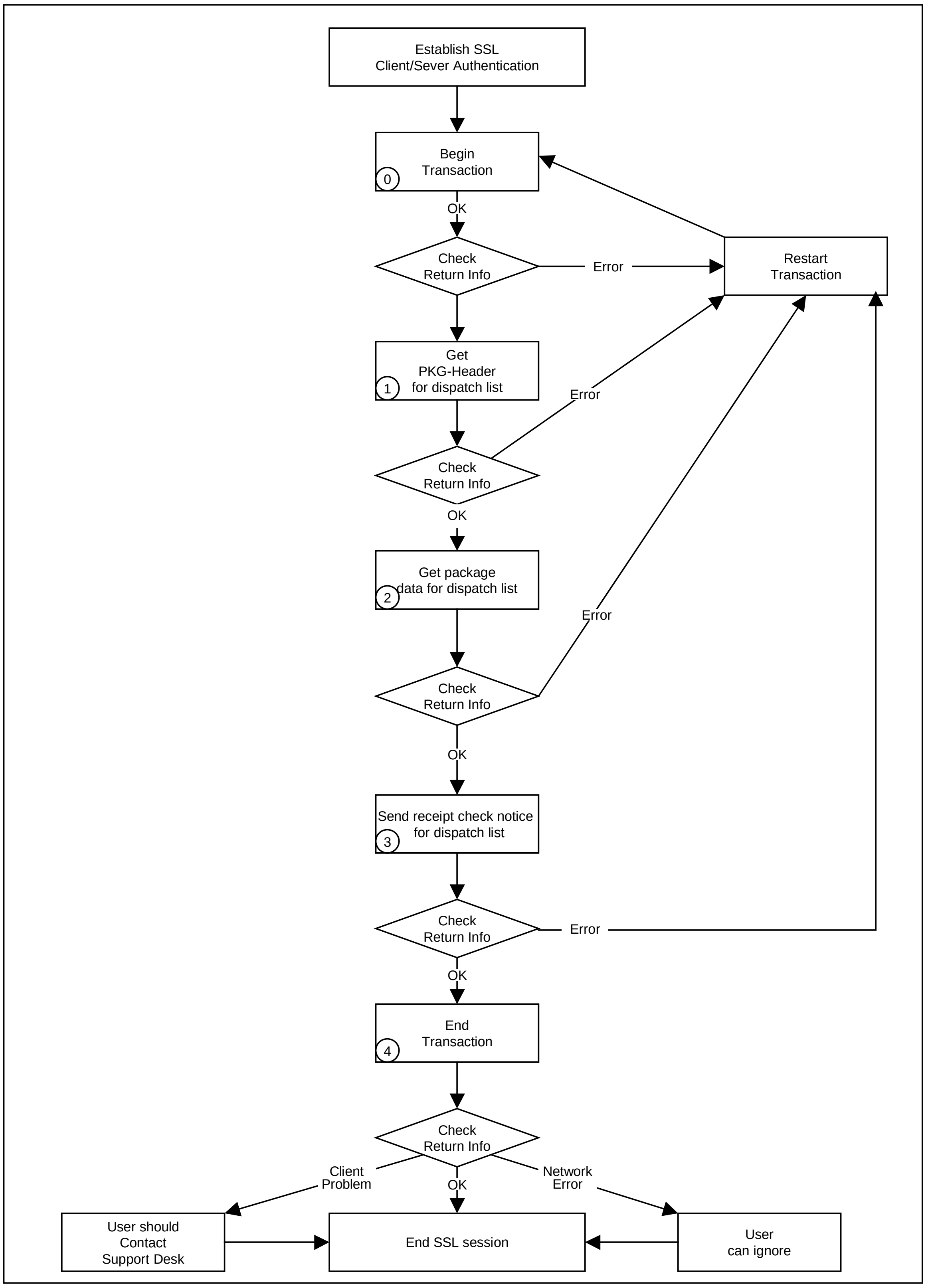


Figure 7 - Application level protocol behavior for dispatch list

5.1.5 Application Layer Protocol for Application receipt list

At the highest level for application receipt list, there are five events that the protocol requires a client and server to support. These events are:
Each office retains the right to choose, based on its free decision, whether to use or not to use the Application Layer Protocol for application receipt list.

(a) Begin Transaction
(b) Get Package Header for application receipt list
(c) Get Package Data for application receipt list
(d) Send Receipt Check Notice for application receipt list
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package header contains the information for sending an application receipt list. It is a WASP containing the package header in XML format.
(iii) The package data contains an application receipt list, which contains an application number.

5.1.5.1 Use of the SSL Tunnel for Application receipt list

Refer to Section 5.1.2.1, "Use of the SSL Tunnel for application."

5.1.1.2 Application level Events for Application receipt list

Start SSL session (See Figure 8)

Step 0: Begin Transaction
Client action:
Get transaction Information.
Server response:
Return values in the *transaction_id and max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in sending an application receipt list.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Get Package Header for application receipt list
Client action:
Send package header Request application receipt list sending.
Server response:
a) OK The response contains the package header, which contains application receipt list summary information.
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance, due to a signature verification error or validation data expiration), the application response code (ARC) value is set to FFF6.

If the number of sendable application receipt list in package header of Server response is “0(zero)” (there is no sendable application receipt list), then go to Step 4.

Step 2: Get Package Data for application receipt list

Client action:

Send package-data request.

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 3: Send Receipt Check Notice for application receipt list

Client action:

Send Receipt Check Notice for application receipt list

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK

b) Error (Client can ignore this response)

Close SSL session

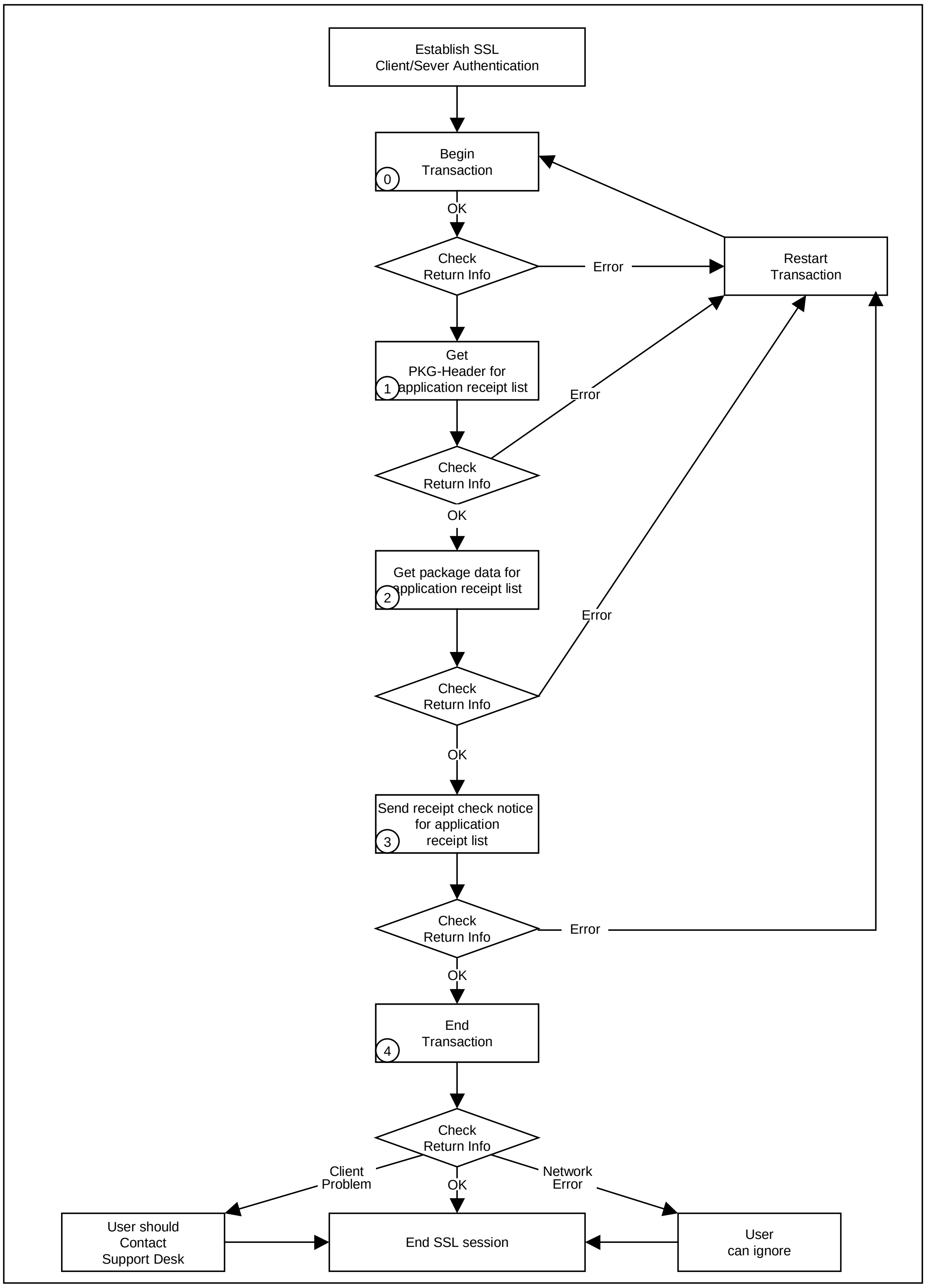


Figure 8 - Application level protocol behavior for application receipt list

5.1.6 Transaction Management Header Elements

The following items, which are all fixed length, are included in all post and response messages:

Attrib. Name	division_hash
Values	ASCII Hexadecimal representation of 160-bit hash value
Length	40 bytes (40 x 8bit characters)
Description	SHA-1 Hash of the current division.

Attrib. Name	protocol_version
Values	Unique
Length	4 bytes (4 x 8bit ASCII char)
Description	A unique identifier for the version of the protocol used to create the transaction data (e.g. 0100 for Version 1.0) First two bytes are for the major version number and last two for the release within this version.

Attrib. Name	transaction_type	
Values	pbeg, ebeg,	
	pend, eend	
	ehdr, phdr,	
	edat, pdat,	
	erct, prct,	
	ephn, pphn	package header for notification
	epdn, ppdn	package data for notification
	ercn, prcn	receipt check notice for notification
	ephd, pphd	package header for dispatch list
	epdd, ppdd	package data for dispatch list
	ercd, prcd	receipt check notice for dispatch list
	epha, ppha	package header for application receipt list
	epda, ppda	package data for application receipt list
	erca, prca	receipt check notice for application receipt list
	ASCII lower case 7-bit ISO 646 e-stands for encrypted, p-plain text	
Length	4 bytes	
Description	Attrib. of the transaction header that identifies the nature of the data transmitted	

Attrib. Name	transaction_id
Values	Unique
Length	36 bytes
Description	A unique identifier assigned by the server associating all transactions involved in the submission of an application. For Begin Transaction this is blank (ASCII x'20').

Attrib. Name	reserved_use
Values	Reserved for domestic use
Length	32 bytes
Description	This data element is for JPO specific use.

Attrib. Name	total_bytes
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The total size in bytes of the object being sent (WASP containing the Package Header, WASP containing the package data, and the WASP containing the receipt).

Attrib. Name	division_size
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The size in bytes of the data component (chunk) of the object being transferred.

Attrib. Name	division_offset
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	• Value representing the starting position of the data within the object being transferred. • Division_offset starts at 0.

Attrib. Name	division_response_code
Values	

Division RCs	*Meaning*
0000	OK
FFFF	General Error
FFFE	Resend Last
FFFD	Wait
FFFC	Protocol Sequence Error

ASCII 4 x 8bit char

Length	4 bytes
Description	Server or client return code used to manage the division mechanism

Attrib. Name	application_response_code
Values	

Application RCs	*Meaning*
0000	OK
FFFF	General Error
0001	OK, Package Known
0002	OK, New Package
1000	Pending
FFFB	Client Problem
FFFA	Network Error
FFF9	Protocol Version Error
FFF8	Hash Value of “division hash” in the Transaction Management Header is erroneous.
FFF7	The hash values in package header and the WAD of package data do not match.
FFF6	The signature is invalid (for instance, due to a signature verification error or validation data expiration).[4]

ASCII 4 x 8bit char

Length	4 bytes
Description	Server or client return code used to manage the application level events

Attrib. Name	encoding_method
Values	

Application RC	*Meaning*

[4] This code is applied when the server cannot verify the authentication in Get package header for notification, Get package header for dispatch list, or Get package header for application receipt list.

	UTF8 — UNICODE UTF8 SJIS — UNICODE Shift-JIS ASCII 4 x 8bit char
Length	4 bytes
Description	Encoding scheme for error message translation.

Attrib. Name	error_message
Values	UNICODE UTF8, UNICODE Shift-JIS
Length	256 bytes (256 x 8bits)
Description	Optional text explaining the reason for error response codes. If an error message is needed for both division and application response codes, these should be concatenated. Each server will choose one of the specified encoding schemes to translate the error message into human readable format.

5.1.6.1 Transaction Management Data elements

Attrib. Name	max_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Maximum bytes allowed for a division.
Example	0000000000008192 (8Kbytes)

5.1.6.2 Server parameters

Attrib. Name	Server_timeout
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	Time in seconds before the server can assume that a client has lost its network connection and the transaction can be abandoned.
Example	0000000000000120 (2 minutes)

Note that the value for the server_timeout at the protocol level is set at the discretion of the individual Office.

5.1.6.3 Client parameters

Attrib. Name	Client_preferred_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Preferred number of bytes to be used for a division.
Example	0000000000004096 (8k)

Attrib. Name	Client_retry_limit
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Number of times the client should resend a division before abandoning the transaction
Example	0000000000000005 (5 retries)

Attrib. Name	Client_retry_wait
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The time in seconds the client should wait before issuing a retry
Example	0000000000000005 (5 secs)

Note that the value for the client_retry_wait at the protocol level is set at the application level.

5.1.7 Division mechanism

When sending data between the client and server, this data is divided into manageable chunks which, together with a transaction management header, are called divisions. Under the control of the client, the size of these divisions can vary during the life of the transactions. This provides a pacing mechanism that can be used to overcome Internet transmission problems.

The initial size of the division data message is set to the smallest of either:

(a) max_division_size returned by the server as a response to the Begin Transaction Request
(b) client_preferred_division_size set in the startup parameters of the client

The client builds one or more divisions made up of the transmission management header and a data message. As each division is sent to the server, the server checks for completeness of the transmission by calculating the hash value of the division.

5.1.7.1 Calculating the division hash value

The hash is calculated on the basis of all fields in the header as well as any data message. The hash, which is calculated using the SHA-1 algorithm, is placed as the first element of each division.

It has been agreed before the server rejects a package as invalid, it should check the version of the protocol before checking the hash value in case a future version of the protocol should adopt a different hash algorithm.

The following fields of the HTTP Post or response message are therefore included in the hash calculation:

5.1.7.1	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application rc*	*Encoding method*	*error message*	*data message*
Length	4	4	36	32	16	16	16	4	4	4	256	???

5.1.8 Event Level Protocol

Transactions described in this section are further illustrated in Figure 9, Figure 10, Figure 11, Figure 12, Figure 13 and Figure 14 below.

5.1.8.1 Begin Transaction

The Begin Transaction post message submitted by the client contains the highest protocol version supported by the client. If the server supports the version provided by the client, it should communicate with the client in accordance with the rules for that version of the protocol and use that version number in all response messages. If the server cannot support the protocol version specified by the client, the application response code should indicate protocol version error, and the version number specified in the response message should be the highest protocol version supported by the server. The server should support earlier versions.

Post Message

5.1.8.1	*Division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*

Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pbeg	Blank	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*Transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	16
Value	X	0100	pbeg	New id	???	16	16	0	0	0	???	???	???

Data Message: max_division_size (16 bytes)

5.1.8.2 Send Package Header

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	phdr	tranid	???	X	Y	Z	0	0	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	phdr	tranid	???	0	0	0	a	b	???	blank	None

5.1.8.3 Send Package Data

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pdat	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WASP containing package data

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pdat	tranid	???	0	0	0	a	b	???	blank	None

5.1.8.4 Get Receipt

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prct	tranid	???	0	0	0	???	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	prct	tranid	???	x	y	z	???	0	???	blank	Receipt

Data Message: WASP containing receipt

5.1.8.5 End Transaction

Post Message

Name	*division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pend	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pend	tranid	???	0	0	0	a	b	???	???	None

5.1.8.6 Get Package Header for notification

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pphn	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pphn	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.8.7 Get Package Data for notification

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppdn	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppdn	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WASP containing package data

5.1.8.8 Send Receipt Check Notice for notification

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcn	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcn	tranid	???	0	0	0	a	b	???	blank	None

5.1.8.9 Get Package Header for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pphd	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pphd	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.8.10 Get Package Data for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppdd	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppdd	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WASP containing package data

5.1.8.11 Send Receipt Check Notice for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*

Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcd	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcd	tranid	???	0	0	0	a	b	???	blank	None

5.1.8.12 Get Package Header for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppha	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppha	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.8.13 Get Package Data for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppda	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppda	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WASP containing package data

5.1.8.14 Send Receipt Check Notice for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prca	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*

Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prca	tranid	???	0	0	0	a	b	???	blank	None

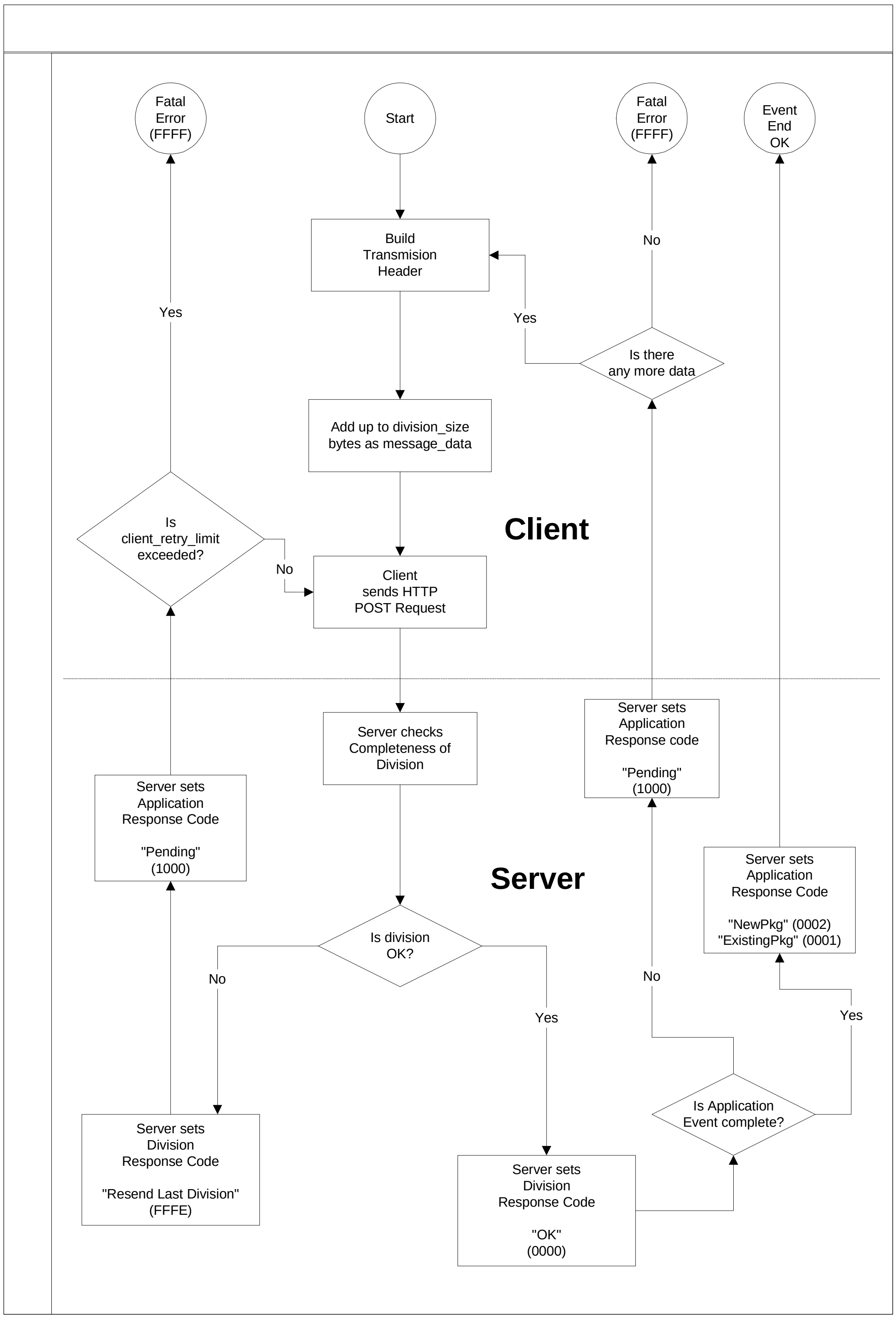

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Event End OK
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "NewPkg" (0002) "ExistingPkg" (0001)
Is division OK?
No
No
Yes
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 9 - Send package header behavior

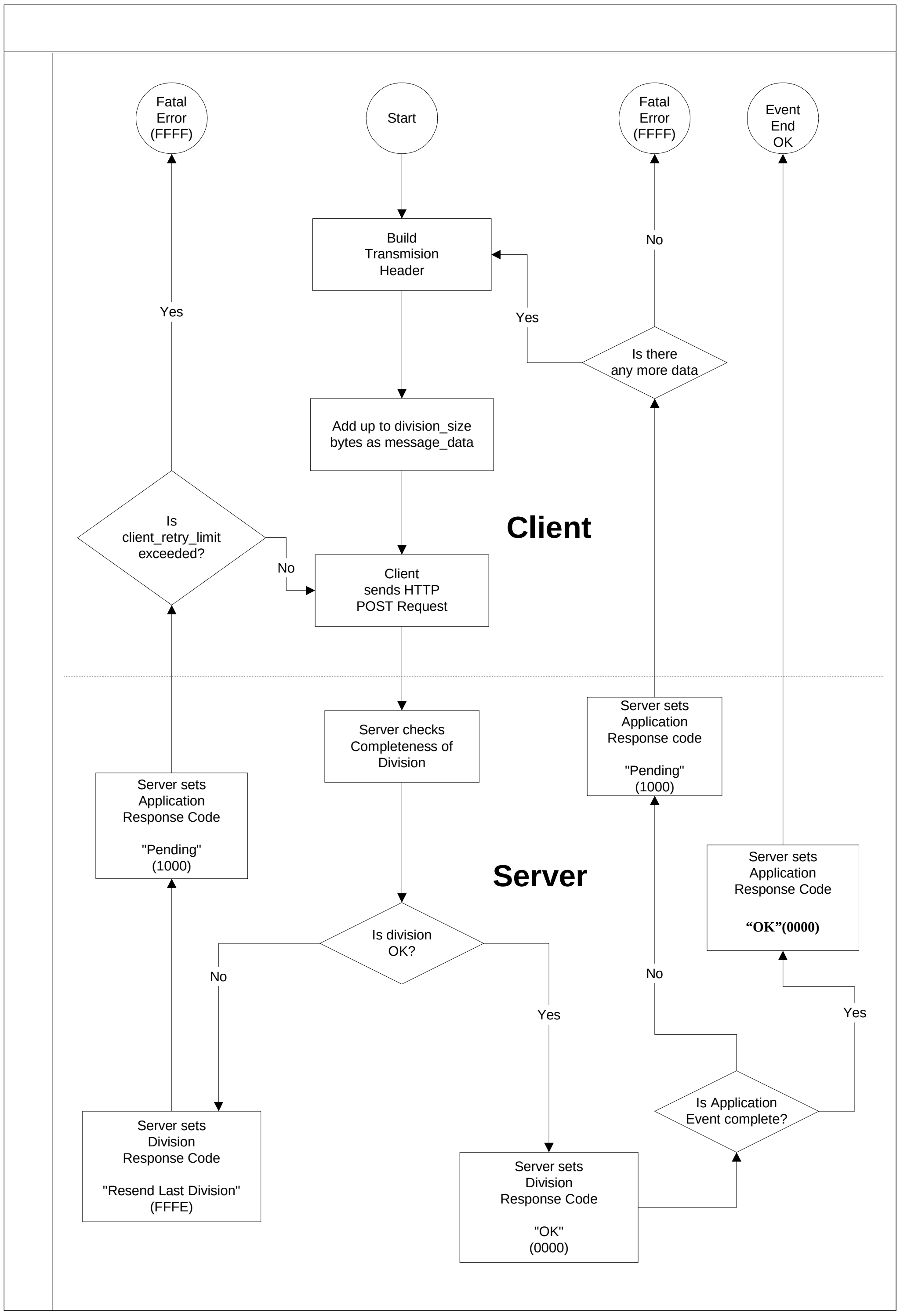

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Event End OK
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "OK"(0000)
Is division OK?
No
No
Yes
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 10 - Send package data behavior

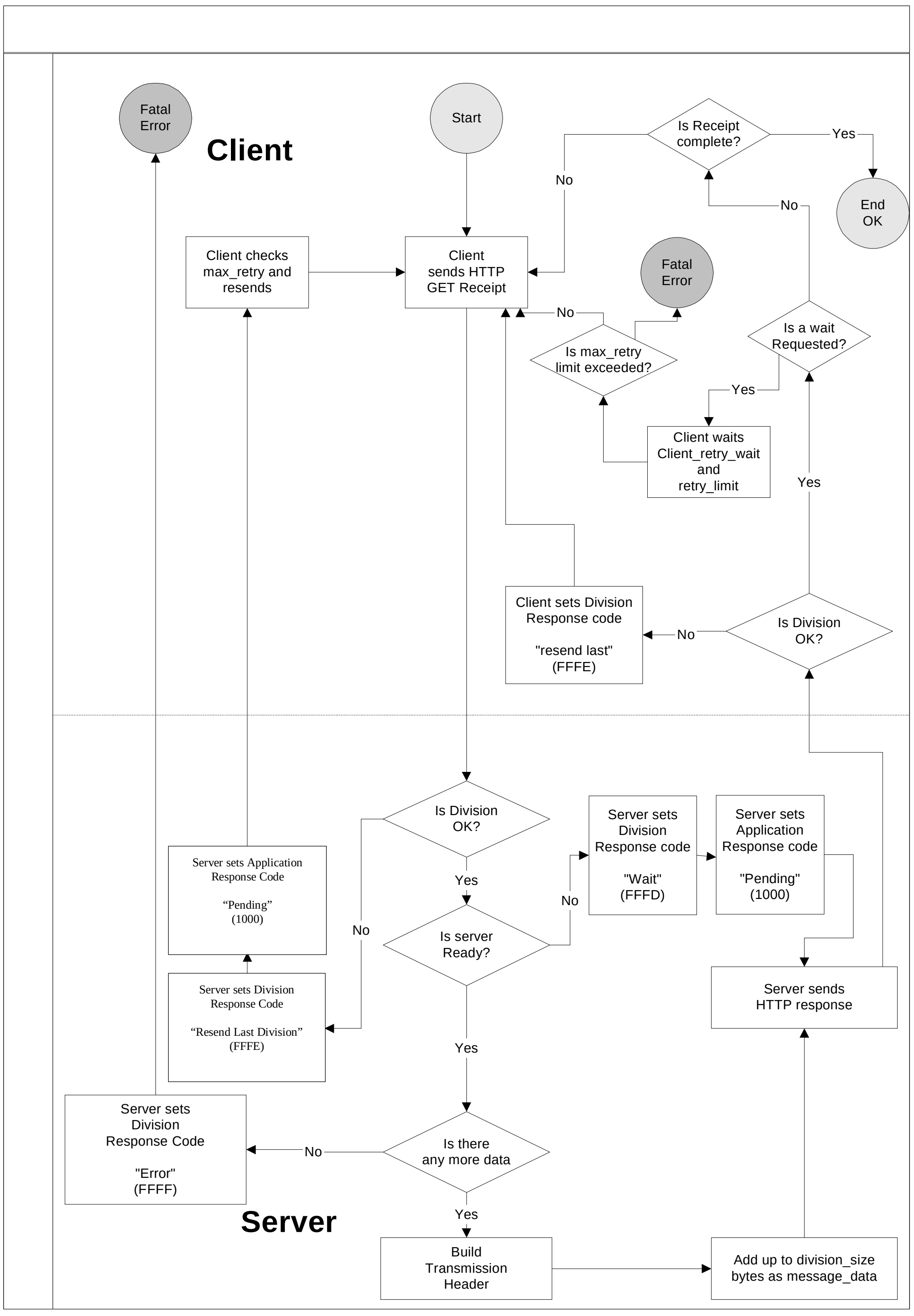

Fatal Error
Start
Client
Is Receipt complete?
Yes
No
End OK
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is max_retry limit exceeded?
Is a wait Requested?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Yes
Server sets Division Response code "Wait" (FFFD)
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
No
Is server Ready?
No
Server sends HTTP response
Server sets Division Response Code "Resend Last Division" (FFFE)
Yes
Server sets Division Response Code "Error" (FFFF)
No
Is there any more data
Yes
Server
Build Transmission Header
Add up to division_size bytes as message_data

Figure 11 - Get receipt behavior

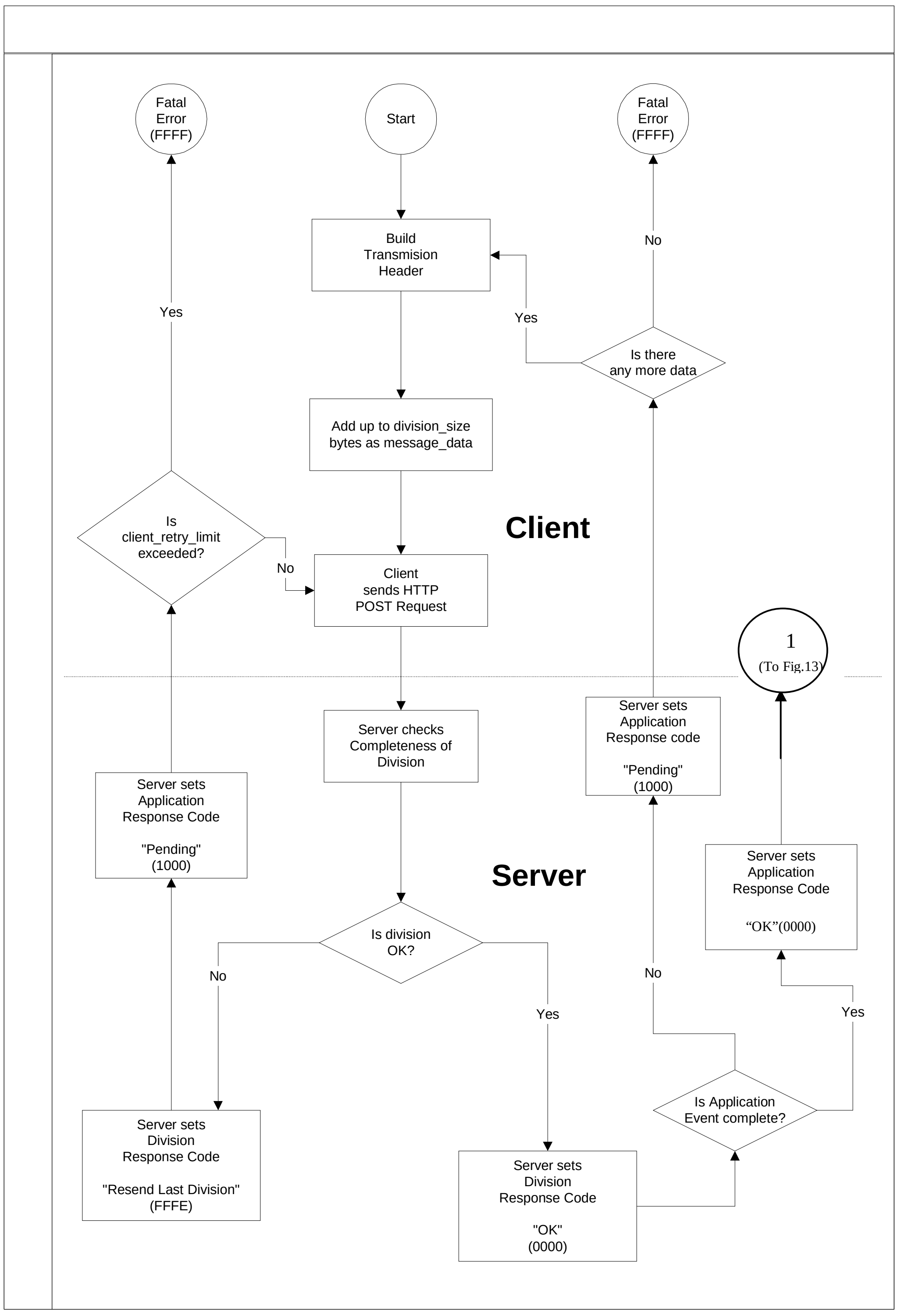

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
1 (To Fig.13)
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "OK"(0000)
Is division OK?
No
Yes
No
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 12 - Get package header behavior for notification <upstream>

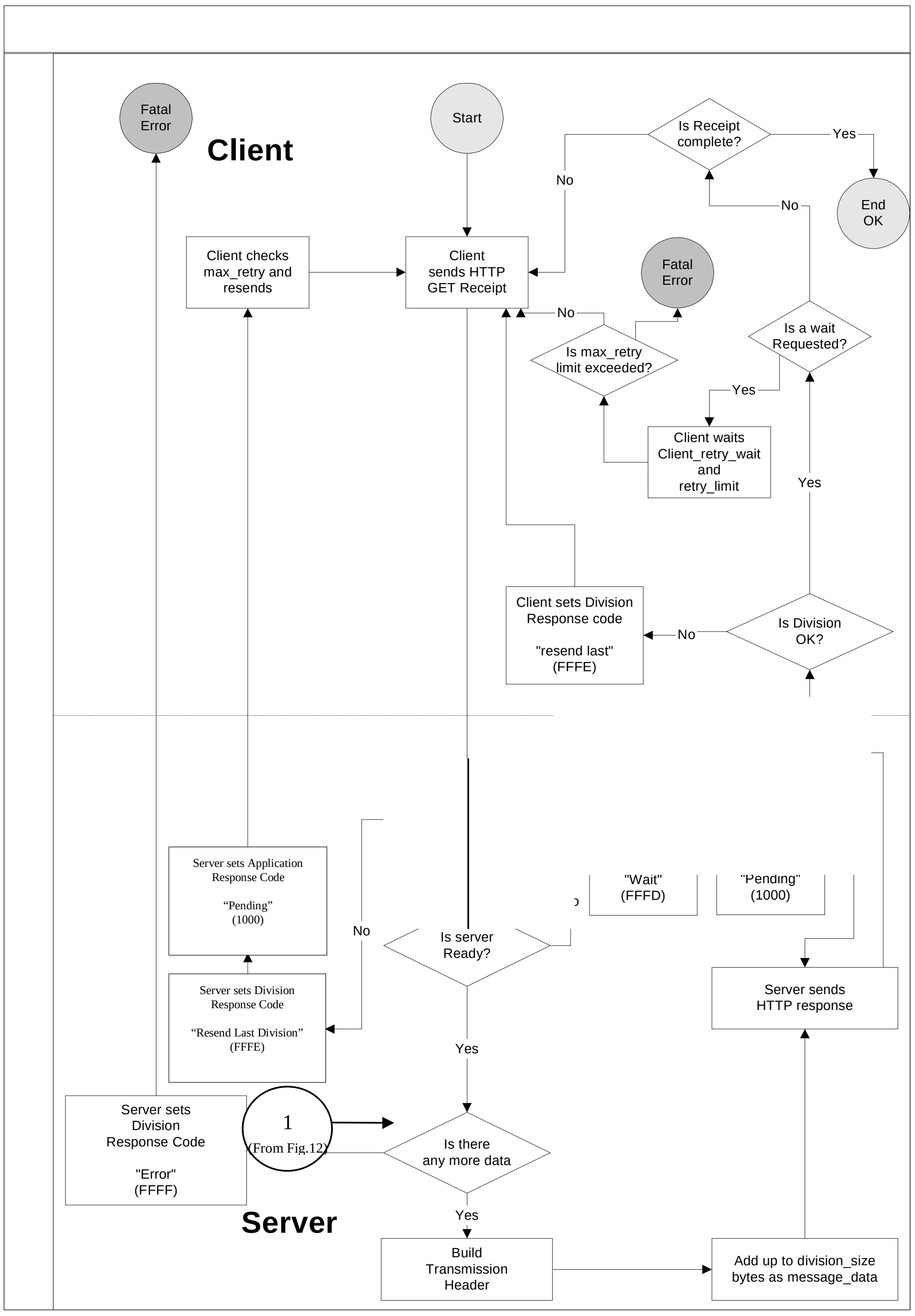

Fatal Error
Client
Start
Is Receipt complete?
Yes
No
End OK
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is max_retry limit exceeded?
Is a wait Requested?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
"Wait" (FFFD)
"Pending" (1000)
Server sets Application Response Code "Pending" (1000)
No
Is server Ready?
Server sends HTTP response
Server sets Division Response Code "Resend Last Division" (FFFE)
Yes
Server sets Division Response Code "Error" (FFFF)
1 (From Fig.12)
Is there any more data
Server
Yes
Build Transmission Header
Add up to division_size bytes as message_data

Figure 13 - Get package header behavior for notification <downstream>

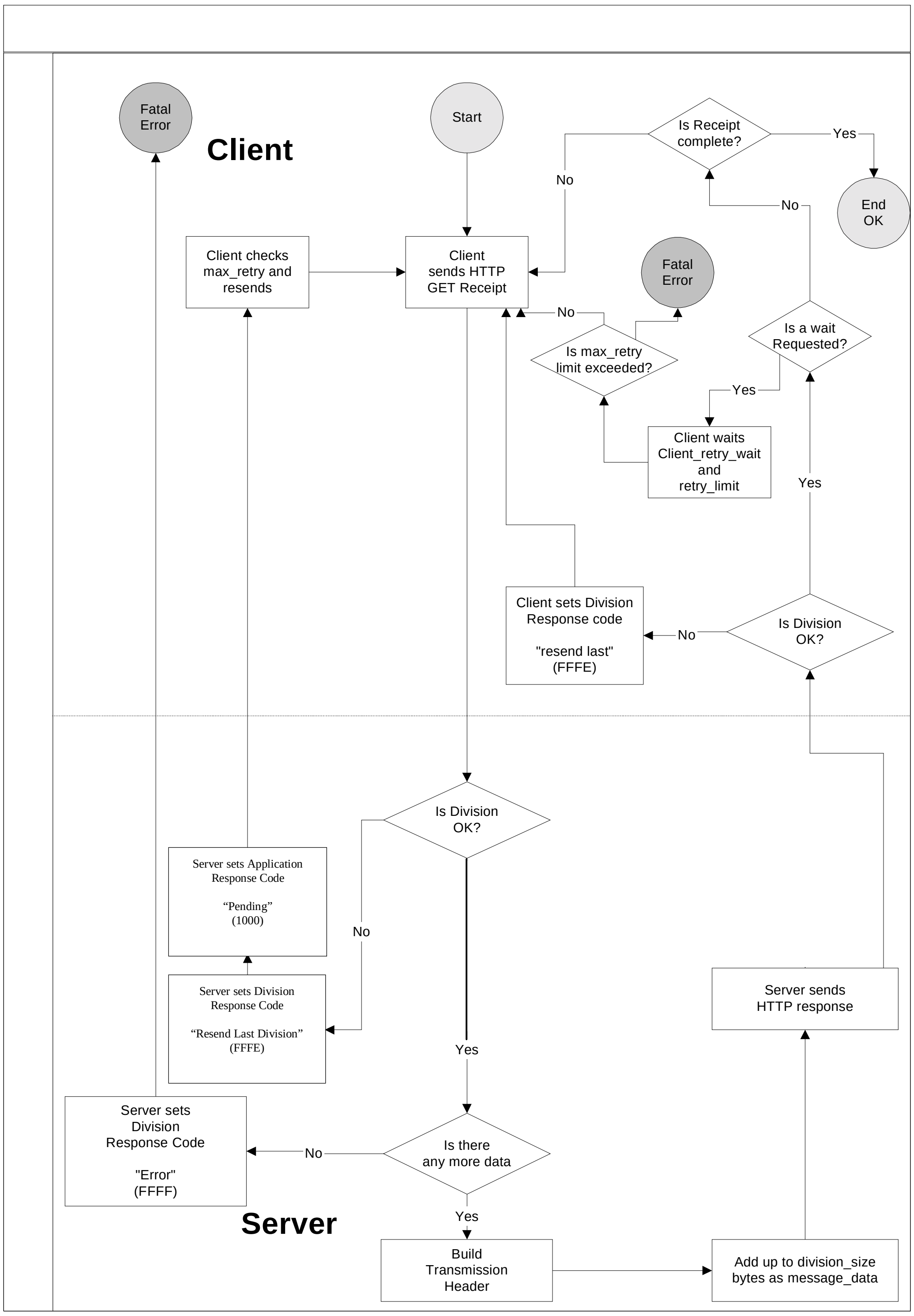

Fatal Error
Start
Client
Is Receipt complete?
Yes
No
No
End OK
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is a wait Requested?
Is max_retry limit exceeded?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Server sets Application Response Code "Pending" (1000)
No
Server sends HTTP response
Server sets Division Response Code "Resend Last Division" (FFFE)
Yes
Server sets Division Response Code "Error" (FFFF)
No
Is there any more data
Server
Yes
Build Transmission Header
Add up to division_size bytes as message_data

Figure 14 - Get package data behavior for notification

5.2 Package/transmission combinations

This section describes the permissible package/transmission combinations, for each of the PCT communications sectors. This standard does not preclude provision of publicly available information by means other than those covered in the standard. Further packaging types (e.g., web-based document delivery) and package/transmission options may become available in the future.

5.1.9 Applicant-Office (international phase) sector

IA documents may be filed by on-line means (using PKI) over the public Internet, over a private network, or transmitted off-line (using PKI or non-PKI) on physical media. The option of on-line filing of an IA utilizing a non-PKI method is not presently permitted,11 except under possible transitional reservations permitted by AIs Section 703(f) (see AI section 7.1.1 as to the consequences of non-PKI filing under such a transitional reservation).

Figure 15 shows a matrix of the various submission mechanism/packaging combinations that are permissible in the Applicant-Office (international phase) sector as specified under this standard. In summary, for each submission mechanism:

(a) On-line/Internet: The SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(b) On-line/secure: The SEP or WASP must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network; 2) the Internet using channel level encryption (e.g. SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(c) Off-line/physical media: either the SEP, WASP, or WAD package must be used. Physical media (e.g., diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

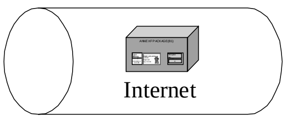
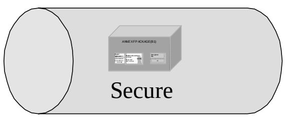
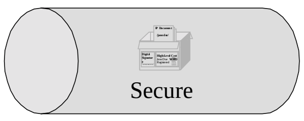
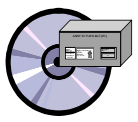
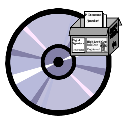
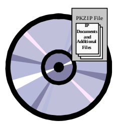
	Signed and Encrypted Package	Wrapped and Signed Package	Wrapped Application Documents
On-line / Internet		🚫 Not permissible	🚫 Not permissible
On-line / secure			🚫 Not permissible
Off-line / media			

Figure 15 –Package/transmission combinations permitted in the Applicant-Office (international phase) sector

5.1.10 Office-Office sector

All Office-Office sector data exchange must be conducted utilizing PKI-based data exchange. IA documents may be exchanged by on-line means over the public Internet or over a private network (such as Tri-Net or WIPONET), or transported on physical media.
Figure 16 shows a matrix of the various submission mechanism/packaging combinations that are permissible as specified under this standard. In summary, for each data exchange mechanism:

(d) On-line/Internet: The SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(e) On-line/secure: The SEP or WASP must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network (e.g. WIPONET, Tri-Net); 2) the Internet using channel level encryption (e.g. SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(f) Off-line/physical media: The SEP or WASP must be used. Physical media (e.g. diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

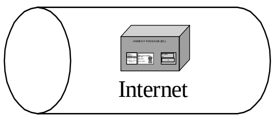
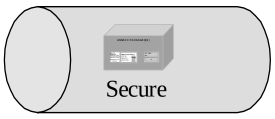
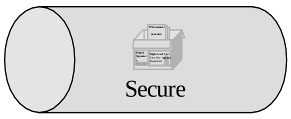
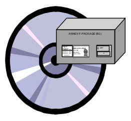
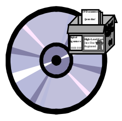
	Signed and Encrypted Package	Wrapped and Signed Package	Wrapped Application Documents
On-line / Internet		🚫 Not permissible	🚫 Not permissible
On-line / secure			🚫 Not permissible
Off-line / media			🚫 Not permissible

Figure 16 – Package/transmission combinations permitted in Office-Office sector

5.1.11 Designated Office sector

The SEP, WASP, or WAD package may be used when exchanging IA documents under the designated Office sector. Figure 17 shows a matrix of the various submission mechanism/packaging combinations that are permissible. In summary, for each data exchange mechanism:

(g) On-line/Internet: the SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(h) On-line/secure: the SEP, WASP, or WAD must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network (e.g. WIPONET, Tri-Net); 2) the Internet using channel level encryption (e.g., SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(i) Off-line/physical media: either the SEP, WASP, or WAD package must be used. Physical media (e.g., diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

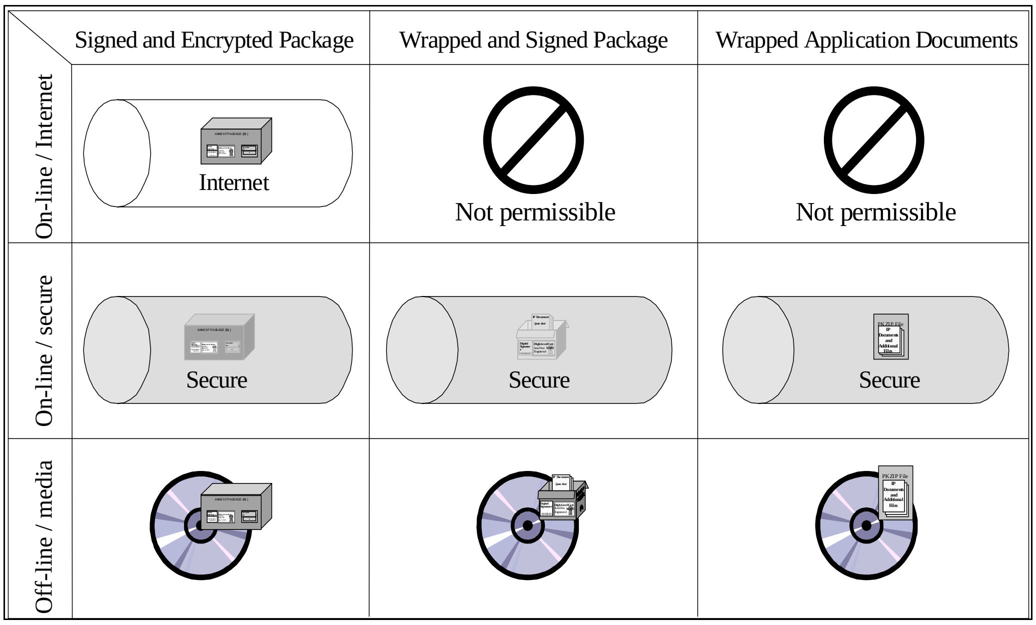


***Figure 17* - Package/transmission combinations permitted in designated Office sector**

[Annex 2 follows/
L'annexe 2 suit]

WIPO CIRCULAR C. PCT 830

April 4, 2002

Madam,
Sir,

Standard for the electronic filing and processing of international applications: Proposals for change

Proposed modifications of the Administrative Instructions under the PCT

1. This circular concerns proposals for change to the standard for the electronic filing and processing of international applications under the Patent Cooperation Treaty (PCT) which will, if adopted, result in modifications of Part 7 and Annex F of the Administrative Instructions under the PCT. It is being sent, for the purposes of consultation pursuant to Rule 89.2(b) of the Regulations under the PCT, to the national and regional patent Offices of, or acting for, all PCT Contracting States, including all receiving Offices, International Searching Authorities, International Preliminary Examining Authorities, and designated and elected Offices, and to interested intergovernmental organizations, as well as to certain non-governmental organizations representing users of the PCT system.

2. The International Bureau has received a number of proposals for change to the standard for the electronic filing and processing of international applications which, if adopted, would result in modifications of Annex F of the Administrative Instructions under the PCT, together with related proposals for amendment to certain provisions of Part 7 of the Administrative Instructions.

3. The proposals are available on WIPO's Web site (see http://www.wipo.int/pct/efiling_standard/en) as annexes to the following "Proposal for Change" (PFC) files:

– PCT/EF/PFC 02/001 relating to Part 7, Section 709 "Means of Communication" (proposal by the Japan Patent Office);

– PCT/EF/PFC 02/002 relating to Annex F, section 4 "IA Documents Packaging" (proposal by the Japan Patent Office);

– PCT/EF/PFC 02/003 relating to Annex F, section 5 "Transmission" (proposal by the Japan Patent Office);

– PCT/EF/PFC 02/004 relating to Annex F, Appendix I, section 4.1 "Package header" (proposal by the Japan Patent Office);

– PCT/EF/PFC 02/005 relating to Annex F, Appendix I, section 3 “DTDs for new PCT applications (E-PCT)” (proposal by the United States Patent and Trademark Office);

– PCT/EF/PFC 02/006 relating to Annex F would add a recommendation on a file naming convention (proposal by the United States Patent and Trademark Office);

– PCT/EF/PFC 02/007 relating to Annex F, Appendix I, section 4.1 “Package header” (proposal by the United States Patent and Trademark Office);

– PCT/EF/PFC 02/008 relating to Part 7, Section 703 “Filing Requirements; Basic Common Standard” (proposal by the United States Patent and Trademark Office).

4. The International Bureau would appreciate receiving any comments on the proposed modifications by May 15, 2002. Comments should preferably be made by e-mail via the “Submit comments” facility on the Web site, but may alternatively be made in writing by letter or by fax to +41 (22) 338 8040. Comments will be included as Annexes to the relevant PFCs. Copies of the proposals on paper are available from the International Bureau on request. Inquiries concerning the proposals should be directed to Mr. Thiam Ming Song (e-mail: song@wipo.int; phone: 41 (22) 338 9640; fax: 41 (22) 338 8040).

5. In Circular C. PCT 827, which is also dated April 4, 2002, the International Bureau proposes other modifications of the Administrative Instructions which would add to Annex F a new section 2.5 “Change procedure” dealing with special procedures for the proposal, consideration and implementation of future modifications of the standard for electronic filing and processing of international applications. It is intended that consideration of the proposals referred to in the present Circular will proceed independently of those other proposals.

6. The International Bureau intends to promulgate by June 30, 2002, the modifications proposed in the present Circular, revised if necessary in the light of comments received. It is envisaged that the modifications would enter into force on January 1, 2003.

Sincerely yours,

Francis Gurry
Assistant Director General

OMPI CIRCULAIRE C. PCT 830

Le 4 avril, 2002

Madame,
Monsieur,

Norme concernant le dépôt et le traitement électroniques des demandes internationales : propositions de modification

Propositions de modification des instructions administratives selon le PCT

1. La présente circulaire concerne des propositions de modification de la norme concernant le dépôt et le traitement des demandes internationales selon le Traité de coopération en matière de brevets (PCT) qui, si elles étaient adoptées, aboutiraient à la modification de la septième partie et de l’annexe F des instructions administratives du PCT. Elle est adressée, aux fins de la consultation prévue à la règle 89.2.b) du règlement d’exécution du PCT, aux offices de brevets nationaux et régionaux de tous les États contractants du PCT ou agissant pour eux, notamment l’ensemble des offices récepteurs, des administrations chargées de la recherche internationale, des administrations chargées de l’examen préliminaire international et des offices désignés et élus, ainsi qu’aux organisations intergouvernementales intéressées et à certaines organisations non gouvernementales représentant les utilisateurs du système du PCT.

2. Le Bureau international a reçu un certain nombre de propositions de modification de la norme concernant le dépôt et le traitement électroniques des demandes internationales qui, si elles étaient adoptées, entraîneraient des modifications de l’annexe F des instructions administratives du PCT, ainsi qu’un certain nombre de propositions concernant les changements correspondants à apporter à certaines dispositions de la septième partie des instructions administratives.

3. Ces propositions peuvent être consultées sur le site Web de l’OMPI (voir http://www.wipo.int/pct/efiling_standard/fr) en tant qu’annexes des dossiers de propositions de modification suivants :

– PCT/EF/PFC 02/001 concernant l’instruction 709 de la septième partie, intitulée “Moyens de communication” (proposition de l’Office des brevets du Japon);

– PCT/EF/PFC 02/002 concernant la section 4 de l’annexe F, intitulée “Empaquetage des documents constitutifs de demandes internationales” (proposition de l’Office des brevets du Japon);

– PCT/EF/PFC 02/003 concernant la section 5 de l’annexe F, intitulée “Transmission” (proposition de l’Office des brevets du Japon);

– PCT/EF/PFC 02/004 concernant la section 4.1 de l’appendice I de l’annexe F, intitulée “En-tête du paquet” (proposition de l’Office des brevets du Japon);

– PCT/EF/PFC 02/005 concernant la section 3 de l’appendice I de l’annexe F, intitulée “DTD pour de nouvelles demandes PCT présentées sous forme électronique” (proposition de l’Office des brevets et des marques des États-Unis d’Amérique);

– PCT/EF/PFC 02/006 concernant l’annexe F, visant à ajouter une recommandation relative à une convention de dénomination des fichiers (proposition de l’Office des brevets et des marques des États-Unis d’Amérique);

– PCT/EF/PFC 02/007 concernant la section 4.1 de l’appendice I de l’annexe F, intitulée “En-tête du paquet” (proposition de l’Office des brevets et des marques des États-Unis d’Amérique);

– PCT/EF/PFC 02/008 concernant l’instruction 703 de la septième partie, intitulée “Exigences de dépôt, norme commune de base” (proposition de l’Office des brevets et des marques des États-Unis d’Amérique).

4. Le Bureau international souhaiterait recevoir tout commentaire sur ces propositions de modification pour le 15 mai 2002 au plus tard. Les commentaires doivent de préférence être envoyés par courrier électronique grâce à la fonction “envoyer un commentaire” prévue sur le site Web, mais peuvent aussi être envoyés par courrier postal ou par télécopie (41-22-338 8040). Les commentaires éventuels seront inclus dans les annexes des différentes propositions de modification. Des copies de ces propositions sous forme imprimée sont disponibles auprès du Bureau international sur demande. Les demandes de renseignements concernant ces propositions sont à adresser à M. Thiam Ming Song (courrier électronique : song@wipo.int; téléphone : 41-22-338 9640; télécopie : 41-22-338 8040).

5. Dans la circulaire C. PCT 827, qui est aussi datée du 4 avril 2002, le Bureau international suggère d’autres modifications des instructions administratives visant à ajouter à l’annexe F une section 2.5 intitulée “Procédure de modification”, qui prévoit des procédures spéciales pour la proposition, l’examen et la mise en œuvre des futures modifications de la norme concernant le dépôt et le traitement électroniques des demandes internationales. L’examen des propositions faisant l’objet de la présente circulaire doit s’effectuer indépendamment de celui de ces autres propositions.

6. Le Bureau international a l’intention de promulguer les modifications proposées dans la présente circulaire le 30 juin 2002, après les avoir révisées si nécessaire en fonction des commentaires reçus. Ces modifications devraient entrer en vigueur le 1er janvier 2003.

Veuillez agréer, Madame, Monsieur, l’assurance de ma considération distinguée.

Francis Gurry
Sous-directeur général

[Annex 3 follows/
L’annexe 3 suit]

COMMENTS BY IP AUSTRALIA

PCT/EF/PFC 02/003

We seek further information regarding this proposed change in order to adequately assess it's impact and usefulness. Additional Application Layer Protocol's have been specified for notification, dispatch list and application receipt list. Further information on their purpose, applicable sectors and wider benefits would be useful in assessing the impact of this change.

Additional transaction types are specified for the package header and receipt check notice for each of the proposed transactions. Ideally, a standard package header and receipt check notice should be agreed for use in all or most cases.

This proposal involves changes to many aspects of the transmission standard and are complex in nature. It is also noted that the proposed change significantly alters the client and server interactions in that they require the client to specify a request for a particular transaction/transmission type. Accordingly, we would request further time to consider the PFC once the additional information is supplied.

[Annex 4 follows/
L’annexe 4 suit]

ANNEX 4/ANNEXE 4

COMMENTS BY THE JAPAN PATENT OFFICE

PCT/EF/PFC 02/003 Transmission (Annex F, section 5)

Section 5.1.8.1 Begin Transaction (Page27)
Figure 13 - Get package header behavior for notification <downstream>

[Comment]

“Figure 13 - Get package header behavior for notification <downstream> “ includes an error in writing because the part of the figure has been moved or lost.

Therefore the JPO would like to suggest the revised figure in the attached “Sheet 1”.

Sheet1

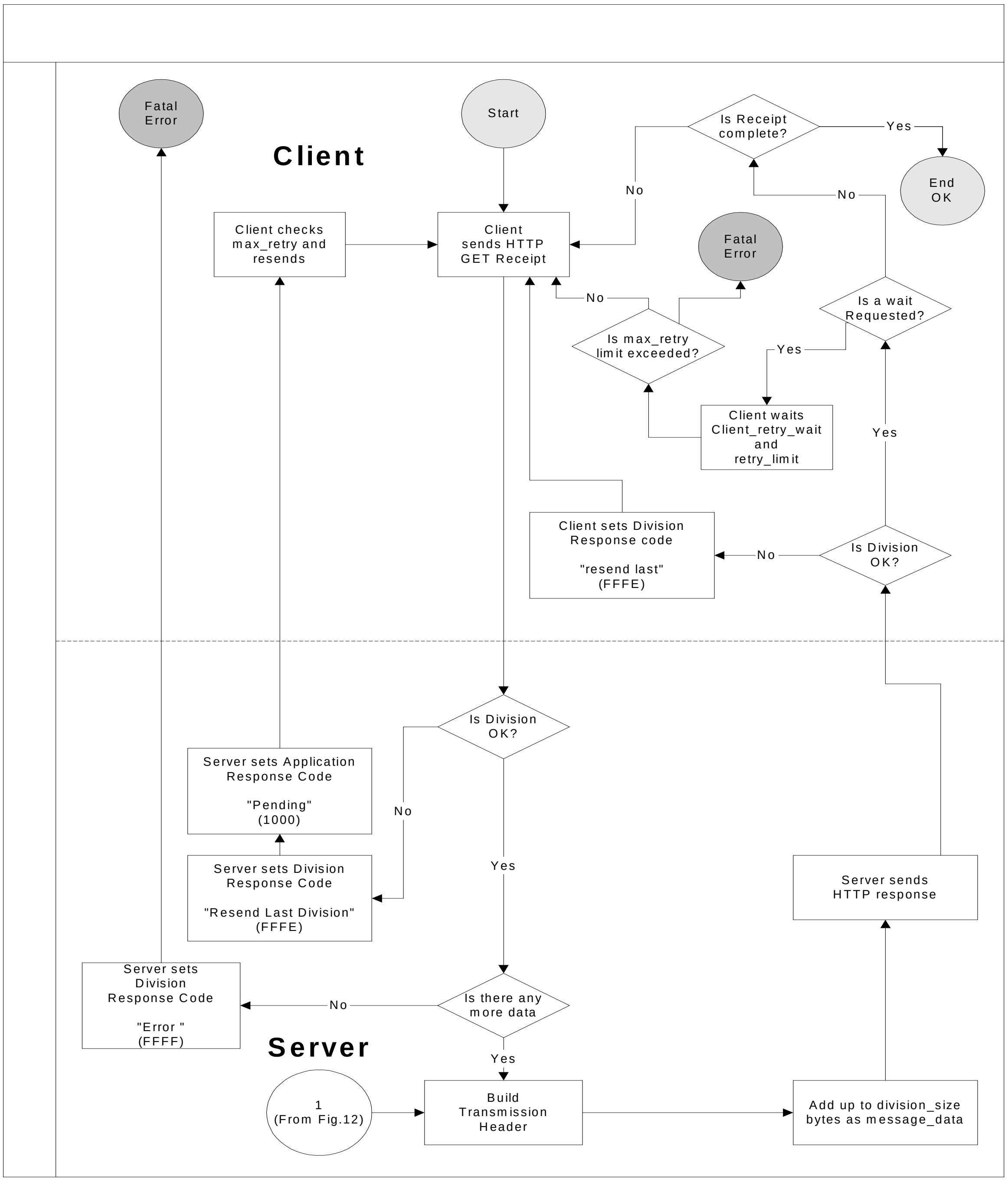

Fatal Error
Start
Client
Is Receipt complete?
Yes
End OK
No
No
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is a wait Requested?
Is max_retry limit exceeded?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Server sets Application Response Code "Pending" (1000)
No
Yes
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sends HTTP response
Server sets Division Response Code "Error " (FFFF)
No
Is there any more data
Server
Yes
1 (From Fig.12)
Build Transmission Header
Add up to division_size bytes as message_data

Figure 13: Get package header behavior for notification <downsteam>

[Annex 5 follows/
L'annexe 5 suit]

ANNEX 5/ANNEXE 5

COMMENTS BY THE UNITED STATES PATENT AND TRADEMARK OFFICE

Circular PCT/EF/PFC 02/003 (Annex F 5.1.3 through 5.1.5.2**)**

1. These provisions include references to terminology that is undefined in Annex F. For example, the meanings of the terms “Dispatch list” and “Application receipt list” are unclear. Further, the distinction between the terms “notification”, “Dispatch list”, and “Application receipt list” is unclear. More explanation in the text of Annex F would be helpful.

2. The protocols for “notification”, “Dispatch list”, and “Application receipt list” are nearly identical. It seems unnecessary to repeat this protocol almost verbatim in three different sections of Annex F.

3. Further, it is unclear what prompts the beginning of one of these transactions. The client must initiate each of these transactions yet there is no explanation of how the client becomes aware that there is a message waiting for him. How will the client know when to initiate one of these transactions? By some means to be determined, with a mechanism that requires the applicant to “pull” the notification from the Office, the Office must indicate to the applicant that a notification is available.

4. The steps detailed under the heading “Application level Events” are confusing. For example, under paragraph 5.1.3.2, Step 1, *Client action*:, the instruction to “Send package header” is confusing. The instruction would be clearer if it read “Send request for package header”. Further, the purpose of the “*Client action:*” instruction under Step 2 of paragraph 5.1.3.2 to “Send package-data request” is unclear.

5. The proposal does not indicate that an applicant may request just one, specific notification, or all pending notifications, as was previously proposed by JPO (San Francisco, 2001 Nov.).

If this proposal is accepted, then the package-header DTD requires modification to support batch office-to-applicant transmissions.

[Annex 6 follows/
L’annexe 6 suit]

ANNEX 6/ANNEXE 6

COMMENTS BY THE KOREAN INTELLECTUAL PROPERTY OFFICE

Comments on PCT/EF/PFC 02/003 relating to Annex F, Section 5 "Transmission" (proposal by the JPO)

There is a need for standards in the electronic transmission of other documents such as notifications, invitations, as well as application forms. We agree with the JPO's proposal for this.

[Annex 7 follows/
L’annexe 7 suit]

REVISED PROPOSAL BY THE JAPAN PATENT OFFICE

Proposal for revision to PFC 02/003

We would like to modify our proposal PFC-02-003. The main modification points are as follows.

Deletion of the section “5.1.4 Application Layer Protocol for Dispatch list” and “5.1.5 Application Layer Protocol for Application receipt list”.

Modification of description in the section 5.1.3.

Addition of the footnotes 11 - 14 to the section 5.1.3.

Modification of the instruction under “Client action” of step 1 and step 2 in the section 5.1.3.2.

5 TRANSMISSION

The IA package can be transmitted over secure or non-secure channels depending on the package type. This section includes the protocol to be followed as well as the package/transmission combinations that are permitted in the Applicant-Office (international phase), Office-Office, and designated Office communication sectors. While additional sectors are referred to in this standard (see section 2.3), permissible transmission/package combinations can be categorized in the three sectors listed above.

5.1 The E-filing interoperability protocol

This section describes both the transmission layer protocol between the clients and the server as well as providing a definition of the behavior required of both the client and the server.

The protocol is designed to support HTTP communication over an SSL Tunnel for all PKI based E-filing solutions and includes the following capabilities:

(a) Enables large applications to be transmitted via multiple HTTP post actions to address reliability and integrity issues
(b) Efficient error detection and correction
(c) Enables offices to control optimal transaction size

Note that this is an evolving protocol, with production systems in development at a number of IP Offices, and further revisions are foreseen.

5.1.1 Principles

The following principles have been adopted for the interoperability protocol:

(a) All communications between client and server is in the form of HTTP post actions initiated by the client
(b) All post requests and resulting responses use the same transaction management header followed by an optional data block
(c) All transmissions use the Division mechanism to divide large blobs of data into manageable chunks with a protocol that allows for retries and pacing

5.1.2 Application Layer Protocol for application

At the highest level for application, there are five events that the protocol requires a client and server to support. These events are:

(a) Begin Transaction
(b) Send Package Header
(c) Send Package Data
(d) Get Receipt
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package data contains the information for submitting application. It is a WASP consisting of various types of files.

(iii) The receipt is an acknowledgment of the submission. The content of this receipt (XML data plus an optional human readable certificate in PDF or TIFF), which is signed by the receiving Office, is defined in Annex F Appendix I. The date of receipt will be determined according to the usual principles applicable to the filing of applications on paper, including filing by electronic means (such as by facsimile transmission), that is, based on the date prevailing at the location of the Office at the time when the complete transmission of the application has been received.

5.1.2.1 Use of the SSL Tunnel for application

These events are all performed within an SSL tunnel that is established before issuing the Begin Transaction event. The SSL tunnel is built using both client and server authentication. The SSL tunnel may be stopped at the end of the transaction or, if a batch of transmissions is foreseen, the SSL tunnel can be left open and only stopped when all transmissions are complete. The SSL tunnel uses the SSL protocol version 3.0.

5.1.2.2 Application level Events for application

Start SSL session (See Figure 5)

Step 0: Begin Transaction

Client action:

Get transaction Information.

Server response:

Return values in the *transaction_id* and *max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in the submission of an application.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Send Package Header

Client action:

Send package header

Server response:

a) OK
b) Error (Abort, go back to step 0)
c) Package already received; go to step 3 to get the receipt.

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance expired), the Application Response Code (ARC) will remain OK, but the server will capture the error and provide a message on the receipt.

Step 2: Send Package Data

Client action:

Send package data

Server response:

a) OK
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package data, the server must verify the signature of the WASP and compare the message digest of the unsigned package against the message digest provided in the Package Header in Step 1 of the transaction before returning the ARC to the client. If both conditions are met, the server should return an ARC indicating OK. If the hash values in package header and the WAD of the package data do not match, the ARC value should be set to FFF7. If the signature is invalid (for instance expired), the ARC will remain OK, but the server will capture the error and provide a message on the receipt.

Step 3: Request Receipt

Client action:

Send request

Server response:

a) OK (Receipt object included in response)
b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK
b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires each individual transaction to be acknowledged by an individual receipt.

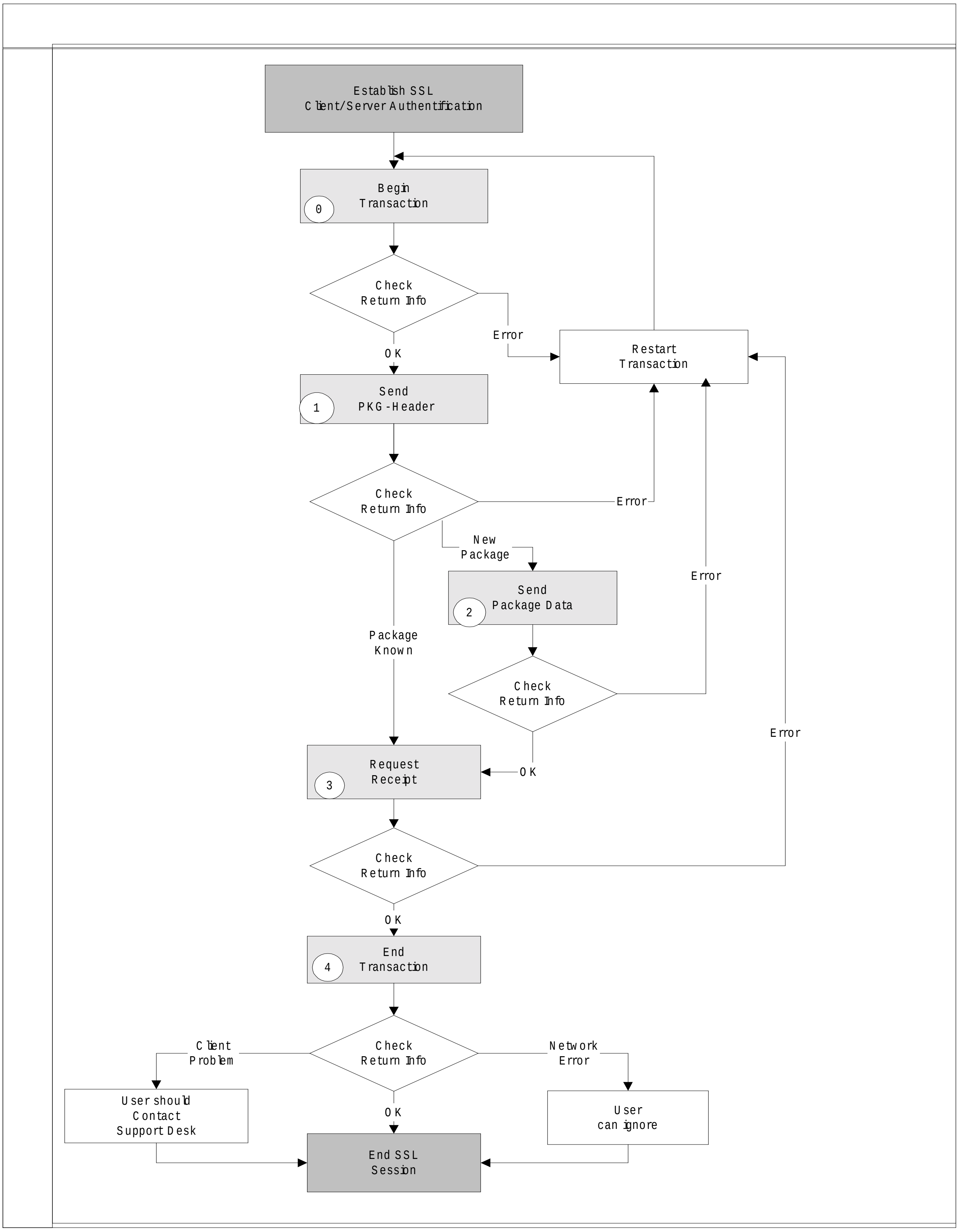


Figure 5 - Application level protocol behavior for application

5.1.3 Application Layer Protocol for notification

At the highest level for notification, there are five events that the protocol requires a client and server to support[5]. These events are:[6]

(a) Begin Transaction
(b) Get Package Header (for notification, or dispatch list, or application receipt list) [7]
(c) Get Package Data (for notification, or dispatch list, or application receipt list) [13]
(d) Send Receipt Check Notice (for notification, or dispatch list, or application receipt list)[13]
(e) End Transaction

In between the Begin and End Transactions, there are two types of WASP and one type of WAP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the request for notification. It is a WASP containing the package header in XML format. (This is applied to request to a server from a client.)
(ii) The package header contains summary information (such as a dispatch-number and the number of notifications to be sent) on the notification to be notified. It is a WASP containing the package header in XML format. (This is applied to response to a client from server.)
(iii) The package data contains the information for dispatching notification. It is the WAP which consists of one or more WASP(s).

5.1.3.1 Use of the SSL Tunnel for notification

Refer to Section 5.1.2.1, "Use of the SSL Tunnel for application."

5.1.1.2 Application level Events for notification

Start SSL session (See Figure 6)

Step 0: Begin Transaction
Client action:
Get transaction Information.
Server response:
Return values in the *transaction_id and max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in sending a notification.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

[5] The office may notify existence of notifications to an applicant before these five events.

[6] This protocol may be used for dispatching of the dispatch list and the application receipt list besides dispatching of the notification. The office can choose freely whether it will dispatch the dispatch list and the application receipt list to an applicant.
The dispatch list is the list of the dispatch numbers to the notifications which the office has dispatched to the applicant, and the application receipt list is the list of the application numbers to the application documents which the office has received from the applicant.

[7] The different value of "transaction-type" every kind of document (notification, dispatch list, application receipt list) is used because the server can identify which kind of document is request. (see the section 5.1.4 for details)

Step 1: Get Package Header

Client action:

Send request for package header (The WASP of package header for request of notification is contained.)

Server response:

a) OK (The response contains the WASP of package header containing summary information (such as dispatch-number, number-of-notification) of notifications.)[8]

b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance, due to a signature verification error or validation data expiration), the application response code (ARC) value is set to FFF6.

If the number of sendable notifications in package header of Server response is "0(zero)" (there is no sendable notifications), then go to Step 4.

Step 2: Get Package Data

Client action:

Send request for Package data

Server response:

a) OK (The response contains the WAP consisted of one or more WASP(s))

b) Error (Abort, go back to step 0)

Step 3: Send Receipt Check Notice

Client action:

Send Receipt Check Notice

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK

b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires that, for each transaction, the client acknowledge the reception by sending Receipt Check Notice to the server.

[8] If the WAP contains multiple WASP, the notice-info of each notification is set in the package header.

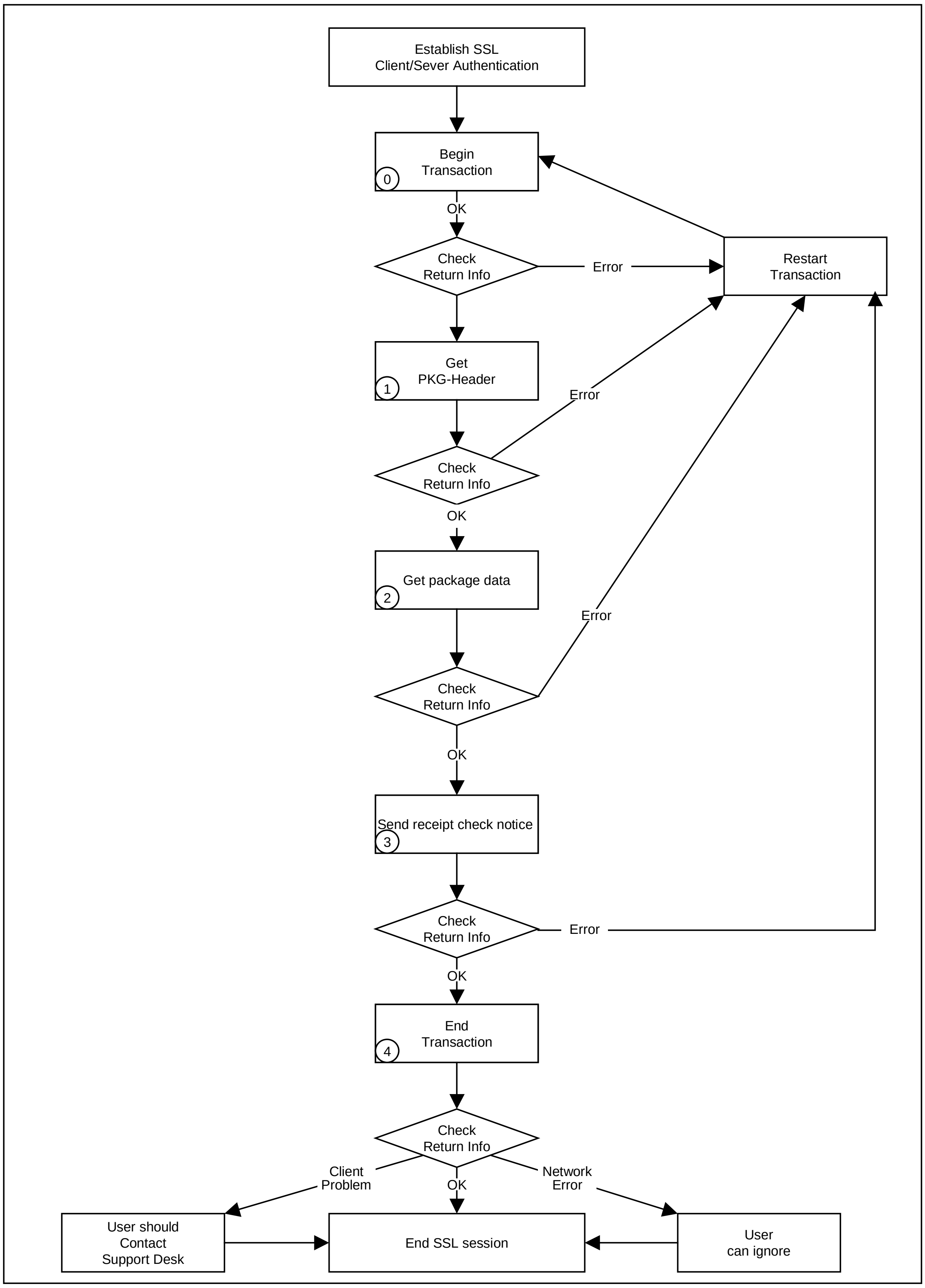


Figure 6 - Application level protocol behavior for notification

5.1.4 Transaction Management Header Elements

The following items, which are all fixed length, are included in all post and response messages:

Attrib. Name	division_hash
Values	ASCII Hexadecimal representation of 160-bit hash value
Length	40 bytes (40 x 8bit characters)
Description	SHA-1 Hash of the current division.

Attrib. Name	protocol_version
Values	Unique
Length	4 bytes (4 x 8bit ASCII char)
Description	A unique identifier for the version of the protocol used to create the transaction data (e.g. 0100 for Version 1.0) First two bytes are for the major version number and last two for the release within this version.

Attrib. Name	transaction_type	
Values	pbeg, ebeg,	
	pend, eend	
	ehdr, phdr,	
	edat, pdat,	
	erct, prct,	
	ephn, pphn	Get package header for notification
	epdn, ppdn	Get package data for notification
	ercn, prcn	Send receipt check notice for notification
	ephd, pphd	Get package header for dispatch list
	epdd, ppdd	Get package data for dispatch list
	ercd, prcd	Send receipt check notice for dispatch list
	epha, ppha	Get package header for application receipt list
	epda, ppda	Get package data for application receipt list
	erca, prca	Send receipt check notice for application receipt list
	ASCII lower case 7-bit ISO 646 e-stands for encrypted, p-plain text	
Length	4 bytes	
Description	Attrib. of the transaction header that identifies the nature of the data transmitted	

Attrib. Name	transaction_id
Values	Unique
Length	36 bytes
Description	A unique identifier assigned by the server associating all transactions involved in the submission of an application. For Begin Transaction this is blank (ASCII x’20’).

Attrib. Name	reserved_use
Values	Reserved for domestic use
Length	32 bytes
Description	This data element is for JPO specific use.

Attrib. Name	total_bytes
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The total size in bytes of the object being sent (WASP containing the Package Header, WASP containing the package data, and the WASP containing the receipt).

Attrib. Name	division_size
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The size in bytes of the data component (chunk) of the object being transferred.

Attrib. Name	division_offset
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	• Value representing the starting position of the data within the object being transferred. • Division_offset starts at 0.

Attrib. Name	division_response_code
Values	see table below; ASCII 4 x 8bit char
Length	4 bytes
Description	Server or client return code used to manage the division mechanism

Division RCs	*Meaning*
0000	OK
FFFF	General Error
FFFE	Resend Last
FFFD	Wait
FFFC	Protocol Sequence Error

ASCII 4 x 8bit char

Attrib. Name	application_response_code
Values	see table below; ASCII 4 x 8bit char
Length	4 bytes
Description	Server or client return code used to manage the application level events

Application RCs	*Meaning*
0000	OK
FFFF	General Error
0001	OK, Package Known
0002	OK, New Package
1000	Pending
FFFB	Client Problem
FFFA	Network Error
FFF9	Protocol Version Error
FFF8	Hash Value of “division hash” in the Transaction Management Header is erroneous.
FFF7	The hash values in package header and the WAD of package data do not match.
FFF6	The signature is invalid (for instance, due to a signature verification error or validation data expiration).[9]

ASCII 4 x 8bit char

Attrib. Name	encoding_method
Values	see table below

Application RC	*Meaning*
UTF8	UNICODE UTF8
SJIS	UNICODE Shift-JIS

[9] This code is applied when the server cannot verify the authentication in Get package header.

	ASCII 4 x 8bit char
Length	4 bytes
Description	Encoding scheme for error message translation.

Attrib. Name	error_message
Values	UNICODE UTF8, UNICODE Shift-JIS
Length	256 bytes (256 x 8bits)
Description	Optional text explaining the reason for error response codes. If an error message is needed for both division and application response codes, these should be concatenated. Each server will choose one of the specified encoding schemes to translate the error message into human readable format.

5.1.4.1 Transaction Management Data elements

Attrib. Name	max_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Maximum bytes allowed for a division.
Example	0000000000008192 (8Kbytes)

5.1.4.2 Server parameters

Attrib. Name	Server_timeout
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	Time in seconds before the server can assume that a client has lost its network connection and the transaction can be abandoned.
Example	0000000000000120 (2 minutes)

Note that the value for the server_timeout at the protocol level is set at the discretion of the individual Office.

5.1.4.3 Client parameters

Attrib. Name	Client_preferred_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Preferred number of bytes to be used for a division.
Example	0000000000004096 (8k)

Attrib. Name	Client_retry_limit
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Number of times the client should resend a division before abandoning the transaction
Example	0000000000000005 (5 retries)

Attrib. Name	Client_retry_wait
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The time in seconds the client should wait before issuing a retry
Example	0000000000000005 (5 secs)

Note that the value for the client_retry_wait at the protocol level is set at the application level.

5.1.5 *Division mechanism*

When sending data between the client and server, this data is divided into manageable chunks which, together with a transaction management header, are called divisions. Under the control of the client, the size of these divisions can vary during the life of the transactions. This provides a pacing mechanism that can be used to overcome Internet transmission problems.

The initial size of the division data message is set to the smallest of either:

(a) max_division_size returned by the server as a response to the Begin Transaction Request
(b) client_preferred_division_size set in the startup parameters of the client

The client builds one or more divisions made up of the transmission management header and a data message. As each division is sent to the server, the server checks for completeness of the transmission by calculating the hash value of the division.

5.1.5.1 *Calculating the division hash value*

The hash is calculated on the basis of all fields in the header as well as any data message. The hash, which is calculated using the SHA-1 algorithm, is placed as the first element of each division.

It has been agreed before the server rejects a package as invalid, it should check the version of the protocol before checking the hash value in case a future version of the protocol should adopt a different hash algorithm.

The following fields of the HTTP Post or response message are therefore included in the hash calculation:

5.1.1.1	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application rc*	*Encoding method*	*error message*	*data message*
Length	4	4	36	32	16	16	16	4	4	4	256	???

5.1.6 *Event Level Protocol*

Transactions described in this section are further illustrated in Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12 below.

5.1.6.1 **Begin Transaction**

The Begin Transaction post message submitted by the client contains the highest protocol version supported by the client. If the server supports the version provided by the client, it should communicate with the client in accordance with the rules for that version of the protocol and use that version number in all response messages. If the server cannot support the protocol version specified by the client, the application response code should indicate protocol version error, and the version number specified in the response message should be the highest protocol version supported by the server. The server should support earlier versions.

Post Message

5.1.1.1	*Division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*

Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pbeg	Blank	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*Transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	16
Value	X	0100	pbeg	New id	???	16	16	0	0	0	???	???	???

Data Message: max_division_size (16 bytes)

5.1.6.2 Send Package Header

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	phdr	tranid	???	X	Y	Z	0	0	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	phdr	tranid	???	0	0	0	a	b	???	blank	None

5.1.6.3 Send Package Data

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pdat	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WASP containing package data

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pdat	tranid	???	0	0	0	a	b	???	blank	None

5.1.6.4 Get Receipt

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prct	tranid	???	0	0	0	???	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*

Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	prct	tranid	???	x	y	z	???	0	???	blank	Receipt

Data Message: WASP containing receipt

5.1.6.5 *End Transaction*

Post Message

Name	*division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pend	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pend	tranid	???	0	0	0	a	b	???	???	None

5.1.6.6 *Get Package Header for notification*

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pphn	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	pphn	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.6.7 *Get Package Data for notification*

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppdn	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppdn	tranid	???	x	y	Z	0	0	???	blank	pkgdata

Data Message: WAP containing WASP

5.1.6.8 *Send Receipt Check Notice for notification*

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcn	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcn	tranid	???	0	0	0	a	b	???	blank	None

5.1.6.9 Get Package Header for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	pphd	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	Pphd	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.6.10 Get Package Data for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppdd	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	Ppdd	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WAP containing WASP

5.1.6.11 Send Receipt Check Notice for dispatch list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcd	tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prcd	tranid	???	0	0	0	a	b	???	blank	None

5.1.6.12 Get Package Header for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppha	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppha	tranid	???	X	Y	Z	a	b	???	blank	pkghdr

Data Message: WASP containing package header

5.1.6.13 Get Package Data for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	ppda	tranid	???	0	0	0	a	b	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	???
Value	X	0100	ppda	tranid	???	x	y	z	0	0	???	blank	pkgdata

Data Message: WAP containing WASP

5.1.6.14 Send Receipt Check Notice for application receipt list

Post Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prca	Tranid	???	0	0	0	0	0	???	blank	None

Response Message

Name	*division hash*	*Protocol Version*	*transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application RC*	*Encoding method*	*error message*	*data message*
Length	40	4	4	36	32	16	16	16	4	4	4	256	0
Value	X	0100	prca	tranid	???	0	0	0	a	b	???	blank	None

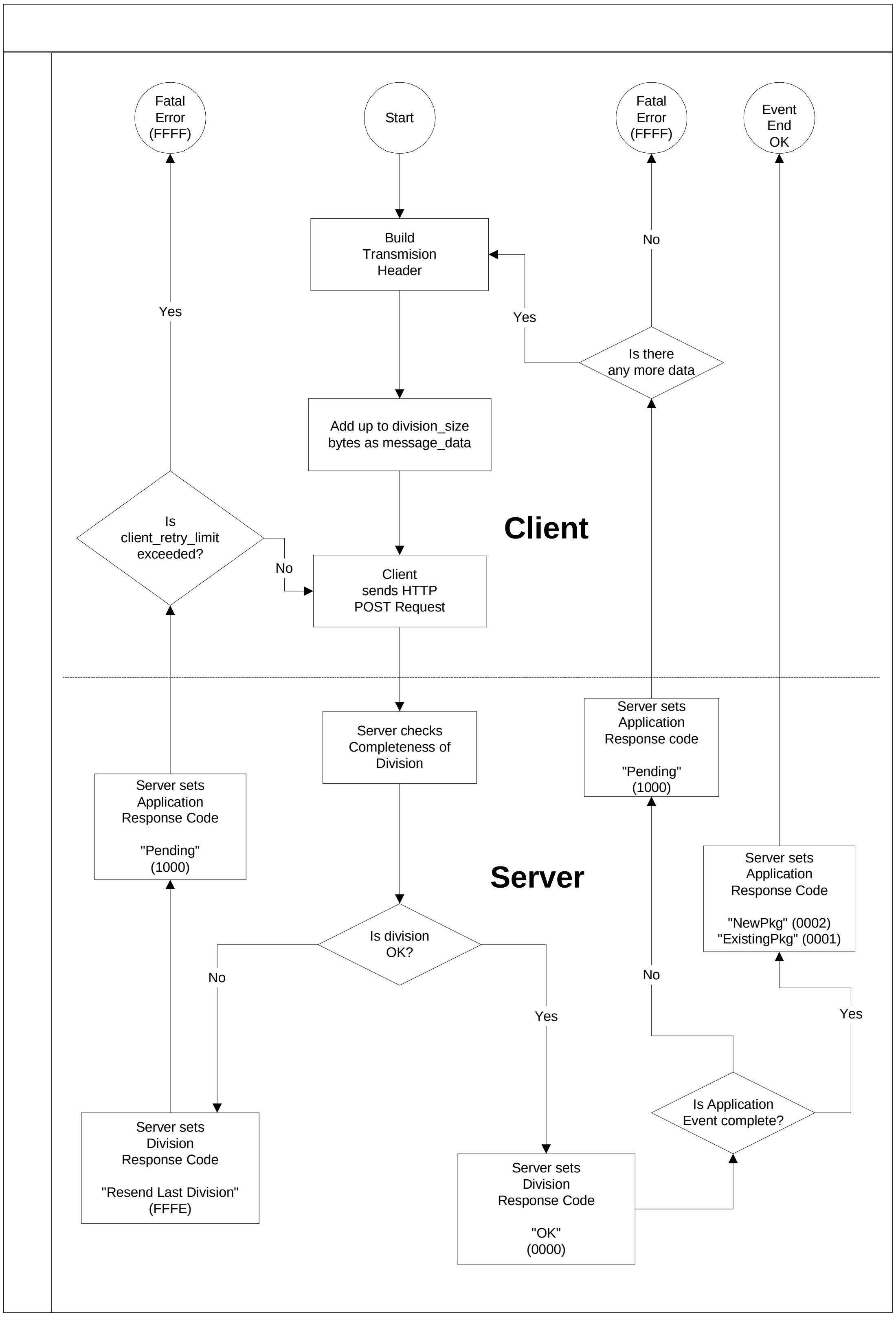

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Event End OK
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "NewPkg" (0002) "ExistingPkg" (0001)
Is division OK?
No
No
Yes
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 7 - Send package header behavior

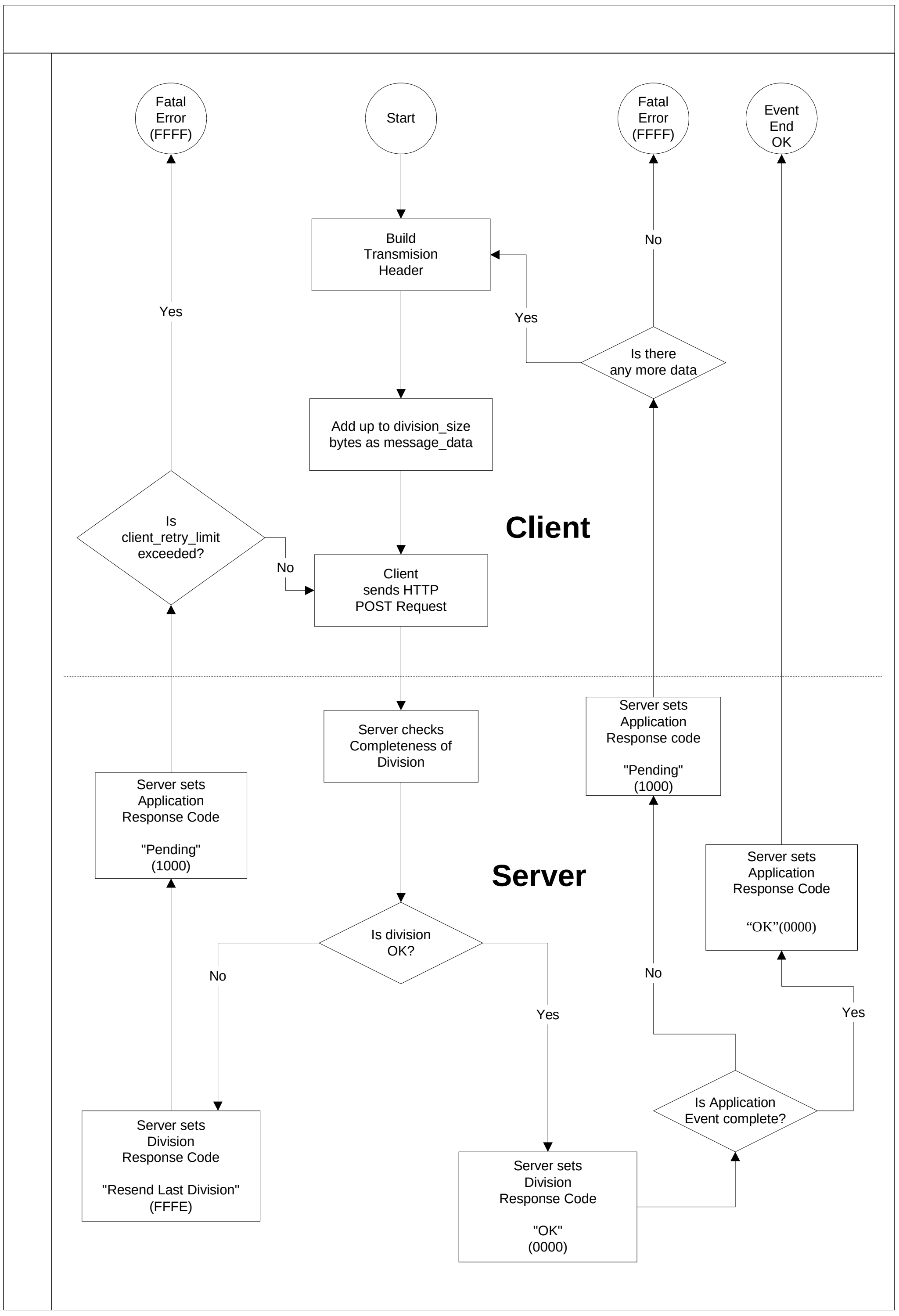

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Event End OK
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server sets Application Response code "Pending" (1000)
Server checks Completeness of Division
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "OK"(0000)
Is division OK?
No
No
Yes
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 8 - Send package data behavior

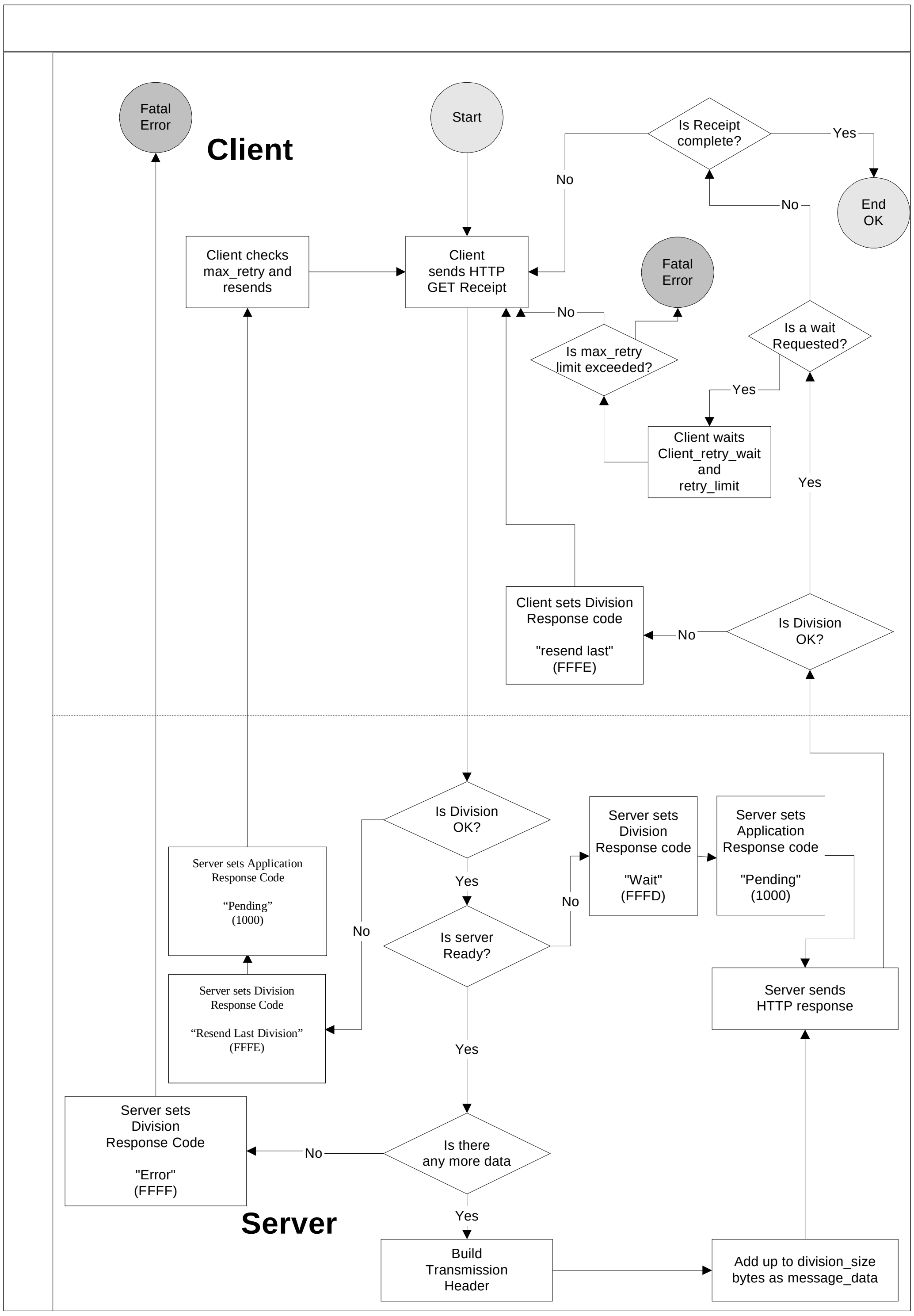

Fatal Error
Client
Start
Is Receipt complete?
Yes
No
End OK
No
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is a wait Requested?
Is max_retry limit exceeded?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Server sets Division Response code "Wait" (FFFD)
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (1000)
Yes
No
No
Is server Ready?
Server sends HTTP response
Server sets Division Response Code "Resend Last Division" (FFFE)
Yes
Server sets Division Response Code "Error" (FFFF)
No
Is there any more data
Yes
Server
Build Transmission Header
Add up to division_size bytes as message_data

Figure 9 - Get receipt behavior

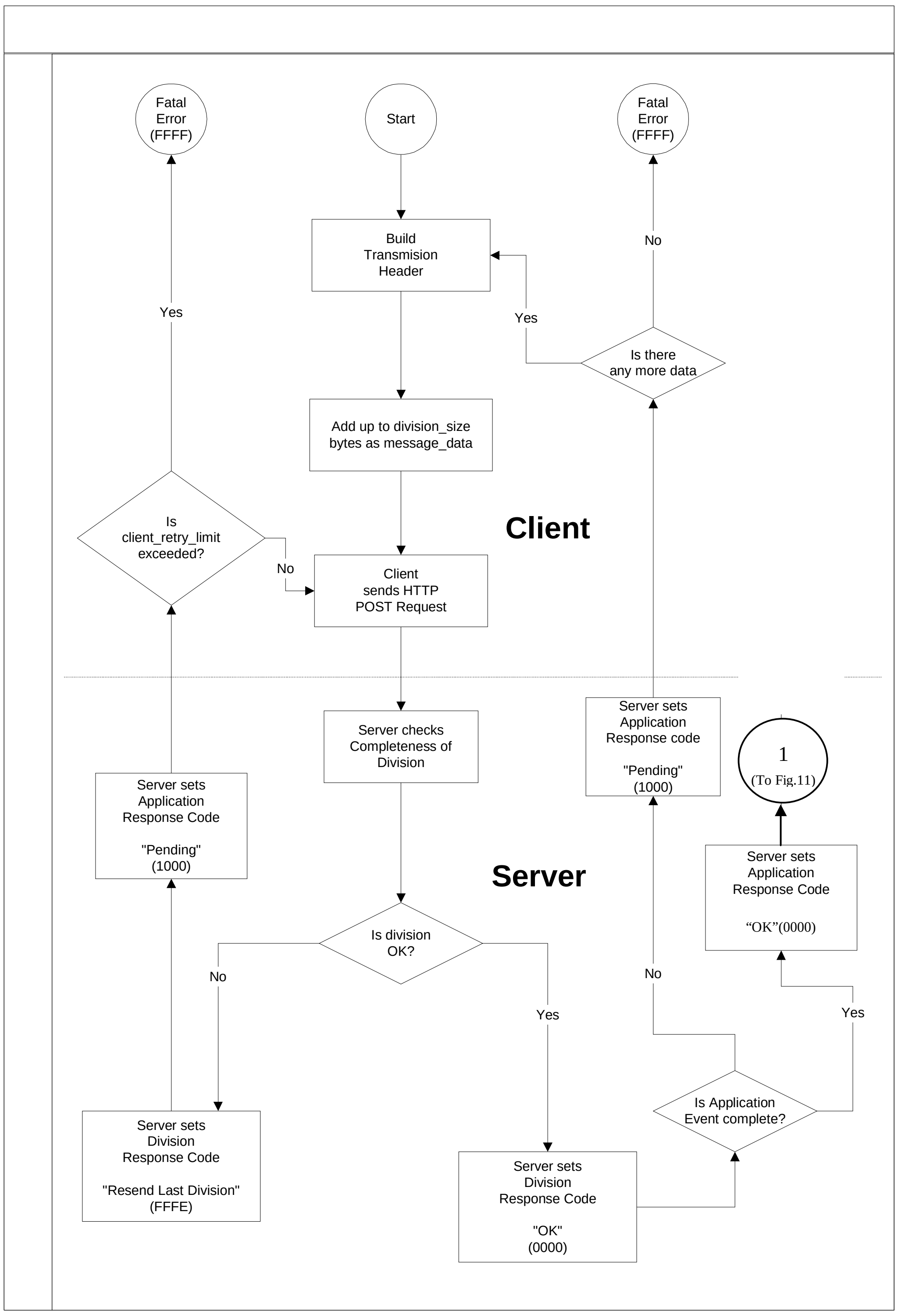

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Build Transmision Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
1 (To Fig.11)
Server sets Application Response Code "Pending" (1000)
Server
Server sets Application Response Code "OK"(0000)
Is division OK?
No
Yes
No
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 10 - Get package header behavior <upstream>

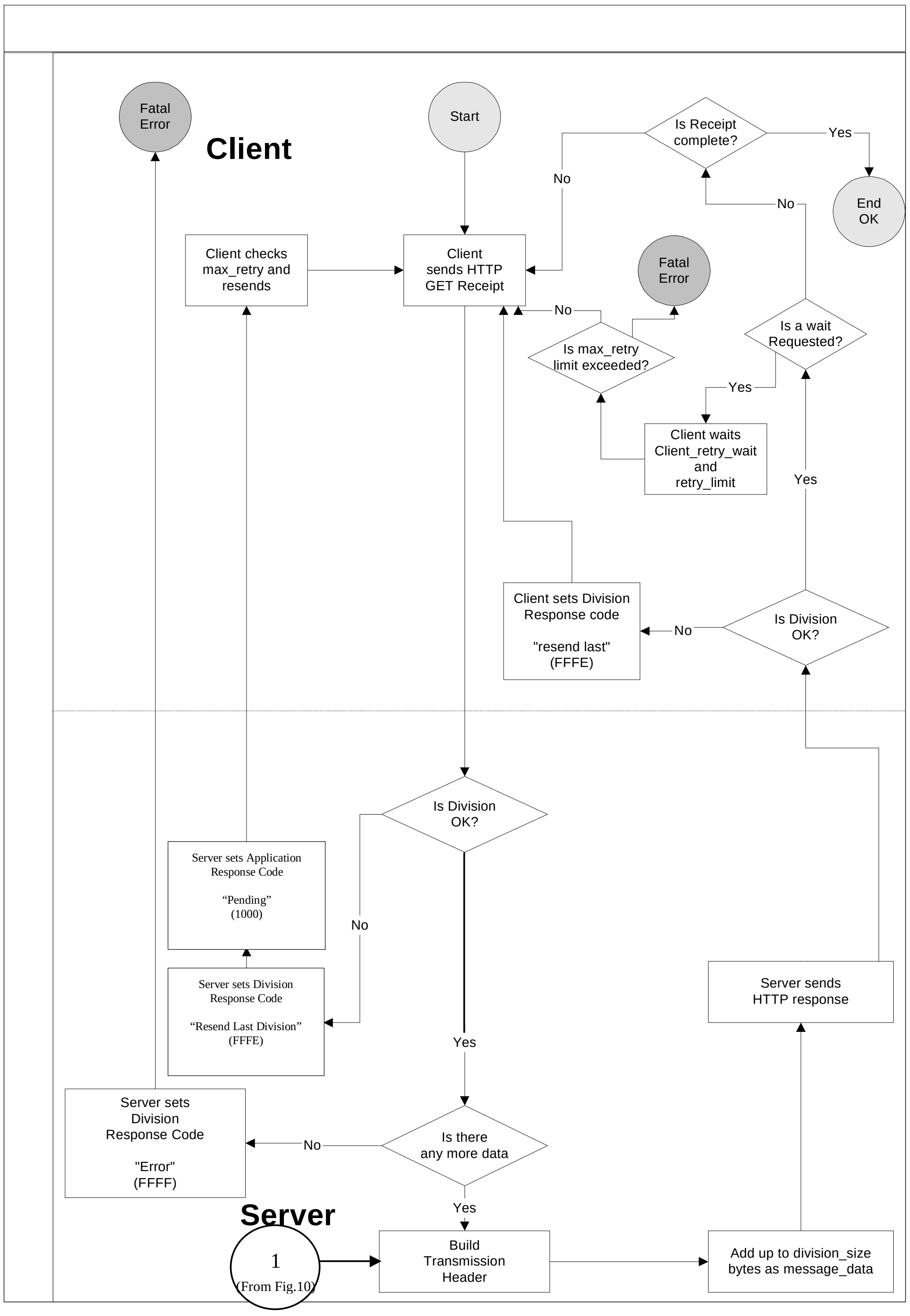

Client
Fatal Error
Start
Is Receipt complete?
Yes
No
End OK
No
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is max_retry limit exceeded?
Is a wait Requested?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Server sets Application Response Code "Pending" (1000)
No
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sends HTTP response
Yes
Server sets Division Response Code "Error" (FFFF)
No
Is there any more data
Yes
Server
1 (From Fig.10)
Build Transmission Header
Add up to division_size bytes as message_data

Figure 11 - Get package header behavior <downstream>

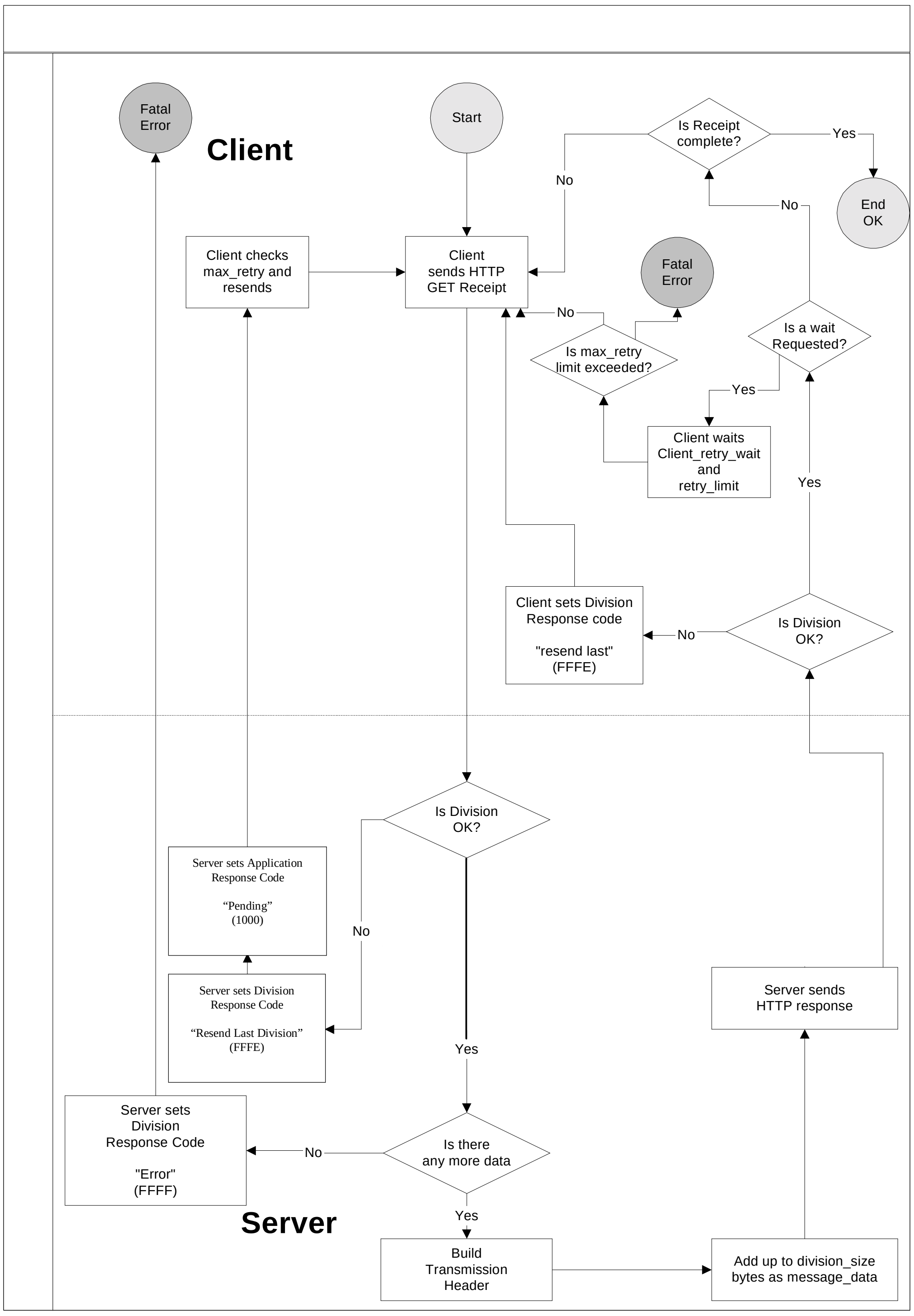

Fatal Error
Client
Start
Is Receipt complete?
Yes
No
End OK
Client checks max_retry and resends
Client sends HTTP GET Receipt
Fatal Error
No
Is max_retry limit exceeded?
Is a wait Requested?
Yes
Client waits Client_retry_wait and retry_limit
Yes
Client sets Division Response code "resend last" (FFFE)
No
Is Division OK?
Is Division OK?
Server sets Application Response Code "Pending" (1000)
No
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sends HTTP response
Yes
Server sets Division Response Code "Error" (FFFF)
No
Is there any more data
Server
Yes
Build Transmission Header
Add up to division_size bytes as message_data

Figure 12 - Get package data behavior

5.2 Package/transmission combinations

This section describes the permissible package/transmission combinations, for each of the PCT communications sectors. This standard does not preclude provision of publicly available information by means other than those covered in the standard. Further packaging types (e.g., web-based document delivery) and package/transmission options may become available in the future.

5.2.1 Applicant-Office (international phase) sector

IA documents may be filed by on-line means (using PKI) over the public Internet, over a private network, or transmitted off-line (using PKI or non-PKI) on physical media. The option of on-line filing of an IA utilizing a non-PKI method is not presently permitted,[10] except under possible transitional reservations permitted by AIs Section 703(f) (see AI section 7.1.1 as to the consequences of non-PKI filing under such a transitional reservation).

Figure 13 shows a matrix of the various submission mechanism/packaging combinations that are permissible in the Applicant-Office (international phase) sector as specified under this standard. In summary, for each submission mechanism:

(a) On-line/Internet: The SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(b) On-line/secure: The SEP, WASP or WAP must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network; 2) the Internet using channel level encryption (e.g. SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(c) Off-line/physical media: either the SEP, WASP, WAP or WAD package must be used. Physical media (e.g., diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

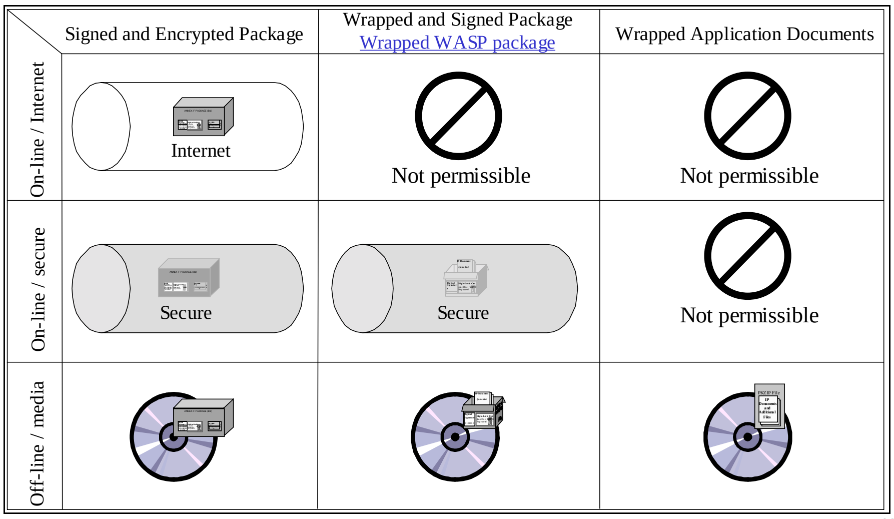


Figure 13 *–Package/transmission combinations permitted in the Applicant-Office (international phase) sector*

[10] See footnote 4.

5.2.2 Office-Office sector

All Office-Office sector data exchange must be conducted utilizing PKI-based data exchange. IA documents may be exchanged by on-line means over the public Internet or over a private network (such as Tri-Net or WIPONET), or transported on physical media.

Figure 14 shows a matrix of the various submission mechanism/packaging combinations that are permissible as specified under this standard. In summary, for each data exchange mechanism:

(a) On-line/Internet: The SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(b) On-line/secure: The SEP or WASP must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network (e.g. WIPONET, Tri-Net); 2) the Internet using channel level encryption (e.g. SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(c) Off-line/physical media: The SEP or WASP must be used. Physical media (e.g. diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

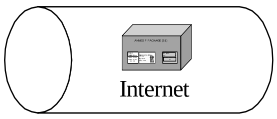
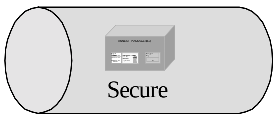
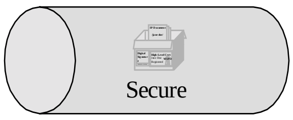
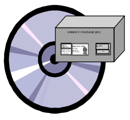
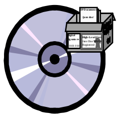
	Signed and Encrypted Package	Wrapped and Signed Package	Wrapped Application Documents
On-line / Internet		🚫 Not permissible	🚫 Not permissible
On-line / secure			🚫 Not permissible
Off-line / media			🚫 Not permissible

Figure 14 – Package/transmission combinations permitted in Office-Office sector

5.2.3 Designated Office sector

The SEP, WASP, or WAD package may be used when exchanging IA documents under the designated Office sector. Figure 15 shows a matrix of the various submission mechanism/packaging combinations that are permissible. In summary, for each data exchange mechanism:

(a) On-line/Internet: the SEP must be used. TCP/IP used to exchange data, in realtime, over the Internet

(b) On-line/secure: the SEP, WASP, or WAD must be used. This is defined as a telecommunication connection established to exchange data, over a network which includes: 1) a private network (e.g. WIPONET, Tri-Net); 2) the Internet using channel level encryption (e.g., SSL); 3) a Virtual Private Network (VPN) connection over the Internet.

(c) Off-line/physical media: either the SEP, WASP, or WAD package must be used. Physical media (e.g., diskette, CD-ROM, DVD, etc.) is used to store IA data with no real-time data exchange.

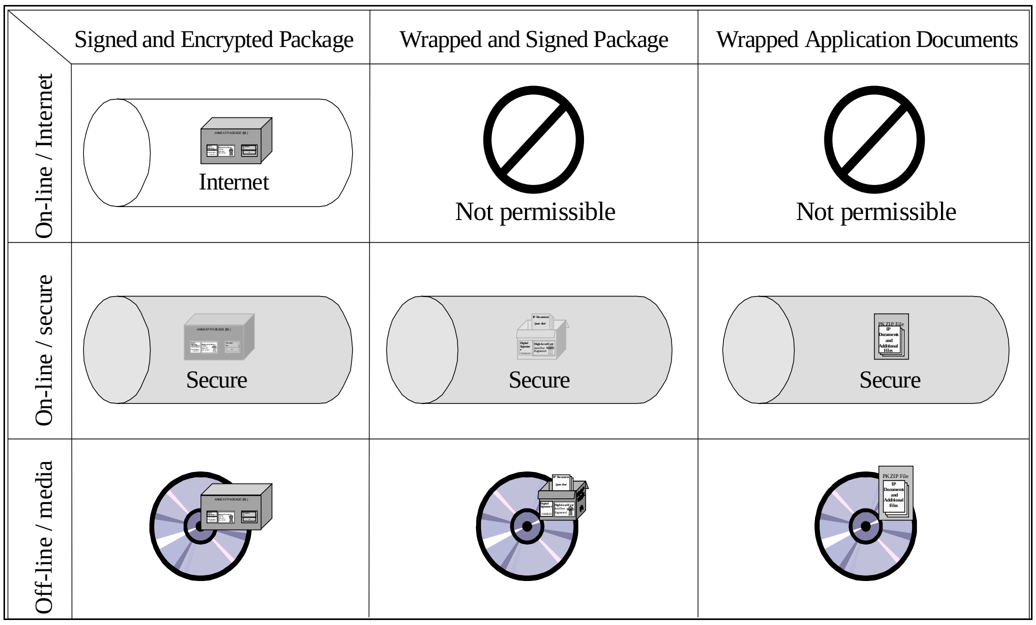


Figure 15 - Package/transmission combinations permitted in designated Office sector

[Annex 8 follows/
L'annexe 8 suit]

COMMENTS BY THE EUROPEAN PATENT OFFICE, THE JAPAN PATENT OFFICE, AND THE UNITED STATES PATENT AND TRADEMARK OFFICE
(THE TRILATERAL OFFICES)

With the following modifications, the Trilateral Offices accept the Japan Patent Office's revised version of PFC02/003:

5 TRANSMISSION

The IA package can be transmitted over secure or non-secure channels depending on the package type. This section includes the protocol to be followed as well as the package/transmission combinations that are permitted in the Applicant-Office (international phase), Office-Office, and designated Office communication sectors. While additional sectors are referred to in this standard (see section 2.3), permissible transmission/package combinations can be categorized in the three sectors listed above.

5.1 The E-filing interoperability protocol

This section describes both the transmission layer protocol between the clients and the server as well as providing a definition of the behavior required of both the client and the server.

The protocol is designed to support HTTP communication over an SSL Tunnel for all PKI based E-filing solutions and includes the following capabilities:

(a) Enables large applications to be transmitted via multiple HTTP post actions to address reliability and integrity issues
(b) Efficient error detection and correction
(c) Enables offices to control optimal transaction size

Note that this is an evolving protocol, with production systems in development at a number of IP Offices, and further revisions are foreseen.

5.1.1 Principles

The following principles have been adopted for the interoperability protocol:

(a) All communications between client and server is in the form of HTTP post actions initiated by the client
(b) All post requests and resulting responses use the same transaction management header followed by an optional data block
(c) All transmissions use the Division mechanism to divide large blobs of data into manageable chunks with a protocol that allows for retries and pacing

5.1.2 Application Layer Protocol for application

At the highest level for application, there are five events that the protocol requires a client and server to support. These events are:

(a) Begin Transaction
(b) Send Package Header
(c) Send Package Data
(d) Get Receipt
(e) End Transaction

In between the Begin and End Transactions, there are three types of WASP sent between the client and the server:

(i) The package header contains essential information for initial processing to identify the submission. It is a WASP containing the package header in XML format.
(ii) The package data contains the information for submitting application. It is a WASP consisting of various types of files.
(iii) The receipt is an acknowledgment of the submission. The content of this receipt (XML data plus an optional human readable certificate in PDF or TIFF), which is signed by the receiving Office, is defined in Annex F Appendix I. The date of receipt will be determined according to the usual principles applicable to the filing of applications on paper, including filing by electronic means (such as by facsimile transmission), that is, based on the date prevailing at the location of the Office at the time when the complete transmission of the application has been received.

5.1.2.1 Use of the SSL Tunnel for application

These events are all performed within an SSL tunnel that is established before issuing the Begin Transaction event. The SSL tunnel is built using both client and server authentication. The SSL tunnel may be stopped at the end of the transaction or, if a batch of transmissions is foreseen, the SSL tunnel can be left open and only stopped when all transmissions are complete. The SSL tunnel uses the SSL protocol version 3.0.

5.1.2.2 Application level Events for application

Start SSL session (See Figure 5)

Step 0: Begin Transaction

Client action:

Get transaction Information.

Server response:

Return values in the *transaction_id* and *max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in the submission of an application.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

Step 1: Send Package Header

Client action:

Send package header

Server response:

a) OK
b) Error (Abort, go back to step 0)
c) Package already received; go to step 3 to get the receipt.

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance expired), the Application Response Code (ARC) will remain OK, but the server will capture the error and provide a message on the receipt.

Step 2: Send Package Data

Client action:

Send package data

Server response:

a) OK
b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package data, the server must verify the signature of the WASP and compare the message digest of the unsigned package against the message digest provided in the Package Header in Step 1 of the transaction before returning the ARC to the client. If both conditions are met, the server should return an ARC indicating OK. If the hash values in package header and the WAD of the package data do not match, the ARC value should be set to FFF7. If the signature is invalid (for instance expired), the ARC will remain OK, but the server will capture the error and provide a message on the receipt.

Step 3: Request Receipt

Client action:

Send request

Server response:

a) OK (Receipt object included in response)
b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK
b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires each individual transaction to be acknowledged by an individual receipt.

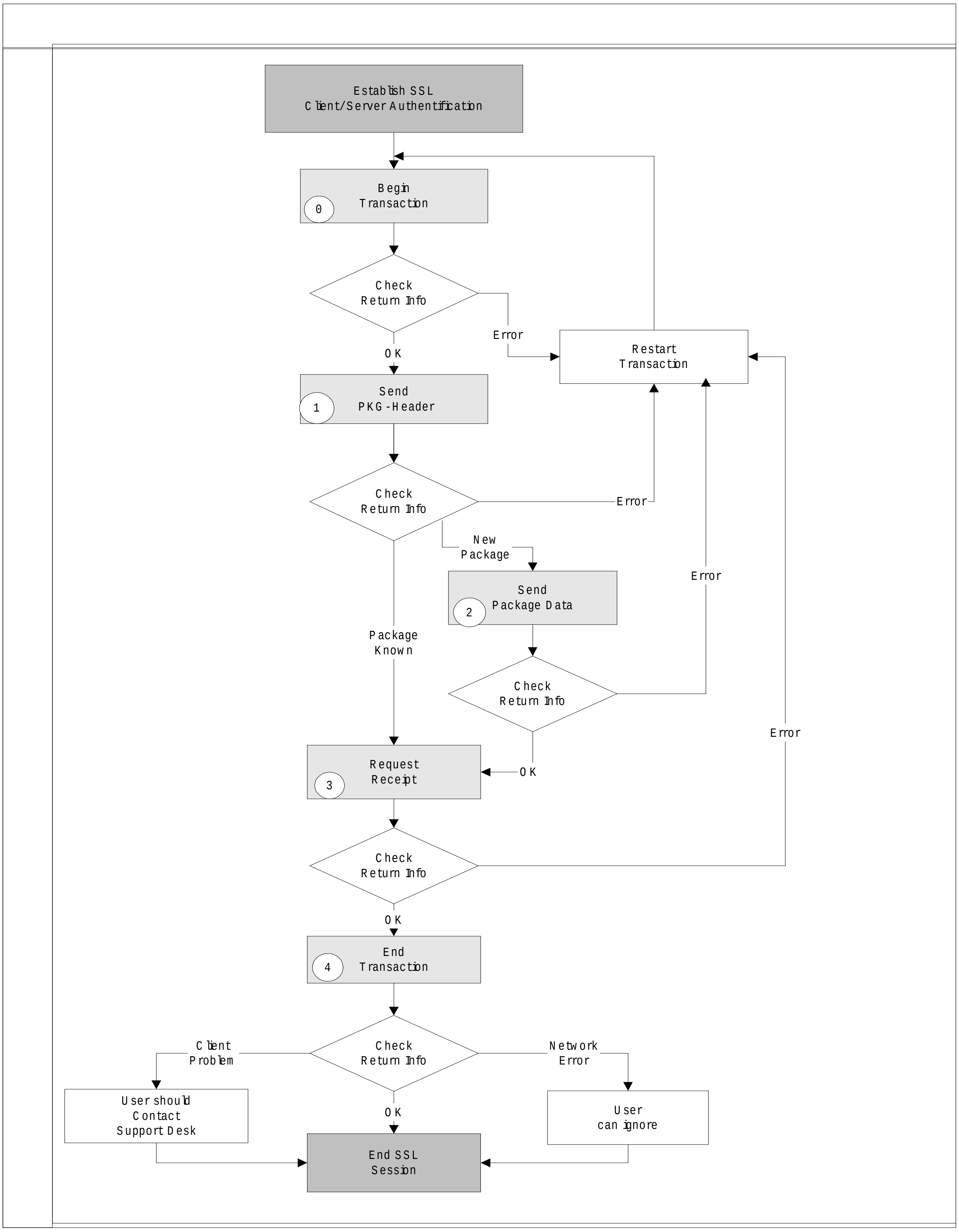


Figure 5 - Application level protocol behavior for application

5.1.3 Application Layer Protocol for notification

At the highest level for notification, there are five events that the protocol requires a client and server to support[11]. These events are:[12]

(a) Begin Transaction
(b) Get Package Header (for notification, or dispatch list, or application receipt list) [13]
(c) Get Package Data (for notification, or dispatch list, or application receipt list) [13]
(d) Send Receipt Check Notice (for notification, or dispatch list, or application receipt list)[13]
(e) End Transaction

In between the Begin and End Transactions, there are two types of WASP and one type of WAP sent between the client and the server:

(i) The *client action* package header contains essential information for initial processing to identify the request for notification. It is a WASP containing the package header in XML format. (This is applied to request to a server from a client.)
(ii) The *server response* package header contains summary information (such as a dispatch-number and the number of notifications to be sent) on the notification to be notified. It is a WASP containing the package header in XML format. (This is applied to response to a client from server.)
(iii) The package data contains the ~~information for dispatching~~ dispatched notification information. It is ~~the~~ a WAP ~~which~~ that consists of one or more WASP(s).

5.1.3.1 Use of the SSL Tunnel for notification

Refer to Section 5.1.2.1, "Use of the SSL Tunnel for application."

5.1.3.2 Application level Events for notification

Start SSL session (See Figure 6)

Step 0: Begin Transaction
Client action:
Get transaction Information.
Server response:
Return values in the *transaction_id and max_division_size* transaction management header elements.

transaction_id is a unique identifier assigned by the server associating all transactions involved in sending a notification.

max_division_size is the maximum number of bytes permitted by the server for the size of a division.

[11] The office may notify existence of notifications to an applicant before these five events.

[12] This protocol may be used for dispatching of the dispatch list and the application receipt list besides dispatching of the notification. The office can choose freely whether it will dispatch the dispatch list and the application receipt list to an applicant.
The dispatch list is the list of the dispatch numbers ~~to~~ for the ~~notifications which~~notifications that the office has dispatched to the applicant, and the application receipt list is the list of the application numbers ~~to~~ for the application documents ~~which~~ that the office has received from the applicant.

[13] The different value of “transaction-type” every kind of document (notification, dispatch list, application receipt list) is used because the server can identify which kind of document is request. (see the section 5.1.4 for details)

Step 1: Get Package Header

Client action:

Send request for package header (The WASP of package header for request of notification is contained.)

Server response:

a) OK (The response contains the WASP of package header containing summary information (such as dispatch-number, number-of-notification) of notifications.)[14]

b) Error (Abort, go back to step 0)

After receiving the last division of the WASP containing the package header, the server must verify the signature of the WASP. If the signature is invalid (for instance, due to a signature verification error or validation data expiration), the application response code (ARC) value is set to FFF6.

If the number of sendable notifications in package header of Server response is "0(zero)" (there is no sendable notifications), then go to Step 4.

Step 2: Get Package Data

Client action:

Send request for Package data

Server response:

a) OK (The response contains the WAP consisted of one or more WASP(s))

b) Error (Abort, go back to step 0)

Step 3: Send Receipt Check Notice

Client action:

Send Receipt Check Notice

Server response:

a) OK

b) Error (Abort, go back to step 0)

Step 4: End Transaction.

Client action:

Send acknowledgment of completion including information about any client problem to the server.

Server response:

a) OK

b) Error (Client can ignore this response)

Close SSL session

In all cases of SSL Tunnel, the current protocol requires that, for each transaction, the client acknowledge the reception by sending Receipt Check Notice to the server.

[14] If the WAP contains multiple WASP, the notice-info of each notification is set in the package header.

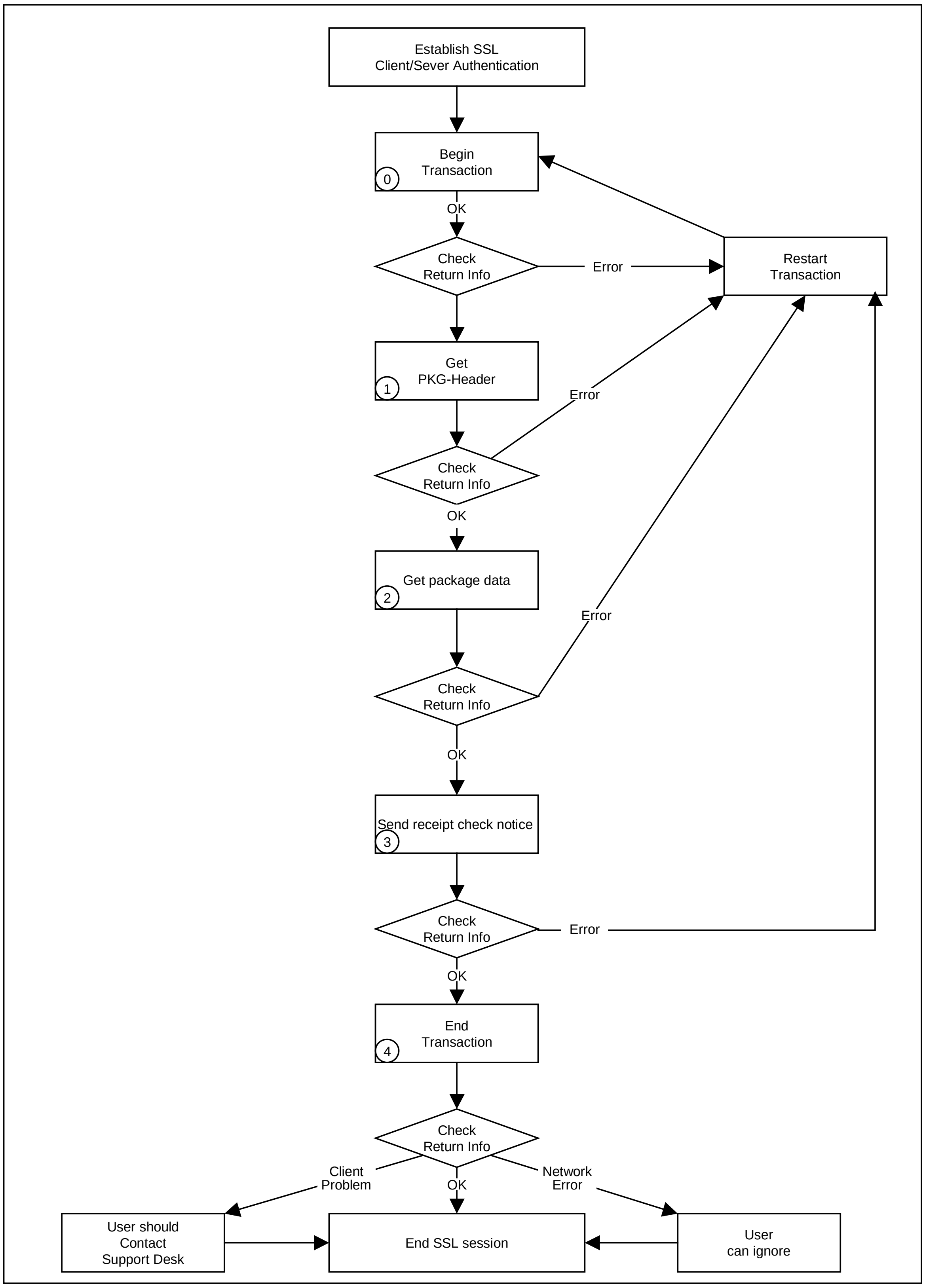


Figure 6 - Application level protocol behavior for notification

5.1.4 Transaction Management Header Elements

The following items, which are all fixed length, are included in all post and response messages:

Attrib. Name	division_hash
Values	ASCII upper case Hexadecimal representation of 160-bit hash value
Length	40 bytes (40 x 8bit characters)
Description	SHA-1 Hash of the current division.

Attrib. Name	protocol_version
Values	Unique
Length	4 bytes (4 x 8bit ASCII char)
Description	A unique identifier for the version of the protocol used to create the transaction data (e.g. 0100 for Version 1.0) First two bytes are for the major version number and last two for the release within this version.

Attrib. Name	transaction_type	
Values	pbeg, ebeg,	
	pend, eend	
	ehdr, phdr,	
	edat, pdat,	
	erct, prct,	
	ephn, pphn	Get package header for notification
	epdn, ppdn	Get package data for notification
	ercn, prcn	Send receipt check notice for notification
	ephd, pphd	Get package header for dispatch list
	epdd, ppdd	Get package data for dispatch list
	ercd, prcd	Send receipt check notice for dispatch list
	epha, ppha	Get package header for application receipt list
	epda, ppda	Get package data for application receipt list
	erca, prca	Send receipt check notice for application receipt list
	ASCII lower case 7-bit ISO 646 e-stands for encrypted, p-plain text	
Length	4 bytes	
Description	Attrib. of the transaction header that identifies the nature of the data transmitted	

Attrib. Name	transaction_id
Values	Unique
Length	36 bytes
Description	A unique identifier assigned by the server associating all transactions involved in the submission of an application. For Begin Transaction this is blank (ASCII x'20').

Attrib. Name	reserved_use
Values	Reserved for domestic use
Length	32 bytes
Description	This data element is for JPO specific use.

Attrib. Name	total_bytes
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The total size in bytes of the object being sent (WASP containing the Package Header, WASP containing the package data, and the WASP containing the receipt).

Attrib. Name	division_size
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The size in bytes of the data component (chunk) of the object being transferred.

Attrib. Name	division_offset
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	• Value representing the starting position of the data within the object being transferred. • Division_offset starts at 0.

Attrib. Name	division_response_code
Values	

Division RCs	*Meaning*
0000	OK
FFFF	General Error
FFFE	Resend Last
FFFD	Wait
FFFC	Protocol Sequence Error

	ASCII 4 x 8bit char
Length	4 bytes
Description	Server or client return code used to manage the division mechanism

Attrib. Name	application_response_code
Values	

Application RCs	*Meaning*
0000	OK
FFFF	General Error
0001	OK, Package Known
0002	OK, New Package
1000	Pending
FFFB	Client Problem
FFFA	Network Error
FFF9	Protocol Version Error
FFF8	Hash Value of "division hash" in the Transaction Management Header is erroneous.
FFF7	The hash values in package header and the WAD of package data do not match.
FFF6	The signature is invalid (for instance, due to a signature verification error or validation data expiration).[15]

	ASCII 4 x 8bit char
Length	4 bytes
Description	Server or client return code used to manage the application level events

Attrib. Name	encoding_method
Values	

Application RC	*Meaning*
UTF8	UNICODE UTF8
SJIS	UNICODE Shift-JIS

[15] This code is applied when the server cannot verify the authentication in Get package header.

	ASCII 4 x 8bit char
Length	4 bytes
Description	Encoding scheme for error message translation.

Attrib. Name	error_message
Values	UNICODE UTF8, UNICODE Shift-JIS
Length	256 bytes (256 x 8bits)
Description	Optional text explaining the reason for error response codes. If an error message is needed for both division and application response codes, these should be concatenated. Each server will choose one of the specified encoding schemes to translate the error message into human readable format.

5.1.4.1 Transaction Management Data elements

Attrib. Name	max_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Maximum bytes allowed for a division.
Example	0000000000008192 (8Kbytes)

5.1.4.2 Server parameters

Attrib. Name	Server_timeout
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	Time in seconds before the server can assume that a client has lost its network connection and the transaction can be abandoned.
Example	0000000000000120 (2 minutes)

Note that the value for the server_timeout at the protocol level is set at the discretion of the individual Office.

5.1.4.3 Client parameters

Attrib. Name	Client_preferred_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Preferred number of bytes to be used for a division.
Example	0000000000004096 (8k)

Attrib. Name	Client_retry_limit
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Number of times the client should resend a division before abandoning the transaction
Example	0000000000000005 (5 retries)

Attrib. Name	Client_retry_wait
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The time in seconds the client should wait before issuing a retry
Example	0000000000000005 (5 secs)

Note that the value for the client_retry_wait at the protocol level is set at the application level.

5.1.5 Division mechanism

When sending data between the client and server, this data is divided into manageable chunks which, together with a transaction management header, are called divisions. Under the control of the client, the size of these divisions can vary during the life of the transactions. This provides a pacing mechanism that can be used to overcome Internet transmission problems.

The initial size of the division data message is set to the smallest of either:

(a) max_division_size returned by the server as a response to the Begin Transaction Request
(b) client_preferred_division_size set in the startup parameters of the client

The client builds one or more divisions made up of the transmission management header and a data message. As each division is sent to the server, the server checks for completeness of the transmission by calculating the hash value of the division.

5.1.5.1 Calculating the division hash value

The hash is calculated on the basis of all fields in the header as well as any data message. The hash, which is calculated using the SHA-1 algorithm, is placed as the first element of each division.

~~It has been agreed b~~Before the server rejects a package as invalid, it should check the version of the protocol before checking the hash value in case a future version of the protocol should adopt a different hash algorithm.

The following fields of the HTTP Post or response message are therefore included in the hash calculation:

5.1.5.1	*Protocol Version*	*Transaction type*	*transaction id*	*Reserved use*	*total bytes*	*division size*	*division offset*	*division RC*	*application rc*	*Encoding method*	*error message*	*data message*
Length	4	4	36	32	16	16	16	4	4	4	256	???

5.1.6 Event Level Protocol

Transactions described in this section are further illustrated in Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12 below.

5.1.6.1 Begin Transaction

The Begin Transaction post message submitted by the client contains the highest protocol version supported by the client. If the server supports the version provided by the client, it should communicate with the client in accordance with the rules for that version of the protocol and use that version number in all response messages. If the server cannot support the protocol version specified by the client, the application response code should indicate protocol version error, and the version number specified in the response message should be the highest protocol version supported by the server. The ~~server~~ client should support earlier versions.

[Annex 9 follows/
L'annexe 9 suit]

COMMENTS BY THE JAPAN PATENT OFFICE

JPO would like to further modify our proposal PFC-02-003. The modification points are as follows.

λ Replacement of Figure 6 – Application level protocol behavior for notification

λ Modification of description in the section 5.1.4 Transaction Management Header Elements
  Addition of the second line from the top,
  Revision of the Description column and Addition of footnote for the Attrib. "transaction_type" ,
  Revision of the Values column and the Description column for Attrib. "reserved_use",
  Revision of the Values column for Attrib. "application_response_code"

λ Modification of description in the section 5.1.4.3 Client parameters
  Addition of the footnote for the Attrib. "Client_retry_limit"

λ Modification of description in the section 5.1.5 Division mechanism
  Addition of the words in the ninth line from the top

λ Replacement of Figure 7 – Send package header behavior.

Only the further revised parts of Annex F section 5 are attached

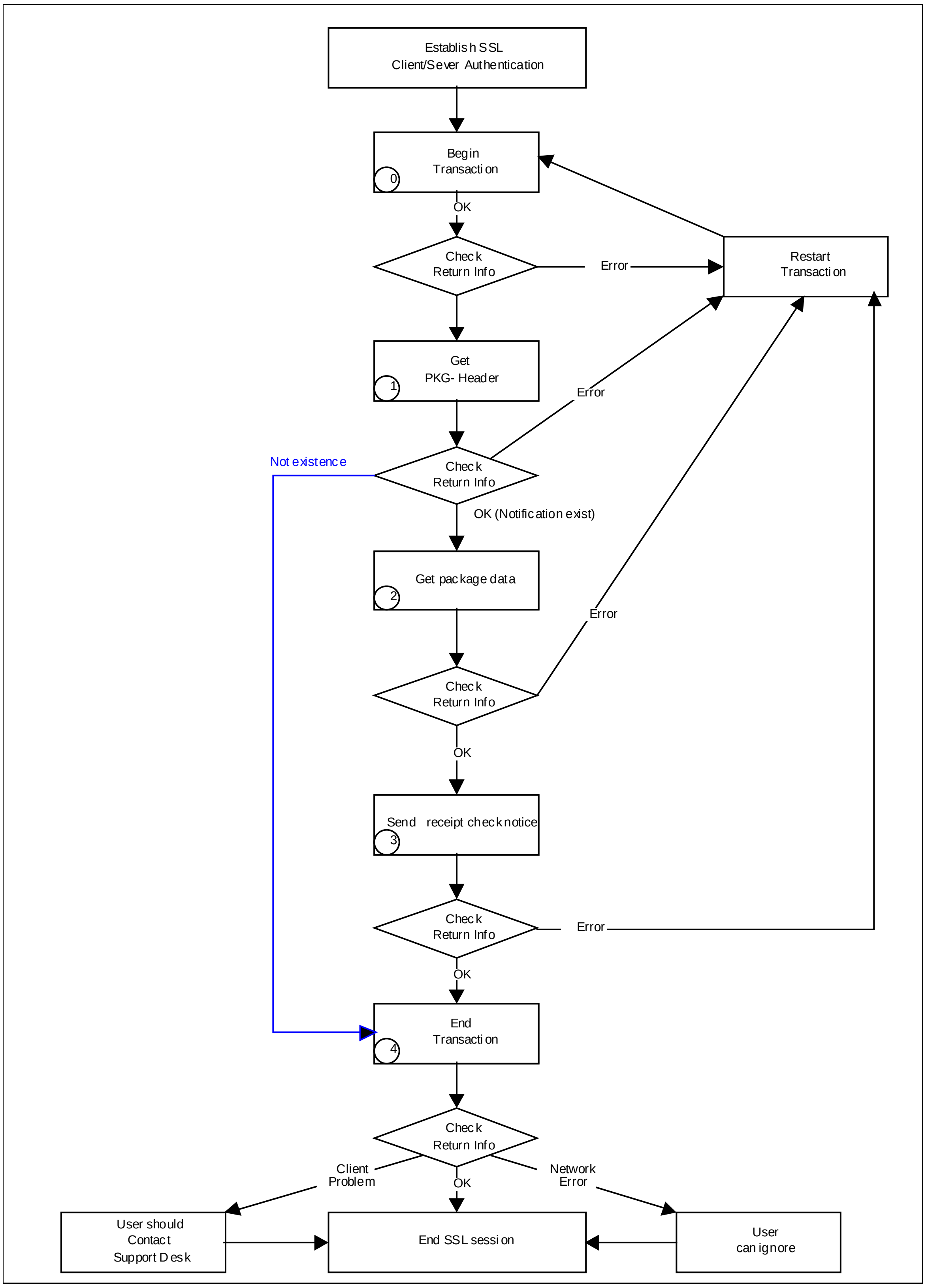


Figure 6 - Application level protocol behavior for notification

5.1.4 Transaction Management Header Elements

The following items, which are all fixed length, are included in all post and response messages:
Besides, unused parameters of header elements are set space (ASCII ‘20’) as long as not configured.

Attrib. Name	division_hash
Values	ASCII Hexadecimal representation of 160-bit hash value
Length	40 bytes (40 x 8bit characters)
Description	SHA-1 Hash of the current division.

Attrib. Name	protocol_version
Values	Unique
Length	4 bytes (4 x 8bit ASCII char)
Description	A unique identifier for the version of the protocol used to create the transaction data (e.g. 0100 for Version 1.0) First two bytes are for the major version number and last two for the release within this version.

Attrib. Name	transaction_type	
Values	pbeg, ebeg,	
	pend, eend	
	ehdr, phdr,	
	edat, pdat,	
	erct, prct,	
	ephn, pphn	Get package header for notification
	epdn, ppdn	Get package data for notification
	ercn, prcn	Send receipt check notice for notification
	ephd, pphd	Get package header for dispatch list
	epdd, ppdd	Get package data for dispatch list
	ercd, prcd	Send receipt check notice for dispatch list
	epha, ppha	Get package header for application receipt list
	epda, ppda	Get package data for application receipt list
	erca, prca	Send receipt check notice for application receipt list
	ASCII lower case 7-bit ISO 646 e-stands for encrypted, p-plain text	
Length	4 bytes	
Description	Attrib. of the transaction header that identifies the nature of the data transmitted. The value beginning with letter d or z is not available.	

Note that the value beginning with the letter d or z is reserved for domestic application or the other transmission.

Attrib. Name	transaction_id
Values	Unique
Length	36 bytes
Description	A unique identifier assigned by the server associating all transactions involved in the submission of an application. For Begin Transaction this is blank (ASCII x’20’).

Attrib. Name	reserved_use
Values	Reserved for domestic use (e.g. Server date and time YYYYMMDDHHMMSS)
Length	32 bytes
Description	This data area is available for the option by each RO. (e.g. To inform a client of the machine time of the RO server).

Attrib. Name	total_bytes
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The total size in bytes of the object being sent (WASP containing the Package Header, WASP containing the package data, and the WASP containing the receipt).
Attrib. Name	division_size
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The size in bytes of the data component (chunk) of the object being transferred.

Attrib. Name	division_offset
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	• Value representing the starting position of the data within the object being transferred. • Division_offset starts at 0.

Attrib. Name	division_response_code
Values	see table below; ASCII 4 x 8bit char
Length	4 bytes
Description	Server or client return code used to manage the division mechanism

Division RCs	*Meaning*
0000	OK
FFFF	General Error
FFFE	Resend Last
FFFD	Wait
FFFC	Protocol Sequence Error

ASCII 4 x 8bit char

Attrib. Name	application_response_code
Values	
Length	4 bytes

Application RCs	*Meaning*
0000	OK
FFFF	General Error
0001	OK, Package Known
0002	OK, New Package
0003	OK, Not Existence
1000	Pending
FFFB	Client Problem
FFFA	Network Error
FFF9	Protocol Version Error
FFF8	Hash Value of “division hash” in the Transaction Management Header is erroneous.
FFF7	The hash values in package header and the WAD of package data do not match.
FFF6	The signature is invalid (for instance, due to a signature verification error or validation data expiration).[16]

ASCII 4 x 8bit char

[16] This code is applied when the server cannot verify the authentication in Get package header.

Description	Server or client return code used to manage the application level events

Attrib. Name	encoding_method
Values	

Application RC	*Meaning*
UTF8	UNICODE UTF8
SJIS	UNICODE Shift-JIS

	ASCII 4 x 8bit char
Length	4 bytes
Description	Encoding scheme for error message translation.

Attrib. Name	error_message
Values	UNICODE UTF8, UNICODE Shift-JIS
Length	256 bytes (256 x 8bits)
Description	Optional text explaining the reason for error response codes. If an error message is needed for both division and application response codes, these should be concatenated. Each server will choose one of the specified encoding schemes to translate the error message into human readable format.

5.1.4.1 Transaction Management Data elements

Attrib. Name	max_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Maximum bytes allowed for a division.
Example	0000000000008192 (8Kbytes)

5.1.4.2 Server parameters

Attrib. Name	Server_timeout
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	Time in seconds before the server can assume that a client has lost its network connection and the transaction can be abandoned.
Example	0000000000000120 (2 minutes)

Note that the value for the server_timeout at the protocol level is set at the discretion of the individual Office.

5.1.4.3 Client parameters

Attrib. Name	Client_preferred_division_size
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Preferred number of bytes to be used for a division.
Example	0000000000004096 (8k)

Attrib. Name	Client_retry_limit
Values	Numeric ASCII with left-hand zero padding
Length	16 bytes (16 x 8bit chars)
Description	Number of times the client should resend a division before abandoning the transaction

Example	0000000000000005 (5 retries)

Note that the maximum number of Attrib. Client_retry_limit is NN (16 times). When a server retries more than 16 times, the transmission may be terminated.

Attrib. Name	Client_retry_wait
Values	Numeric ASCII with left-hand zero padding (e.g. 0000000123456789)
Length	16 bytes (16 x 8bit chars)
Description	The time in seconds the client should wait before issuing a retry
Example	0000000000000005 (5 secs)

Note that the value for the client_retry_wait at the protocol level is set at the application level.

5.1.5 Division mechanism

When sending data between the client and server, this data is divided into manageable chunks which, together with a transaction management header, are called divisions. Under the control of the client, the size of these divisions can vary during the life of the transactions. This provides a pacing mechanism that can be used to overcome Internet transmission problems.

The initial size of the division data message is set to the smallest of either:

(c) max_division_size returned by the server as a response to the Begin Transaction Request
(d) client_preferred_division_size set in the startup parameters of the client

The client builds one or more divisions made up of the transmission management header and a data message. As each division is sent in the divided order to the server, the server checks for completeness of the transmission by calculating the hash value of the division.

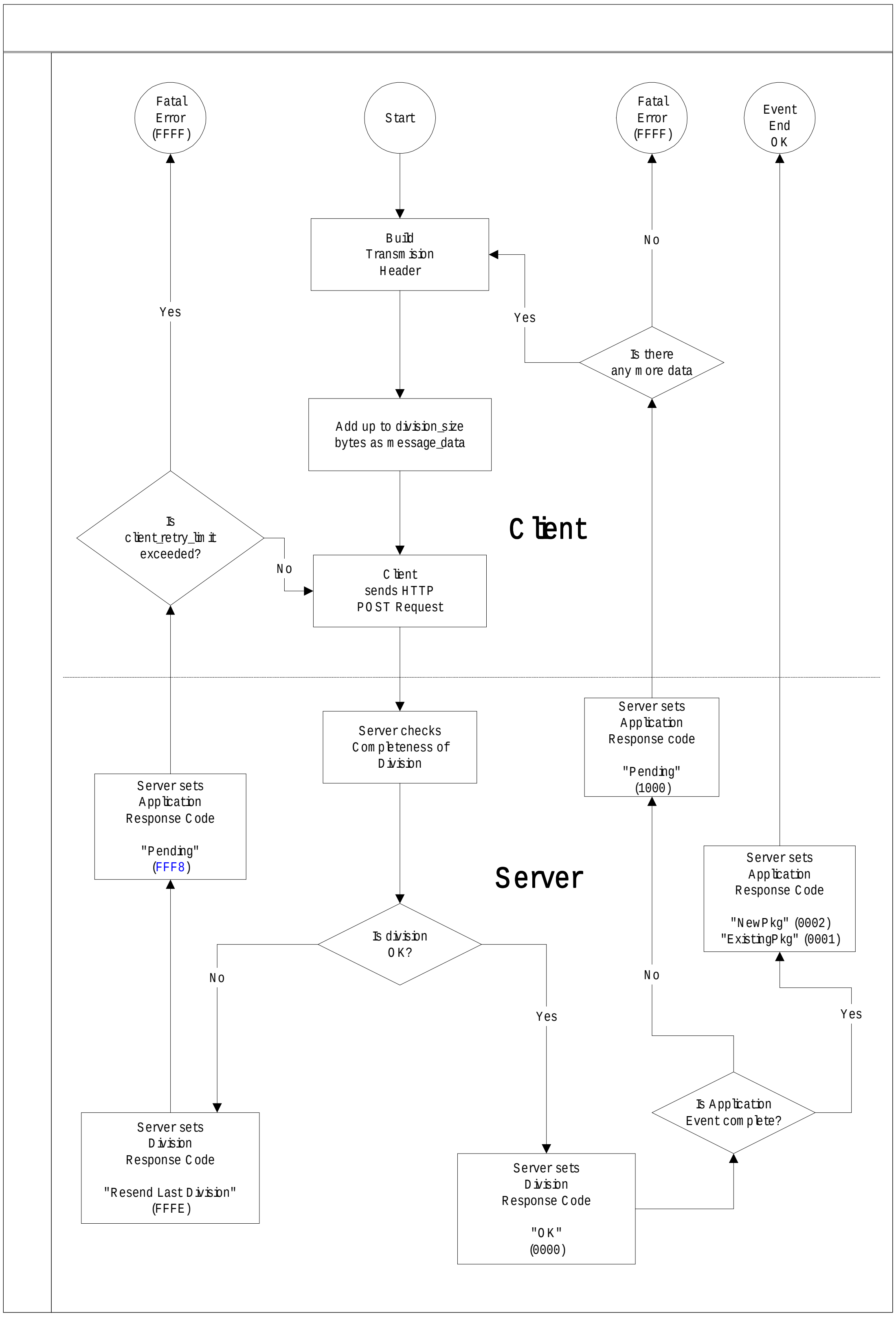

Fatal Error (FFFF)
Start
Fatal Error (FFFF)
Event End OK
Build Transmission Header
No
Yes
Yes
Is there any more data
Add up to division_size bytes as message_data
Client
Is client_retry_limit exceeded?
No
Client sends HTTP POST Request
Server checks Completeness of Division
Server sets Application Response code "Pending" (1000)
Server sets Application Response Code "Pending" (FFF8)
Server
Server sets Application Response Code "NewPkg" (0002) "ExistingPkg" (0001)
Is division OK?
No
No
Yes
Yes
Is Application Event complete?
Server sets Division Response Code "Resend Last Division" (FFFE)
Server sets Division Response Code "OK" (0000)

Figure 7 - Send package header behavior

[End of Annex and of file/
Fin de l'annexe et du dossier]