Specification of the JavaCard APDU protocol in JML

To check your applet with ESC/Java2, you will have to use our JML specifications for the Java Card API classes as follows

```
escjava2 -bootclasspath ..../esc2_jc_api YourApplet.java
```

In particular, your applet will use the APDU protocol, as provided by the API class `javacard.framework.APDU.java`.

The APDU class

An applet's `process` method receives an APDU, on which it invokes

```
public static byte[] getBytes()
public static short getInBlockSize()
public static short getOutBlockSize()
```

and

```
public short setIncomingAndReceive()
public short receiveBytes(short bOff)
public short setOutgoing()
public void  setOutgoingLength(short len)
public void  sendBytes(short bOff, short len)
  ⋮
```

in a certain order!

Informal JavaDoc spec

receiveBytes

```
public short receiveBytes(short bOff)
                   throws APDUException
```

Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset `bOff`. Gets all the remaining bytes if they fit.

Notes:

- *The space in the buffer must allow for incoming block size.*
- *In T=1 protocol, if all the remaining bytes do not fit in the buffer, this method may return less bytes than the maximum incoming block size (IFSC).*
- *In T=0 protocol, if all the remaining bytes do not fit in the buffer, this method may return less than a full buffer of bytes to optimize and reduce protocol overhead.*

Parameters:

`bOff` – the offset into APDU buffer.

Returns:

number of bytes read. Returns 0 if no bytes are available.

Throws:

APDUException – with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` not called or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.
- `APDUException.BUFFER_BOUNDS` if not enough buffer space for incoming block size.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

Informal JavaDoc spec

receiveBytes

`public short receiveBytes(short bOff) throws APDUException`

> Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset bOff. Gets all the remaining bytes if they fit.

Parameters: `bOff` - the offset into APDU buffer.
Returns: number of bytes read. Returns 0 if no bytes are available.
Throws: APDUException - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` not called or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.
- `APDUException.BUFFER_BOUNDS` if not enough buffer space for incoming block size.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

Informal JavaDoc spec

Specification of the invocation order in the JavaDoc is not very clear.

A specification as message sequence chart, finite state machine (FSM), etc. would be better.

Our JML spec of APDU expresses the order using a FSM.

Reference implementation

Our FSM is based on the reference implementation rather than the JavaDoc.

The reference implementation of `APDU` uses 7 flags

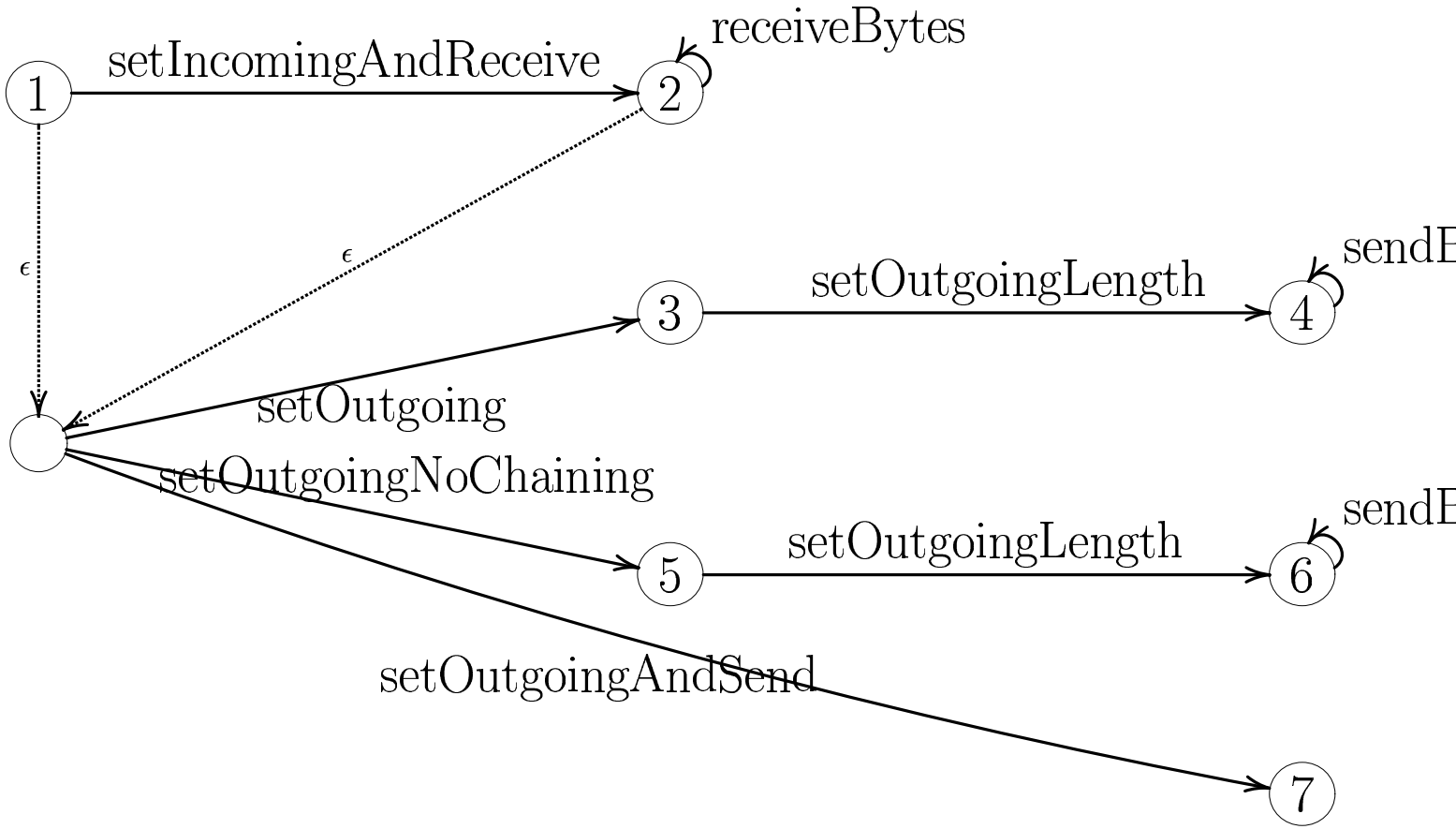
```
incomingFlag, outgoingFlag, outgoingLenSetFlag,
lrIs256Flag, sendInProgressFlag, noChainingFlag,
noGetResponseFlag
```

to enforce invocation order, eg.

```
public short receiveBytes(short bOff) throws APDUException
{ if (!getIncomingFlag() || getOutgoingFlag() )
      APDUException.throwIt( APDUException.ILLEGAL_USE );
   ...
```

but protocol has a lot less than 2^7 states !

FSM for APDU


1
setIncomingAndReceive
2
receiveBytes
ϵ
ϵ
3
setOutgoingLength
4
sendB
setOutgoing
setOutgoingNoChaining
5
setOutgoingLength
6
sendB
setOutgoingAndSend
7

Using a ghost field and FSM to specify APDU

```
//@ public ghost int _APDU_state;

public short setIncomingAndReceive()
 /*@ public behavior
   @    requires _APDU_state == 1 && ... ;
   @     ensures _APDU_state == 2 && ... ;
   @*/


public short receiveBytes(short bOff)
 /*@ public behavior
   @    requires _APDU_state == 2 && ... ;
   @     ensures _APDU_state == 2 && ... ;
   @*/
```

Relating reference implementation to formal spec

Invariants relating the abstract state to its concrete representation, eg:

```
/*@ invariant
  @       _APDU_state == 2
  @  <==>
  @         getIncomingFlag() && !getOutgoingFlag();
  @*/
```

(Checked with ESC/Java)

Fix in Java Card 2.2

In JavaCard 2.2, the APDU protocol is specified as a FSM. There the class includes a method

```
byte getCurrentState()
```

which returns the state of the APDU object, which has the value of one of several constants `STATE_INITIAL`, `STATE_OUTGOING`, `STATE_OUTGOING_LENGTH_KNOWN`, . . .

More detailed JML spec of `receiveBytes(short bOff)`

```
/*@ requires _APDU_state == 2                                  &&
  @          0 <= bOff                                          &&
  @          bOff + getInBlockSize() <= BUFFERSIZE;
  @
  @ assignable _APDU_state, _Lc, buffer[bOff..bOff+\result-1];
  @
  @  ensures _APDU_state == 2                                  &&
  @          0 <= \result && \result <= \old(_Lc)               &&
  @          _Lc == \old(_Lc) - \result                         &&
  @          bOff + \result <= BUFFERSIZE                       &&
  @          (* data received in buffer[bOff..bOff+\result-1] *);
  @
  @  signals (APDUException e) e.getReason() == APDUException.IO_ERROR
  @                          || e.getReason() == APDUException.T1_IFD_ABOR
  @*/
```

Here ghost field _Lc is the length of incoming command.

Bug in reference impl. of `receiveBytes`

The reference implementation does NOT meet this spec, but requires a stronger precondition than

```
bOff + getInBlockSize() <= BUFFERSIZE,
```

namely

```
bOff + getInBlockSize() <  BUFFERSIZE.
```

This is probably a bug.