ISO-8583 SDK 2 for Java and .NET
documentation and description (version 3.50, December 2015)

Table of contents

1. Introduction

This ISO–8583 SDK is designed for financial messaging application developers who need a simple solution to integrate ISO-8583 messaging into their applications like payment terminals and other credit card payment or authorization applications. This SDK is made as simple as possible and has also good performance. Our internal tests show that the parser/formatter is capable do to **about 25000+ parsing/formatting cycles per second** of an average ISO8583 message if with data in memory is used without input and output on an average AMD A10 3,4GHz (jdk 1.7) class desktop single thread/core. The SDK supports the major ISO8583 versions 1987 (version 0, message types 0xxx) and 1993 (version 1, message types 1xxx) and **since SDK version 2.5 support for version 2003(version 2, message types 2xxx) was added**.

Since version 3.5 there is added support to parse packed field in parseplan/formatplain mixed with plain fields depending on field configuration, new packed field types in plain context were implemented

LPVARNUMBERPACKED
NUMBERPACKED
LPVARBINARY
LLPVARBINARY
TRACK2PACKED
LPVARTEXT
LLPVARTEXT
LLXPVARBINARY
LLXPVARTEXT

2. System requirements

In order to use this ISO8583 SDK it is required to have JDK/JRE 1.3 or later version in use. We have not tested the SDK with Java ME but we believe that it could be possible, if some additional JAVA standard classes are made available JAVA ME (but some cases they may not be needed to be added).
For **.NET** users .NET framework **2 or later** is required to use SDK for .net.

3. Features

The most important class of this SDK is **com.a2zss.ISO8583.ISO8583II** that is a class that provides parser and formatter calls. The SDK supports all 128 standard ISO8583 fields including primary and secondary bitmap and other fields up to field 128, Currently the third bitmap (is used in some ISO8583 dialects to support up to 192 fields) is not supported. The SDK supports the major ISO8583 versions 1987 (version 0) and 1993 (version 1) and 2003 (version 2). Parser calls can parse the raw IS08583 data from a input stream or data buffer and will return a Message object that contains all fields that were present in this data/stream.

Additionally the SDK has following advanced features:

- Support for customized field type and length – the developer is able to change the standard parser and formatter to parse/format certain fields in different length and format if such custom format is used within

specified environment in order to support more possible variations/dialects of ISO8583 standard used .
- Support to parse data from input stream or pre buffered data buffer (byte array)
- Support to format data from a constructed Message object based its contents (fields and bitmap objects) and return a data buffer with formatted data that application can write to a stream.
- Support for 4 bit (BCD) packed numeric field encoding (0..9 are presented by half byte values 0000..1001)
- Support for 2 types of bitmaps 8 byte binary (default) and 16 byte hex format bitmaps

The benefits of using ISO8583 SDK version 2 instead of version 1:

1) simplified API
2) multiple instances of parsers/formatters can be used concurrently with different configuration to parse/format different specifications concurrently (in same process (jvm))

4. Key classes and methods

Package com.a2zss.ISO8583 – is the core package that contains core public classes that implements the SDK API methods.
ISO8583II – the main class containing the parser and formatter calls.
Config – the class to change parser/formatter parameters
ISOField – the class to implement the data elements of an Message
Bitmap – the bitmap type field where are bits set of which fields will follow in message
Message – a bundle of ISOFields and Bitmaps
InputStreamBuffer – can be used as virtual input stream if needed to parse pre buffered data but the parser method to parse from buffer is not available.

Key methods of ISO8583II class

Parser/formatter methods:

ISO8583II{instance}.**setConfig**(Config con) – sets the configuration the parser/formatter should use.

ISO8583II{instance}.**parseStream**(java.io.InputStream is) – detects automatically the plain or packed or packed+encoded format and returns the Message, this method can be only used if the messages parsed are without custom headers or footers.

ISO8583II{instance}.**parsePlain**(java.io.InputStream is) - to parse plain (unpacked) ISO8583 message from an input stream
(like something starting with “0100[BITMAP0]”)

ISO8583II{instance}.**parseLLLLPlain**(java.io.InputStream is)- to parse plain (unpacked) ISO8583 message from an input stream with specified length. (like something starting with “01230100[BITMAP0]...”

ISO8583II{instance}.**parseLLPlain**(java.io.InputStream is)- to parse plain (unpacked) ISO8583 message from an input stream with specified length. (like something starting with “LL0100[BITMAP0]...” (where LL is two byte (packed/binary length)

ISO8583II{instance}.**parsePacked**(byte[] in) – to parse packed format of ISO8583 message from and pre buffered data buffer
(packed format is where numeric fields and lengths of variable length fields are compressed two decimal 4 bit numbers in one byte)

ISO8583II{instance}.**parsePacked**(java.io.InputStream is) - same as parsePacked(byte[] in) but parsed data source is inputsream

Formatter methods:

ISO8583II{instance}.**formatPlain**(Message msg) – used to format plain ISO8583 message data

ISO8583II{instance}.**formatLLLLPlain**(Message msg) - used to format plain ISO8583 message data with length

ISO8583II{instance}.**formatLLPlain**(Message msg) – used to format plain ISO8583 message with 2 byte packed length before ISO8583 section. Where the LL – is total length of ISO8583 message section.

ISO8583II{instance}.**formatPacked**(Message msg) – used to format ISO8583 message with 4 bit (BCD) packed numeric fields.

Key methods of Config class

Config{instance}.**setBitmapFormat(boolean hex)**
to set a bitmap format to hex or binary
value true mans 16 byte (hex bitmap format)
value false means 8 byte binary bitmap format

Config{instance}.**setBinpackedlength(boolean bin)**
the default value is false.
this feature is useful if parsing/formatting packed formats where numeric fields and their lengths (LxVARx types) are 4 bit bcd packed. If this feature is set true then the lengths are parsed not as 4 bit BCD packed but as binary packed number data
Example: Decimal number 10 if 4 bit BCD packed is equal to one byte 0x10 hex
but if binary packed then Decimal number 10 is equal to one byte 0x0A hex.

Config{instance}.**setCustomMaxLength**(int bit, int ver, int len) – can be used to customize certain field length, the customization should be done at the startup. Affects the config instance
example: conf.setCustomMaxLength(3, 0, 6);

Config{instance}.**setEbcdic**(boolean ebcdic) – if set true then methods parse**Plain and formatPlain treat the undrlying ISO8583 data in EBCDIC encoded characters
example: conf.setEbecdic(true);

Note: For .net version there is special method:
Config{static}.setLicenseFile(String fn)
Where the fn is license file absolute name. This can be used if current working directory is not known or license file is located outside of current working directory.

Config{instance}.**setCustomType**(int bit, int ver, int type) – used to set a custom type for certain fields, the customization should be done at the startup. The valid values for type are constants defined in Type (FType in C# edition)interface. Example: conf.setCustomType(3, 0, Type.NUMBER);
Available types:
There is total of 16 types available and any of the fields (except bitmaps) can be any of listed types:

Type (FType .NET ed)	*Description*
Type.BINARY	the field is type of BINARY (the field is parsed as fixed length binary content)
Type.LLLLLLVARTEXT	the field is type of LLLLLLVARTEXT (used in some custom ISO formats)
Type.LLLLLVARTEXT	the field is type of LLLLLVARTEXT (used in some custom ISO formats)
Type.LLLLVARTEXT	the field is type of LLLLVARTEXT (used in some custom ISO formats)
Type.LLLVARBINARY	the field is type of LLLVARBINARY (the field is parsed as 3 byte of length and remaining of bytes as binary content)
Type.LLLVARNUMBER	the field is type of LLLVARNUMBER (the field is parsed as 3 byte of length and remaining of bytes as numeric content)
Type.LLLVARTEXT	the field is type of LLLVARTEXT (the field is parsed as 3 byte of length and remaining of bytes as text content)
Type.LLVARBINARY	the field is type of LLVARBINARY (the field is parsed as 2 byte of length and remaining of bytes as binary content)
Type.LLVARNUMBER	the field is type of LLVARNUMBER (the field is parsed as 2 byte of length and remaining of bytes as numeric content)
Type.LLVARTEXT	the field is type of LLVARTEXT (the field is parsed as 2 byte of length and remaining of bytes as text content)
Type.NUMBER	the field is type of NUMBER (the field is parsed as fixed length numeric content)

Type (FType .NET ed)	*Description*
Type.TEXT	the field is type of TEXT (the field is parsed as fixed length text content)
Type.TRACK1 **(NOT USED)**	the field is type of track1 (Note: NOT USED, track 1 is parsed as regular LLVAR use Type.LLVARTEXT instead)
Type.TRACK2	the field is type of track2 (like LLVAR where 1st 2 bytes define the compressed length of contents LLVAR used to parse field as track 2 contents)
Type.TRACK3	the field is type of track3 (like LLVAR used to parse field as track 3 contents)
Type.LVARTEXT (v2.2)	the field is type of LVARTEXT (the field is parsed as 1 byte of length and remaining of bytes as text content)
Type.LVARNUMBER (v2.2)	the field is type of LVARNUMBER (the field is parsed as 1 byte of length and remaining of bytes as numeric content)
LPVARNUMBERPACKED (3.5)	Packed length Packed number within plain parse/format context
NUMBERPACKED (3.5)	Packed number within plain parse/format context
LPVARBINARY (3.5)	Packed length binary plain parse/format context
LLPVARBINARY	Packed length binary plain parse/format context
TRACK2PACKED	Packed length L (1 byte) packed track 2 plain parse/format context
LPVARTEXT	Packed length L (1 byte) packed text plain parse/format context
LLPVARTEXT	Packed length LL (2 bytes) packed text plain parse/format context
LLXPVARBINARY	Exploded Packed length LL (2 bytes) packed binary plain parse/format context
LLXPVARTEXT	Exploded Packed length LL (2 bytes) packed binary plain parse/format context

Note: each L is assumed as decimal number ('0'..'9'), all L's together reflecting the value of length of contents of the corresponding field.
In case of compressed format each L is assumed (one byte) as two 4 bit value 0..9 (half byte).
In case of binary length compressed format each L is assumed one byte binary value.

printConfig(java.io.PrintStream out)
prints out the current configuration values

Packed format enhancements: From version 2.5 update 2,3 there are new options packed messages (when used formatPacked/parsePacked).
Config{instance}.**setPackedTrack2PaddedRightWith0(true|false)**
set true if track2 type data in packed form is padded with half byte '0' at the end instead 0 at the beginning (default behaviour) or instead of the 'F' at the end
Config{instance}.**setPackedTrack2PaddedRightWithF(true|false)**
set true if track2 type data in packed form is padded with half byte 'F' at the end instead 0 at the beginning
Config{instance}.**setPackedTrack3PaddedRightWithF(true|false)**
set true if track3 type data in packed form is padded with F at the end instead 0 at the beginning (default behaviour).

ISOField(int bit, byte[] buf, int v, Config c) – constructor to create the field object (ISOField instance).
Where bit is the field number (generally 1..128) as defined in ISOBits interface,
buf is the data that should be the value of the field contents, and v is the ISO8583 major v is the version used (0 – V1987 or 1 – V1993)- If constructor without version is used then the default version assumed is 0 (V1987),
Config c – is the config instance used with parser formatter.

ISOField{instance}.**setValue(byte[] b)** – sets the binary content value of field instance (object)

ISOField{instance}.**setValue(java.lang.String value)** – sets a string value of field instance (object)

ISOField{instance}.**getStringValue()** and **toString()** - returns the string value (representation) of field instance contents

ISOField{instance}.**getPlainValue()** - returns the field (buffer) contents in same format as it was in parsed message data

ISOField{instance}.**getBinaryValue()** - returns the field (buffer) contents in same original format as it was in parsed message data but without lengths.

Key methods of Bitmap class

Bitmap{instance}.setBit(int bitno, boolean state) – set a specivied bit in bitmap true sets bit to 1, false sets bit to 0
Bitmap{instance}.getBit(int bitno) – returns true if sepcified bit is 1 or false if 0

Key methods of Message class

Message class (object) is used to hold or carry the information containing the iso fields. So to build a message you need to create one or two bitmap objects, set active bits of bitmaps and then add all field objects that bits you set active in bitmap objects.

Message{instance}.**getISOField(**int bit) – returns null (in not present) or a ISOField object included in this message representing the field with specified bit number.

Message{instance}.**putISOField(**java.lang.Object obj) – sets/adds an ISOField object including a bitmap object to a message.

For more detailed information about available classes, constants and methods please see the API javadoc description (http://www.a2zss.com/products/ISO8583API2/index.html). And look the supplied example code.

5. Subfield parsing and formatting utilities:

LLTTVField - Parser and formatter for **LLTTV** type subfields
wich are in format LL - length (includes the tag length and value length)
TT - tag, V - the tag value

LLLTTVField
Parser and formatter for **LLLTTV** type subfields
wich are in format LLL - length (includes the tag length and value length)
TT - tag, V - the tag value

TtLIVField - Parser and formatter for **TtLIV** subfields
(for example used in field 55 ICC Related Data) which are in format
Tt - tag (name) 1 or 2 bytes
Ll - length 1 or 2 bytes 1 byte if length<128 (value length)
V - the tag value

For more detailed information about available classes, constants and methods please see the API javadoc description (http://www.a2zss.com/products/ISO8583API35/index.html). And look the supplied example code.

4. ISO-8583 SDK Default Config

Headers and footers and MTI

FieldNo	Type(maxlen) (v0 19987)	Type(maxlen) (v1 1993, v2 2003)
-4 FF_FOT (ex)	TEXT (0)	TEXT (0) (extended field)
-3 FH_IDENT (ex)	UNDEFINED (0)	UNDEFINED (0) (extended field)
-2 FH_DCP (ex)	UNDEFINED (0)	UNDEFINED (0) (extended field)
-1 MTI (std)	NUMBER (4)	NUMBER (4) message identif required all messages

FieldNo	Type(maxlen) (v0 19987)	Type(maxlen) (v1 1993)
0 bitmap1	BINARY (8)	BINARY (8) required for all messages
1 bitmap2	BINARY (8)	BINARY (8)
2	LLVARNUMBER (19)	LLVARNUMBER (19)
3	NUMBER (6)	NUMBER (6)
4	NUMBER (12)	NUMBER (12)
5	NUMBER (12)	NUMBER (12)
6	NUMBER (12)	NUMBER (12)
7	NUMBER (10)	NUMBER (10)
8	NUMBER (8)	NUMBER (8)
9	NUMBER (8)	NUMBER (8)
10	NUMBER (8)	NUMBER (8)
11	NUMBER (6)	NUMBER (6)
12	NUMBER (6)	NUMBER (12)
13	NUMBER (4)	NUMBER (4)
14	NUMBER (4)	NUMBER (4)
15	NUMBER (6)	NUMBER (6)
16	NUMBER (4)	NUMBER (4)
17	NUMBER (4)	NUMBER (4)
18	NUMBER (4)	NUMBER (4)
19	NUMBER (3)	NUMBER (3)
20	NUMBER (3)	NUMBER (3)
21	NUMBER (3)	NUMBER (3)
22	TEXT (12)	TEXT (12)
23	NUMBER (3)	NUMBER (3)
24	NUMBER (3)	NUMBER (3)
25	NUMBER (2)	NUMBER (4)
26	NUMBER (4)	NUMBER (4)
27	NUMBER (1)	NUMBER (1)
28	NUMBER (6)	NUMBER (6)
29	NUMBER (3)	NUMBER (3)
30	NUMBER (24)	NUMBER (24)
31	LLVARTEXT (99)	LLVARTEXT (99)
32	LLVARNUMBER (11)	LLVARNUMBER (11)
33	LLVARNUMBER (11)	LLVARNUMBER (11)
34	LLVARNUMBER (28)	LLVARNUMBER (28)
35	TRACK2 (37)	TRACK2 (37)
36	TRACK3 (104)	TRACK3 (104)
37	TEXT (12)	TEXT (12)
38	TEXT (6)	TEXT (6)
39	TEXT (2)	NUMBER (3)
40	NUMBER (3)	NUMBER (3)
41	TEXT (8)	TEXT (8)
42	TEXT (15)	TEXT (15)
43	TEXT (40)	LLVARTEXT (99)
44	LLVARTEXT (99)	LLVARTEXT (99)
45	LLVARTEXT (76)	LLVARTEXT (76)
46	LLLVARTEXT (204)	LLLVARTEXT (204)
47	LLLVARTEXT (999)	LLLVARTEXT (999)
48	LLLVARTEXT (999)	LLLVARTEXT (999)
49	NUMBER (3)	NUMBER (3)
50	NUMBER (3)	NUMBER (3)
51	NUMBER (3)	NUMBER (3)

```
52    BINARY (8)              BINARY (8)
53    LLVARBINARY (48)        LLVARBINARY (48)
54    LLLVARTEXT (120)        LLLVARTEXT (120)
55    LLLVARBINARY (255)            LLLVARBINARY (255)
56    LLVARNUMBER (35)        LLVARNUMBER (35)
57    NUMBER (3)              NUMBER (3)
58    LLVARNUMBER (11)        LLVARNUMBER (11)
59    LLLVARTEXT (999)        LLLVARTEXT (999)
60    LLLVARTEXT (999)        LLLVARTEXT (999)
61    LLLVARTEXT (999)        LLLVARTEXT (999)
62    LLLVARTEXT (999)        LLLVARTEXT (999)
63    LLLVARTEXT (999)        LLLVARTEXT (999)
64    BINARY (8)              BINARY (8)
65    BINARY (8)              BINARY (8)
66    LLLVARTEXT (204)        LLLVARTEXT (204)
67    NUMBER (2)              NUMBER (2)
68    NUMBER (3)              NUMBER (3)
69    NUMBER (3)              NUMBER (3)
70    NUMBER (3)              NUMBER (3)
71    NUMBER (8)              NUMBER (8)
72    LLLVARTEXT (999)        LLLVARTEXT (999)
73    NUMBER (6)              NUMBER (6)
74    NUMBER (10)             NUMBER (10)
75    NUMBER (10)             NUMBER (10)
76    NUMBER (10)             NUMBER (10)
77    NUMBER (10)             NUMBER (10)
78    NUMBER (10)             NUMBER (10)
79    NUMBER (10)             NUMBER (10)
80    NUMBER (10)             NUMBER (10)
81    NUMBER (10)             NUMBER (10)
82    NUMBER (10)             NUMBER (10)
83    NUMBER (10)             NUMBER (10)
84    NUMBER (10)             NUMBER (10)
85    NUMBER (10)             NUMBER (10)
86    NUMBER (16)             NUMBER (16)
87    NUMBER (16)             NUMBER (16)
88    NUMBER (16)             NUMBER (16)
89    NUMBER (16)             NUMBER (16)
90    NUMBER (10)             NUMBER (10)
91    NUMBER (3)              NUMBER (3)
92    NUMBER (3)              NUMBER (3)
93    LLVARNUMBER (11)        LLVARNUMBER (11)
94    LLVARNUMBER (11)        LLVARNUMBER (11)
95    LLVARTEXT (99)          LLVARTEXT (99)
96    LLLVARBINARY (999)            LLLVARBINARY (999)
97    TEXT (17)               TEXT (17)
98    TEXT (25)               TEXT (25)
99    LLVARTEXT (11)          LLVARTEXT (11)
100   LLVARNUMBER (11)        LLVARNUMBER (11)
101   LLVARTEXT (17)          LLVARTEXT (17)
102   LLVARTEXT (28)          LLVARTEXT (28)
103   LLVARTEXT (28)          LLVARTEXT (28)
104   LLLVARTEXT (100)        LLLVARTEXT (100)
105   NUMBER (16)             NUMBER (16)
106   NUMBER (16)             NUMBER (16)
107   NUMBER (10)             NUMBER (10)
108   NUMBER (10)             NUMBER (10)
109   LLVARTEXT (84)          LLVARTEXT (84)
110   LLVARTEXT (84)          LLVARTEXT (84)
111   LLLVARTEXT (999)        LLLVARTEXT (999)
112   LLLVARTEXT (999)        LLLVARTEXT (999)
113   LLLVARTEXT (999)        LLLVARTEXT (999)
114   LLLVARTEXT (999)        LLLVARTEXT (999)
115   LLLVARTEXT (999)        LLLVARTEXT (999)
```

```
116   LLLVARTEXT (999)        LLLVARTEXT (999)
117   LLLVARTEXT (999)        LLLVARTEXT (999)
118   LLLVARTEXT (999)        LLLVARTEXT (999)
119   LLLVARTEXT (999)        LLLVARTEXT (999)
120   LLLVARTEXT (999)        LLLVARTEXT (999)
121   LLLVARTEXT (999)        LLLVARTEXT (999)
122   LLLVARTEXT (999)        LLLVARTEXT (999)
123   LLLVARTEXT (999)        LLLVARTEXT (999)
124   LLLVARTEXT (999)        LLLVARTEXT (999)
125   LLLVARTEXT (999)        LLLVARTEXT (999)
126   LLLVARTEXT (999)        LLLVARTEXT (999)
127   LLLVARTEXT (999)        LLLVARTEXT (999)
128   BINARY (8)              BINARY (8)
```

Notes: maxlen reflects the maximum length allowed for variable length fields or the actual fixed length of fixed length fields.
All field lengths and types can be customized using program calls except bitmaps are default BINARY (8) and other possible is Hex bitmaps where they are actually TEXT (16). It could be customized using method setBitmapFormat(true) to hex text (16) or
setBitmapFormat(false) binary (8) that is the default setting.
Types TRACK2 and TRACK3 are special cases of variable length fields.

Supported field types:

```
Type 0      NUMBER
Type 1      TEXT
Type 2      LLVARNUMBER
Type 3      LLVARTEXT
Type 4      LLLVARNUMBER
Type 5      LLLVARTEXT
Type 6      BINARY
Type 7      LLVARBINARY
Type 8      LLLVARBINARY
Type 9      TRACK2
Type 10     TRACK3
Type 11     TRACK1
Type 12     LLLLVARTEXT
Type 13     LLLLLVARTEXT
Type 14     LLLLLLVARTEXT
Type 15     LVARNUMBER
Type 16     LVARTEXT
Type 17     LPVARNUMBERPACKED
Type 18     NUMBERPACKED
Type 19     LPVARBINARY
Type 20     LLPVARBINARY
Type 21     TRACK2PACKED
Type 22     LPVARTEXT
Type 23     LLPVARTEXT
Type 24     LLXPVARBINARY
Type 25     LLXPVARTEXT
Type 26     -
Type 27     -
Type 28     -
Type 29     -
Type 30     CDNUMBER
```