Answers as of 9/20/2001 from J Consortium to Japanese ISO/IEC committee About Questions on JEFF PAS submission DIS 20970

NB: original comments and questions in **black**, answers in **blue**.

Major Comments

1) Clarify the relationship with Sun Microsystems Inc.'s Java specifications.
The document is a self-contained specification of the syntax of JEFF format that does not use any specifications from Sun as normative references. This format can be used to distribute and store ready-for execution programs written with various object-oriented languages. In the document, as an illustration, is provided some indications on how to translate Sun's Java class file byte code instructions into JEFF. The references to Sun's specifications are given only as example references, not normative ones. It has been established that other languages not originating from Sun can be as well represented with JEFF.

2) Describe the contents clearly and understandable. Define terms precisely.
Yes, we have made an attempt to address your feedback more clearly and precisely. Please find our detailed answers to your specific feedback below. We would be pleased to work with you to provide further details.

3) Provide the precise information as an International Standards.

The JTC 1 Directives do not require an initial PAS submission to meet the ISO/IEC format.

From clause 18.4.2 of the JTC 1 Directives
"18.4.2 PAS Submission
...
JTC 1 does not regulate the format of the specification submitted. However, Recognized PAS Submitters are encouraged to apply, if flexibility still exists, a documentation style close to the ISO/IEC style in order to ease the later alignment process at the time of any revision."
...

As required by JTC 1, the JEFF standard will be presented in the format prescribed by the ISO/IEC Directives at the time of any revision.

If a ballot resolution meeting is required, then we will provide the following information in the correct format for consideration by the ballot resolution group for possible inclusion in the JEFF standard.

1. Introduction
    The reason and purpose of this standard
2. Scope
    Who uses this standard?
3. Reference
4. Definition
    terminology

Minor Comments:

1) Follow the ISO/ICE JTC 1 standard format so that the necessary information will be presented precisely.

Yes, see answer for General Question # 3.

2) The meta-syntax of structure notation is poorly specified and is hardly understandable. Though it is mentioned as "C-like," there are apparent differences with the C structure. For example, unlike C, item offsets within the structure are not constant (see 3.2.3). Offsets depend on the position of each structure instance.

Some items are "optional" in some structures. This means that these items are, at times, completely removed from the structure. So far we looked through the specification, nothing is apparent from the structure notations, whether the item is optional or not.

Yes, The “pseudo-C” structure notation can be clarified by the following enhancements.

- Explicit Padding

Every structure including padding should declare it explicitly:

E.g.

“TU1 <0-3 byte pad>
TU4 nValue

Zero to three bytes set to zero to insure that first byte of nValue is aligned on a four-byte boundary.”

- Explicit Separation of the Cases for the Optional Elements

The structures should not include optional elements. This can be done by presenting different structures for each case.

E.g.
The class header structure VMClassHeader is used for both the classes and the interfaces. If the structure is used for an interface, the last three items are not present. A more explicit explanation would be:

“For the classes, the class area header has the following structure:

```
VMClassHeader {
VMOFFSET ofThisClassIndex;
VMPINDEX pidPackage;
VMACCESS aAccessFlag;
TU2 nClassData;
VMOFFSET ofClassConstructor;
VMOFFSET ofInterfaceTable;
VMOFFSET ofFieldTable;
VMOFFSET ofMethodTable;
VMOFFSET ofReferencedFieldTable;
VMOFFSET ofReferencedMethodTable;
VMOFFSET ofReferencedClassTable;
VMOFFSET ofConstantDataSection;
VMOFFSET ofSuperClassIndex;
TU2 nInstanceData;
VMOFFSET ofInstanceConstructor;
}
```

For the interfaces, the class area header has the following structure:

```
VMClassHeader {
VMOFFSET ofThisClassIndex;
VMPINDEX pidPackage;
VMACCESS aAccessFlag;
TU2 nClassData;
VMOFFSET ofClassConstructor;
VMOFFSET ofInterfaceTable;
VMOFFSET ofFieldTable;
VMOFFSET ofMethodTable;
VMOFFSET ofReferencedFieldTable;
VMOFFSET ofReferencedMethodTable;
VMOFFSET ofReferencedClassTable;
VMOFFSET ofConstantDataSection;
}
```

The items of the **VMClassHeader** structure are as follows: …”

- Clear Definition of the Lists and Arrays

The following definition must be added at the beginning of the document, with the explanations on the “pseudo-C” structures:

“This document contains notations to represents lists and arrays of elements. An array or a List is the representation of a set of several consecutive structures. In an array, the structures are identical with a fix size and there are no padding bytes between them. In a list, the structures may be of variable length and some padding bytes may be added between them. When a list is used, the corresponding comments precise the length of each structure and the presence of padding bytes”

- Explicit Declaration of the Count of Elements in the Lists and in the Arrays

Every list and every array must be defined with an explicit count of elements.

E.g.

The constant string structure:

```
“VMConstUtf8 {
TU2 nStringLength;
TU1 nStringValue[];
}”
```

Should be replaced by:

```
“VMConstUtf8 {
TU2 nStringLength;
TU1 nStringValue[nStringLength];
}”
```

3) There are lots of ambiguous or hard to understand expressions. The followings are some examples we found in the text:

p2-2.2 Definition of <VMConstUTF8> of <Language Types>

This has different syntax from the rest of the specification.
So far we guess -- see comments on "p8-3.2.1 <Notations>" --
array size had better be explicitly specified.

"TU1 nStringValue[];" ==> "TU1 nStringValue[nStringLength];"

Yes, cf. answer to 2)

p4-2.3.2 <Dimension Value> of <Type Descriptor>
Explanations for <Dimension Value> are not understandable. Same kind of problems are with the following three locations:
p18-3.3.2.6(cont'd) <nTypeDimension>
p27-3.3.5.2 <nDataTypeDimension> of <Descriptor>
p32-4.2.6 <translated structure> of <The multinewarray Opcode>
A list of the possible code may be more easy to understand:

Type	Type value	Dimension	Class Index
void	0x00	0 or absent	absent
short	0x01	0 or absent	absent
int	0x02	0 or absent	absent
long	0x13	0 or absent	absent
byte	0x04	0 or absent	absent
char	0x05	0 or absent	absent
float	0x06	0 or absent	absent
double	0x17	0 or absent	absent
boolean	0x08	0 or absent	absent
reference	0x0A	0 or absent	index of the instance class
short[]	0x61	1 or absent	absent
int[]	0x62	1 or absent	absent
long[]	0x63	1 or absent	absent
byte[]	0x64	1 or absent	absent
char[]	0x65	1 or absent	absent
float[]	0x66	1 or absent	absent
double[]	0x67	1 or absent	absent
boolean[]	0x68	1 or absent	absent
reference[]	0x6A	1 or absent	index of the instance class

short[][][]…	0x81	dimension	absent
int[][][]…	0x82	dimension	absent
long[][][]…	0x83	dimension	absent
byte[][][]…	0x84	dimension	absent
char[][][]…	0x85	dimension	absent
float[][][]…	0x86	dimension	absent
double[][][]…	0x87	dimension	absent
boolean[][][]…	0x88	dimension	absent
reference[][][]…	0x8A	dimension	index of the instance class

p7-3.1.3 <Internal Classes and External Classes>, and
p7-3.1.4 <Fields and Methods>

The name sorting order specified as "following the crescent lexicographic order" is ambiguous.

Even if it is assumed that no correlation table is required, above statement can still be interpreted as:

a) by 16-bit Java char code comparison,
b) by 32-bit Unicode character code comparison, or
c) by UTF8 byte code comparison.

Which is correct?

In the latest case, it is also required that the UTF8 strings stored in the file structure are canonicalized.
The specification should state:
“following the ascending lexicographic order, i.e. the ascending16-bit Unicode character order of the symbolic names.”.

p7-3.1.4 <Fields and Methods>

"All the symbolic names representing the internal class fields are stored at the beginning of the table."

Above statement conflicts with the deduction that follows the statement.

The indexation algorithm described in the text should be reformulated as follows:

The field indexes are computed as follows:
Let n be the number of different symbolic names associated to the internal class fields
1 - The symbolic names of the internal class fields are indexed according to their ascending lexicographic order, with index increment of 1, indexes ranging from zero up to n-1.
2 - The symbolic names of the external class fields that are not also symbolic names of internal class fields are indexed according to their ascending lexicographic order, with index increment of 1, starting at n.

In this algorithm, the ascending lexicographic order is the ascending 16-bit Unicode character order of the symbolic names.

p9-3.2.3 <Alignment and Padding>

<VMPINDEX> is defined in 2.3, but not in this table.

This type should be added

p9-3.3 table of <The File Structure>
It is not clear what are optional, and how optional in "Optional Attribute Section" and "list of optional attributes."
This section should be renamed "Attribute Section". Then the definition should be:

"Attributes Section
This optional section contains the attributes for the file, the classes, the methods and the fields."

p13-3.3.2.1 <aAccessFlag> of <Class Header>

These are said to be "values." They should be "bit-vectors."

Yes

p17-3.3.2.5 <ofCode> of <Internal Method Table>

This implies each <ofCode> must have different <nNativeReference> offsets AND EVERY <nNativeReference> MUST HAVE UNIQUE VALUES.

Is the latter restriction really necessary? Isn't it too much restrictive?

In <ofCode> of <Internal Method Table>, the last sentence is misleading and should be replaced by:
For a native method, the value is the offset of one of the nNativeReference values. Each native method must have a different ofCode value.

p19-3.3.2.8 <ncMaxLocals> of <Bytecode Block Structure>

Are "cell" and "word" the same or not?
Yes, “word” should be replaced by “cell”.

p19-3.3.2.8 <bytecode> of <Bytecode Block Structure>

Are "address" and "offset" the same or not?
Here, “address” should be replaced by “16-bit offset from the beginning of the class section”.

p27-3.3.6 <File Signature> (or <Digital Signature>?)

> "The VMFileSignature structure is not defined."
>
> Does this mean that, in this version of specification, <File Sinature> must not be used and the offset value must be zero?
>
> Or, does this mean that suppliers may define their own structure?

The sentence “The VMFileSignature structure is not defined.”

Should be replaced by the following equivalent definition:

“3.3.6 Digital Signature

The JEFF specification does not impose any algorithm or any scheme for the signature of a JEFF file. The digital signature of the JEFF file is stored in a VMFileSignature structure defined as follows:

```
VMFileSignature {
  TU1  nSignature[];
}
```

where the byte array nSignature contains the signature data. The length of the array can be deduced from the position of the VMFileSignature structure and the total size of the JEFF."

p32-4.2.6(cont'd) <nDimensions> of <translated structure> of <The multianewarray Opcode>

Explain should be put precisely. Is this item necessary?? There is <nArrayDimensions> item that also specifies array dimension.

"nDimensions" has the same value as the "nDimensions" of the original structure. While "nArrayDimensions" is defined by the dimension of descriptor identified by "nIndex" in the original structure. These two values may be different.

p34-4.2.10 <nIndex> of <The wide <opcode> Opcodes>

None explained. Or, should we refer the description of original instruction?

Yes, nIndex has the same value than in the original instruction.

p35-4.2.11 <nIndex> and <nConstant> of <The wide iinc Opcode>

None explained. Or, should we refer the description of original instruction?

Yes, nIndex has the same value than in the original instruction.

Editorial Comments

p3-2.3.1 <Meaning> of <ACC_PUBLIC> of <Class> of <Access Flags>

"... from outside its package." ==>
"... from outside of its package."

Yes

p3-2.3.1 <Meaning> of <ACC_PUBLIC> of <Field> of <Access Flags>

"... from outside its package." ==>
"... from outside of its package."

Yes

p5-2.3.2 <Class Index> of <Type Descriptor>

"... array of class, ..." ==> "... array of a class, ..."

Yes

p5-2.3.3 <Offsets> 2nd paragraph

"... Depending of where ..." ==> "... Depending on where ..."

Yes

p8-3.1.5 (cont'd) 3rd bullet

"... Depends of the ..." ==> "... Depends on the ..."

Yes

p.11-3.1.1(cont'd) <nByteOrder>

"the term "integer value" designs ..." ==>
"the term "integer value" defines ..."

"... two-, four- and height-bytes ..." ==>
"... two-, four- and eight-bytes ..."

Yes

p17-3.3.2.5(cont'd) <nMethodCount> of <Internal Method Table>

"... number of method ..." ==> "... number of methods ..."

Yes

p19-3.3.2.8 <ncMaxStack> of <Bytecode Block Structure>

"... during execution of ..." ==>
"... during the execution of ..."

Yes

p20-3.3.2.9 <nCatchCount> of <Caught Exception Table>

"... number of element ..." ==> "... number of elements ..."

Yes

p20-3.3.2.10 <Constant Data Section>

"... They are always referred through an offset. ..." ==>
"... They are always referred through offsets. ..."

Yes

p23-3.3.3(cont'd) <dofClassAttrubutes> of <Attrubute Section>

"... value iszero if ..." ==> "value is zero if ..."

Yes

p26-3.3.5.1 <nDescriptorCount> of <Constant Data Pool Section>

"... take in account the ... descriptors." ==>
"... take the ... descriptors into account."

Yes

p27-3.3.5.3 <nArgCount> of <Method Descriptor>

"... 0 for a method without argument." ==>
"... Zero for methods without arguments."

Yes

p28-4 <Bytecodes>

"... instruction without argument. ..." ==>
"... instruction without arguments. ..."

Yes

p29-4.2.1 <The tableswitch Opcode>

"... converted in 16-bit signed value, ..." ==>
"... converted into 16-bit signed values, ..."

Yes

p29-4.2.2 <The lookupswitch Opcode>

"TU4 nPairs" ==> "TS4 nPairs"

or

"... singed 32-bit values: nDefault, nPairs, ..." ==>
"... singed 32-bit value nDefault, unsigned 32-bit value nPairs, ..."

Yes, "nPairs" is an unsigned value

p31-4.2.4(cont'd) <ofClassIndex> of <Opcodes With Class Arguments>
"... a class or an array of classes."
==> "... a class or an array of a class."

Yes

p32-4.2.6(cont'd) <ofClassIndex> of <The mulitanewarray Opcode>
"... a class or an array of classes."
==> "... a class or an array of a class."

Yes

p36-4.2.14 <original instruction> of <The sipush Opcode>

"TU1 nByte1" ==> "TS1 nByte1"

Yes

p36-4.2.15 <The newconstarray Opcode>

"... depends of ..." ==> "... depends on ..."

Yes

"TU1 nByte1" ==> "TS1 nByte1"

??? There is no “TU1 nByte1” in this part.

p43-5

"Restriction" ==> "Restrictions"

Yes

==

VOTES ON ISO/IEC DIS 20970

Date 2001/11/31 ?ISO/IEC/JTC 1

The National Body of Japan (JISC)

To cast a vote on a draft International Standard, national bodies shall complete and sign this ballot paper, and return same with any comments to the ISO Central Secretariat.

All national bodies are invited to vote.
P-members of the joint technical committee concerned have an obligation to vote.

( ) We approve the technical content of the draft as presented

(editorial or other comments may be appended)

(x) We disapprove for the technical reasons stated at annex

(x) Acceptance of specified technical modifications will change our vote to approval

( ) We abstain (for reasons below)

Remarks:
Major Comments:

1) Clarify the relationship with Sun Microsystems Inc.'s Java specifications.

2) Describe clearly and understandable. Define terms precisely.

3) Provide the precise information for International Standards.

Note 1 – Reference to ISO standards
The specification will be updated to refer the ISO standards (specifications and notations)
Utf8 will be replaced by UTF-8
The character codes and the character ranges will be written with ISO notations in the next revision as part of the maintenance process.

"Next revision" is not good enough. Also for the first edition correct character codes, and correct notation for them, must be used.

References to UTF8 and UTF16 were removed and replaced with string definition resolving the issues raised by JISC. See new section 2.3 in the new revision.

Note 2 – Character Encoding
The JEFF specification does not contain any data represented with the format "JCHAR".

So there is no way of representing values of (Java) type 'char' in JEFF? Strange!

The reference to the "JCHAR" type will be removed (clause 2.2 and 3.2.3). Thus, the UTF-16 format in not used in the specification.
JCHAR is not used in class files and was not referred to in the JEFF Specification. It has been removed.

MB	Index		Comment	Answer from J Consortium
Sweden	1. MAJOR clause 2.2	te	JCHAR: Change "16- bit Unicode char" to "UTF-16 code unit"; change "Unicode max." to "0xFFFF". Note that the code point space in Unicode ranges from U+ 0000 to U+ 10FFFF, while the largest UTF-16 code unit is 0xFFFF. To represent characters in the supplementary planes (plane 01 to 10) in UTF- 16, two UTF- 16 code units are used per encoded supplementary character.	The line of the array will be removed (see note 2). Very strange response! Are JEFF interpreters not going to handle characters at all? If it can handle ints, longs, etc. chars should also be handleable. Class files do not contain characters, they only contain strings. And JEFF standard only specifies strings.
Sweden	2A MAJOR clause 2.2	te	The byte (octet, see below) order of a JEFF file must be fixed to so- called network octet order, i. e. big-endian. Having endianness depend on bits in a header makes JEFF directly **unsuitable** as a direct-execution format	The choice of the byte order allows choosing the most efficient representation for a direct execution. Most available processors are not using big-endian and a fixed big-endian order in the format would impose a byte per byte reading and a loss of time in execution. It is perfectly normal to require fixed byte order, and to require so-called "network byte order". The argument presented above is very unpersuasive, since the bytecodes have to be read byte (octet) by byte anyway. It is a requirement for JEFF files to be efficiently executable directly from ROM; therefore, the file byte order and alignment must match the processor. In addition, JEFF files are not required to be read byte by byte.
Sweden	2B MAJOR clause 2.2	te	Further, there is reference to a "global byte order", but no such byte order is defined. Instead there is byte order meddling per datatype (except for characters...) in the JEFF specifications, using bits in a header that is not needed for a properly defined program file format.	Yes, a direct reference to the value of the "nByteOrder" item of the clause 3.3.1 will replace the "global byte order" Please delete all byte order toggles! They are an unnecessary complication, and helps nothing. See answer to 2A. All characters are represented using the UTF-8 encoding (no byte order is required with this format).

				All constant strings should still be represented in UTF-16, which is the representation of choice in Java (as well as the Win32 APIs and Mac APIs). See new definition in section 2.3.
Sweden	3 MAJOR clause 2.2	te	Strings in Java are sequences of UTF-16 code units (called char in Java). They must be (logically) strings of UTF-16 code units also during runtime, so it seems needlessly cumbersome, and highly inefficient, to actually represent strings in UTF- 8 during runtime. (UTF-8 is spelled so, NOT "Utf8".)	Yes for the spelling of UTF-8 (see note 1) The JEFF specification does not target any specific runtime implementation. I'm not sure what that means, and how it would be relevant to the comment. Concerning Java, most of the identifiers must be written using the first 127 characters of Unicode. It is definitely NOT TRUE that "most identifiers MUST be" so. There is no such limitation in Java. Most identifiers in the Java standard libraries happen to use just ASCII, but that is only because the names are mnemonic for English, not due to any other limitations. Besides, this was about strings, not identifiers. Thus, UTF-8 provides a good compression format. UTF-8 is not a compression format! See new definition in section 2.3.
Sweden	4 MAJOR clause 2.2	te	The UTF- 8 format is defined by ISO/ IEC 10646 - 1 (not yet fully up to date) and Unicode 3. 1 (further, Unicode 3.2 is likely to make the irregular cases illegal). It's NOT defined by the Java Virtual Machine Specification.	Yes, the specification will be updated to refer the ISO standards See note 1 See answer to 3.
Sweden	5 MAJOR clause 3.1.1 and 3.1.2	te	The encoding for names is not given, which it should be. Is it UTF- 16? Or (proper) UTF- 8?	These names are not represented in the format. Thus they don't need to be encoded. You have to decide whether you represent identifiers (names) in UTF-8 (as you say above) or if they aren't represented (which is strange in this case!). All names are encoded in JEFF files as type VMstrings.
Sweden	6 clause 3.1.1	Te	Change "Unicode 0x002F" to "U+ 002F, SOLIDUS".	Yes, the specification will be updated to use ISO notations See note 1 See answer to 3.
Sweden	7 clause 3.1.2	Te	Change "Unicode 0x002F" to "U+ 0020, SPACE".	Yes, the specification will be updated to use ISO notations See note 1 See answer to 3.
Sweden	8 MAJOR	ge	use the term 'octet' instead of 'byte' since 'byte' is sometimes used in a looser sense.	The specification follows th e most common terminology and uses the word "byte" with same meaning than "octet". What is the proper word in ISO terminology: byte or octet? If this is "octet" we can make a preliminary vocabulary statement warning that the word "byte" in the specificatio n is always used a synonymous of "octet". The word "byte" is used in standards like C, where its size is not completely determined (but nowadays at least 8 bits; but it can be more). Octet is always exactly 8 bits, much better suited for a modern platform-independent format. Yes, this is added in the new Terminology section.
Sweden	9 MAJOR clause 3.2.2 and 3.3.1	te	always use so- called network octet order, i. e big-endian. (That implies that that header fields in JEFF is not needed.)	No, see 2A Yes. See above. JEFF requires byte order, see 2A.
Sweden	10 MAJOR	te	the VM_ USE_ UNICODE bit shall always be set	Yes

	clause 3.3.1		(i. E the bit is not needed in the file format...).	
Sweden	11 MAJOR	ge	replace "bytecode" with "octet" and replace "bytecodes" with "directly executable instructions".	The following definitions will be added: A "bytecode" is the binary value of the encoding of a JEFF instruction. By extension, "bytecode" is used to designate the instruction itself. Please follow the comment gi ven. Bytecode is not an octet, See the definition in the JEFF standard.
Sweden	12 MAJOR clause 3.3.2.10	te	Utf8; see above	See note 1 and note 2
Sweden	13 MAJOR clause 3.3.3.1	te	byte order; see above	See 2A
Sweden	14 clause 3.3.4	te	Utf8; see above	See note 1 and note 2
Sweden	15 MAJOR clause 5	te	"cannot exceed 64Kb"?? 64 Kelvin- bit? I guess you mean 64 KiB, kilo- binary octets (" kibibyte"). Similarly, Gb (gibabit?) should really be GiB (giga- binary octets, "gibibyte"). And expand "4 Giga" to the a ppropriate numeric expression.	64Kb will be replaced by 65536 bytes (or binary octets) 4 Giga will be replaced by 4294967296 bytes (or binary octets) KiB and GiB are perfectly ok (with the numbers spelled out in parentheses). Octets are binary octets. There is no terminology for non-binary octets. (You should parse the comment as "(kilo-binary) octets". Kilobinary is 1024, whereas kilo is 1000. The units GiB, MiB, and KiB are from IEC 60027-2. In response to the ANSI remarks, this has been changed.
Sweden	16 MAJOR	ge	self- containedness: opcodes are given, but they are not given any semantics, except by vague reference to the class file format (bibl. ref. [1]). This specification is thus NOT self- contained contrary to claims. These claims must thus be removed.	The semantics of the opcodes and instructions is the concern of a virtual machine specification. The JEFF specification is not concerned with this operational semantics aspect, Thus, it is not self-contained. only by the structural description of a format, and, as a format specification, is self-contained. Without the interpretation, the format is pointless. If JEFF were used to provide Java programs, the semantics defined for the Java virtual machine specifications would apply to the opcodes, So, it is a metaformat? How can you then give ANY opcodes? Caluse 4 is full of them. but this is a pragmatic consideration outside of JEFF specification as well as would be the strict definition of which programming language JEFF could be used to e ncode. The syntax description of the JEFF instruction will be reworded to be more self-descriptive. JEFF only specifies a file format. Operational semantics are beyond the scope of this standard.
Sweden	17 MAJOR	te	self- containedness: opcodes are given, but they are not given any semantics, except by vague reference to the class file format (bibl. ref. [1]). This specification is thus NOT self- contained contrary to claims. These claims must thus be removed.	See 16 See above.

Sweden	18 MAJOR	ge	it is stated that the given format "can be read as grammar"; please give a properly formulated grammar in the conventional style then.	The use of a formal grammar notation like the BNF for instance is useful when defining recursive structures of which intermedia te elements can be found in the decomposition of various higher -level elements. This is not so much the case with a format like JEFF and the systematic expression in BNF formalism would not add to the readability of the specification. However, for presenting a synthetic view of the specification, a BNF formulation of the main articulation of a JEFF file will be added in appendix. So why bother with the unreadable(!) “ungrammar” at all? Yes, correct. A BNF will not be provided.
Sweden	19 MAJOR	te	Other missing normative specifications are (in a "Normative references" clause that is currently missing): ISO/ IEC 10646- 1: 2000 (defines UTF-8 and UTF- 16), and its amendment 1 (2002), as well as future amendments, ISO/ IEC 10646- 2: 2001, and its future amendments, The Unicode Standard, versions, 3. 0, 3.1, and 3.2, and future versions (use the definitions of UTF- 16 and UTF-8 from this reference, 3.1 in particular, since this maximises interoperability).	Yes, the specification will be updated to refer the ISO standards See note 1
Sweden	20 MINOR	ge	There are no "Scope" and "Terms and definitions" clauses. The "Overview" should be made into non-normative (and unnumbered) "Introduction" section.	Yes, these clauses and sections will be considered in the next revision as part of the maintenance process. Do it now. There is no reason to wait. Yes, correct. These clauses are added. JISC also requested this addition.

Answer to Rex Jaeschke ‘s comments on the JEFF Specifications (August 8 2001)

By Jean-Bernard Blanchet (jean-bernard.blanchet@cardsoft.com)

Notes on the document:

Summary of Comment on the JEFF Specifications jeff_spec_1.0.doc
(August 2, 2001)
NCITS/CT22
By Rex Jaeschke (rex@RexJaeschke.com)

Page 5
Note 1 OK

Page 6
Notes 1, 2, 3, 4, 5, 6, 7 OK

Note 8
OK If IEC 559 defines the same format as IEEE 754-32 and IEEE 754-64

Notes 9, 10, 11
The two columns “max value” and “min value” do not provide any additional information. They should be removed.

Notes 12
The note on the fp format is useless. It should be removed.

Note 13 OK

Page 7

Note 1 OK

Note 2 OK

Note 3 OK (and Virtual Machine Cell is note defined)
I propose :
“The VMNCELL Type is an unsigned two-byte integer interpreted as an index in an array of U4 values.”

Note 4 OK

Note 5 OK
I propose:
“Modify the behavior of the jeff_invokespecial bytecodes included in the bytecode area list of this class.”

Page 8

Note 1
The definition is given 2.3

To limit the use of forward reference, I propose to use the following order for the definitions in the paragraph 2.3 :

VMACCESS
VMNCELL
VMOFFSET
VMDOFFSET
VMCINDEX
VMPINDEX
VMMINDEX
VMFINDEX
VMTYPE

Notes 2, 3, 4, 5, 6, 7, 8, 9, 10, 11   OK

Note 12
The void return type is never used in an array type.

Page 9

Note 1 OK

Note 2
The examples are not normative. They are just an illustration of the above explanations.

Note 3
The following note should be added:

“The presence or the absence of the optional elements of a type descriptor is explicitly specified everywhere a type descriptor is used in the specification.”

Note 4
I propose the following definition:

“A VMOFFSET is an unsigned 16-bit value located in a class area section (See 3.3.2). This value is an offset in bytes from the beginning of the class header of the class area section.”

Notes 5, 6   OK

Page 10

Notes 1, 2 OK

Notes 3, 4
I propose the use of a non-italic constant-width font for all the string examples

Note5
This is explained in the line above. Anyway the string should be displayed in a single line.

Page 11

Notes 1, 2 OK

Note 3
This table represents the class index range, not the package index range.

Note 4 OK

Note 5 “speedup” should be replaced by “speed up”

Note 6 OK

Page 12

Note 1 OK

Note 2
I propose:
“The fields are ordered in a list”

Notes 3, 4 OK

Page 13

Notes 1, 2, 3, 4   OK

Note 5
I propose

“Definition of the File Structures”

Note 6
This should be replaced by:
“The file structure is composed of six sections ordered as follows:”

Page 14

Note 1 OK

Note 2
The sentence:
“The class section describes the content of each class (inheritance, fields, methods and code).”
Should be replaced by
“The class section describes the content and the properties of each class”

Notes 3, 4   OK

Page 15

Note 1
I propose to replace the sentence with
“A set of information describing some properties of the internal classes.”

Page 16

Note 1

The following lines

“3.3.2 Class Area
For each class included in the file, a class area contains the information specific to the class.”

Should be replaced by

“3.3.2 Class Section
For each class included in the file, a class area contains the information specific to the class. The Class Section contains these class areas stored consecutively in an ordered list following the crescent order of the corresponding class indexes.”

Page 21

Note 1
I propose the following definition

“This array of TU4 values contains as many elements as the class has native methods. To each TU4 value corresponds one and only one native method of the class. The TU4 values are stored following the order of storage of the corresponding VMMethodInfo structure. The TU4 values are not specified and reserved for future use.”

Page 23

Note 1

The following lines

“3.3.2.8 Bytecode Block Structure
This part is a block of bytecode corresponding to the method body:”

Should be replaced by

“3.3.2.8 Bytecode Area List
This section is a list of consecutive bytecode block structures. To each bytecode block structure corresponds one and only one non-native, non-abstract method of the internal method table of this class area. The bytecode block structures are stored following the order of storage of the corresponding methods in the internal method table.

Each bytecode block is represented by the following structure:”

Page 24

Note 1

The following lines

“3.3.2.9 Caught Exception Table
This structure gives the exception handling information for a method…”

Should be replaced by

“3.3.2.9 Exception Table List
This section is a list of consecutive exception table structures. To each exception table structure corresponds one and only one method of the internal method table of this class area. Some methods have no corresponding exception table structure. The exception tables are stored following the order of storage of the corresponding methods in the internal method table.

An exception table gives the exception handling information for a method…”

Note 2 OK

Note 3
Both spelling are used, Utf8 is more common.

Page 26

Note 1 OK

Page 29

Note 1 OK

Page 31

Notes 1, 2 OK

Note 3

The following lines

“3.3.6 File Signature
The VMFileSignature structure is not defined.”

Should be replaced by the following equivalent definition:

“3.3.6 Digital Signature

The JEFF specification does not impose any algorithm or any scheme for the signature a JEFF file. The digital signature of the JEFF file is stored in a VMFileSignature structure defined as follows:

```
VMFileSignature {
  TU1  nSignature[];
}
```

Where the byte array nSignature contains the signature data. The length of the array can be deduced from the position of the VMFileSignature structure and the total size of the JEFF.”

Page 32

Note 1

Operand is a better definition. “parameter” should be replaced by “operand” in the bytecode definitions.

Notes 2, 3, 4   OK

Note 5 OK
The word “normatively” is redundant here and should be removed.

Page 35

Note 1 OK
“JEFF” should be written in capital letters everywhere in the document

Page 47

Note 1, 2, 3, 4   OK

Additional Precisions

1 - Chapter 3.3.3.1 p. 26
The following comment should be added :
”The VM_ATTR_BYTE_ORDER flag must be set if the VM_ATTR_INDEXES, VM_ATTR_VMOFFSETS, or VM_ATTR_VMDOFFSETS flags are specified.”

2 - Chapter 3.3.3.1, p.26
In the comment of the value VM_ATTR_CST_LENGTH, the following comments should be added:
“The length of the attribute is constant and given by the nTypeLength item. This flag can only be used if the length of the attribute structure is not subject to variations caused by the type alignment and if the length can be encoded with a TU2 variable.”

3 - Chapter 3.3.3.3 p.28
In the comments of the nData item, the following comment should be added:

”The fields of the real structure corresponding to this attribute must follow all the alignment and padding constraints given in section 3.2.3”

4 - Chapter 4.2.15, p. 39
In the definition of the newconstarray bytecode, in the comment of the nLenght item, the following precision should be given:
“The nLength value is the length, in elements, of the new array. This value cannot be zero.”

5 - Chapter 4.3.2 p. 43
The bytecode jeff_ret is listed as a 1-byte length byte code while it's a 2-byte-length bytecode.

6 - Chapter 5, p.46
The presence of the total number of packages, classes, methods and fields in the file header induces the following limitations:

- the VMPINDEX values are included in the range [0, 65534]
- the VMCINDEX values are included in the range [0, 65534]
- the VMFINDEX values are included in the range [0, 2^32 - 2]
- the VMMINDEX values are included in the range [0, 2^32 - 2]