Java Card 2.0 Language Subset and Virtual Machine Specification

October 13, 1997

Revision 1.0 Final

---

1. Introduction

The information in this document describes the subset of standard Java which is supported in the Java Card 2.0 specification. This document is not intended to stand on its own; rather it relies heavily on existing documentation of standard Java. In particular, two books are required for the reader to understand the material presented herein.

[1] Gosling, James, Bill Joy, and Guy Steele. *The Java™ Language Specification*. Addison-Wesley, 1996, ISBN 0-201-63451-1 – provides a baseline definition of the Java language. The language subset defined here is based on the language specified in this book.

[2] Lindholm, Tim, and Frank Yellin. *The Java™ Virtual Machine Specification*. Addison-Wesley, 1996, ISBN 0-201-63452-X – defines the standard operation of the Java Virtual Machine. The virtual machine material presented in this subset is based on the definition specified in this book.

2. A Subset of Java

Java Card is a new system for programming smartcards. It is based on the Java language and virtual machine. Java Card programs are written with standard Java development tools, but may be installed and executed on smartcards. It would be ideal if Java Card programs could be written using all of the Java language, but a full Java Virtual Machine implementation is far too big to fit on even the most advanced smartcards available today.

A typical smartcard has under 1K of RAM and 16K of ROM. The code for implementing string manipulation, single and double-precision floating point arithmetic, and thread management would be larger than the ROM space on a card. Even if it could be made to fit, there would be no space left over for class libraries or application code. Then there is the question of RAM use. The only workable option is to implement Java Card as a subset of Java. In other words, we must leave some features out.

Fortunately, smartcard programs are by their nature extremely simple things. This allows us to omit features from Java with little or no impact on the kinds of programs we would write using Java Card.

3. Language

Java Card programs are written in Java. They are compiled using Java compilers. Java Card is a subset of Java, and familiarity with Java is required to understand Java Card programming. The items discussed in this section are not described to the level of a language specification. For complete documentation on the Java language, refer to *The Java Language Specification* [1].

3.1 Unsupported Items

The items listed in this section are elements of the Java language which are not supported in Java Card systems.

3.1.1 Features

3.1.1.1 Dynamic Class Loading

A Java Card system is not able to load classes dynamically. Classes are either masked into the card during manufacturing or installed through a secure installation process after the card has been issued. Programs executing on the card may only refer to classes which already exist on the card, as there is no way to download classes during the normal execution of application code. See *Java Card 2.0 – Programming Concepts* for more information.

3.1.1.2 Security Manager

The security model of Java Card systems differs from standard Java in fairly significant ways. Language security policies are implemented by the virtual machine. There is no Security Manager class which makes policy decisions on whether to allow operations.

3.1.1.3 Threads

The Java Card Virtual Machine does not support multiple threads of control. Neither class `Thread` or any of the thread-related keywords can be used in Java Card programs.

3.1.1.4 Cloning

Java Card does not support cloning of objects. Java Card's version of class `Object` does not implement a `clone()` method, and there is no `Clonable` interface provided.

3.1.1.5 Garbage Collection & Finalization

Java Card does not require a garbage collector. Nor does Java Card allow explicit deallocation of objects, as this would break Java's required pointer-safety. Therefore, application programmers may not assume that objects which are allocated are ever deallocated. Storage for unreachable objects will not necessarily be reclaimed.

Finalization is also not required. `finalize()` will not necessarily be called automatically by the virtual machine, and programmers should not rely on this behavior.

3.1.2 Keywords

The following keywords indicate types which are not supported for Java Card, or unsupported options related to Threads or memory management.

`char`	`float`	`synchronized`	`volatile`
`double`	`long`	`transient`	

3.1.3 Types

Java Card does not support types `char`, `double`, `float` or `long`, or operations on those types. It also does not support arrays with more than one dimension.

3.1.4 Classes

In general, none of the standard Java classes are supported in Java Card. Some classes from the `java.lang` package are supported (§3.2.4), but none of the rest are. Some noteworthy classes which are not supported are `String`, `Thread` (and all thread-related classes), wrapper classes such as `Boolean` and `Integer`, and class `Class`.

3.2 Supported Items

It is much more difficult to succinctly describe what is left in Java Card than to describe what is missing. If a language feature is not explicitly described as unsupported, it is part of the supported subset. Notable supported features are described in this section.

3.2.1 Features

3.2.1.1 Packages

Java Card programs follow the standard rules for Java packages. Java Card classes are written as `java` source files, which include package designations. Package mechanisms are used to identify and control access to classes, static fields and static methods. In all respects, packages in Java Card are used exactly the way they are in standard Java.

3.2.1.2 Dynamic Object Creation

Java Card programs can dynamically create objects, both class instances and arrays. This is done, as usual, by using the `new` operator. Objects are allocated out of the heap.

As noted in (§3.1.1.5), a Java Card Virtual Machine will not necessarily garbage collect objects. Any objects allocated on the card may continue to exist and consume resources even after they become unreachable. See *Java Card 2.0 – Programming Concepts* for more information.

3.2.1.3 Virtual Methods

Java Card objects are standard Java objects. Invoking virtual methods on objects in Java Card is exactly the same as in Java. Inheritance is supported, including the use of the `super` keyword.

3.2.1.4 Interfaces

Java Card classes may define or implement Interfaces as in standard Java. Invoking virtual methods on interface types works as expected. Type checking and the `instanceof` operator also work correctly with interfaces.

3.2.1.5 Exceptions

Java Card programs may define, throw and catch exceptions, as in standard Java. Class `Throwable` and its relevant subclasses are supported. (Some `Exception` and `Error` subclasses are omitted as those exceptions cannot occur in Java Card. See §4.3 for specification of errors and exceptions.)

3.2.2 Keywords

The following keywords are supported in Java Card. Their use is the same as in standard Java.

`abstract`	`default`	`if`	`package`	`switch`
`boolean`	`do`	`implements`	`private`	`this`
`break`	`else`	`import`	`protected`	`throw`
`byte`	`extends`	`instanceof`	`public`	`throws`
`case`	`final`	`int`	`return`	`try`
`catch`	`finally`	`interface`	`short`	`void`
`class`	`for`	`native`	`static`	`while`
`continue`	`goto`	`new`	`super`	

3.2.3 Types

Java Card supports the use of the standard Java types `boolean`, `byte`, `short`, and `int`. Objects (class instances and single-dimensional arrays) are also supported. Arrays can contain the supported primitive data types, objects, and other arrays.

Some Java Card implementations do not support use of the `int` data type.

3.2.4 Classes

Most of the classes in the `java.lang` package are not supported in Java Card. The following classes from `java.lang` are supported on the card in a limited form.

3.2.4.1 Object

Java Card classes descend from `java.lang.Object`, as in standard Java. Most of the methods of `Object` are not available in the Java Card API, but the class itself exists to provide a root for the class hierarchy.

3.2.4.2 Throwable

Since Java Card supports the use of exceptions, it supports class `Throwable` and its subclasses, where applicable. Most of the methods of `Throwable` are not available in the Java Card API, but the class itself exists to provide a common ancestor for all exceptions.

3.2.4.3 System

Class `java.lang.System` is not supported. Java Card supplies a class `javacard.framework.System` which provides an interface to system behavior.

3.3 Conditional Support

Several features of the Java language are only supported in certain conditions. These features are described below.

3.3.1 int

The `int` keyword and 32-bit integer data types will not necessarily be supported on all Java Card implementations. A Java Card Virtual Machine which does not support the `int` data type will reject programs which use that type.

3.3.2 native

Native methods must be available when creating the classes for the card's mask. Support of native methods in code installed post-issuance is optional.

3.4 Limitations

The limitations of card hardware prevent Java Card programs from supporting the full range of functionality of certain Java features. The features in question are supported, but a particular virtual machine may limit the range of operation to less than that of standard Java.

To ensure a level of portability for application code, this section establishes a minimum required level for partial support of these language features.

The limitations here are listed as maximums from the application programmer's perspective. Applets which do not violate these maximum values will be portable across all Java Card implementations. From the Java Card VM implementer's perspective, each maximum listed indicates a minimum level of support which will allow portability of applets.

In several cases, variations in data type encoding within the virtual machine make portability of Java Card source code difficult to predict. These cases are so noted.

3.4.1 Objects

3.4.1.1 Methods

Classes can implement a maximum of 127 instance methods (including inherited methods).

3.4.1.2 Class Instances

Java Card class instances can contain a maximum of 255 bytes of data in their fields. Internal data encoding, and therefore the maximum number of fields in objects, may vary from one virtual machine to another.

3.4.1.3 Arrays

Java Card arrays can hold a maximum of 32767 fields.

3.4.2 Methods

The maximum size of Java Card stack frame is 127 bytes. This includes the parameters, locals, and operand stack. Internal data encoding, and therefore the number of items which may be allocated on the stack, may vary from one virtual machine to another.

3.4.3 Switch Statements

Java Card systems which do not support the `int` data type are limited to a maximum of 65536 cases in switch statement. Systems with `int` support have the same maximum as standard Java.

3.4.4 Class Initialization

There is limited support for initialization of static field values in `<clinit>` methods. Static fields may only be initialized to primitive constant values, or arrays of primitive constants. Primitive constant data types include `boolean`, `byte`, `short`, and `int`.

4. VM

4.1 `class` File Subset

The Java Card Virtual Machine operates on standard Java `class` files. As the Java Card Virtual Machine supports only a subset of the behavior of the standard Java Virtual Machine, it also supports only a subset of the standard `class` file format.

4.1.1 Not Supported

4.1.1.1 Field Descriptors

Field descriptors may not contain *BaseType* characters **C**, **D**, **F** or **L**. *ArrayType* descriptors for arrays of more than one dimension may not be used.

4.1.1.2 Constant Pool

Constant pool table entry tags which indicate unsupported types are not supported.

Constant Type	Value
CONSTANT_String	8
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6

Table 4.1 Unsupported constant pool tags

Constant pool structures for types `CONSTANT_String_info`, `CONSTANT_Float_info`, `CONSTANT_Long_info` and `CONSTANT_Double_info` are not supported.

4.1.1.3 Fields

In `field_info` structures, the access flags `ACC_VOLATILE` and `ACC_TRANSIENT` are not supported.

4.1.1.4 Methods

In `method_info` structures, the access flag `ACC_SYNCHRONIZED` is not supported. The access flag `ACC_NATIVE` is not necessarily supported in applet class files.

4.1.2 Supported

4.1.2.1 ClassFile

All items in the `ClassFile` structure are supported.

4.1.2.2 Field Descriptors

Field descriptors may contain *BaseType* characters **`B`**, **`I`**, **`S`** and **`Z`**, as well as any *ObjectType*. *ArrayType* descriptors for arrays of a single dimension may also be used.

4.1.2.3 Method Descriptors

All forms of method descriptors are supported.

4.1.2.4 Constant Pool

Constant pool table entry tags for supported data types are supported.

Constant Type	**Value**
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_Integer	3
CONSTANT_NameAndType	12
CONSTANT_Utf8	1

Table 4.2 Supported constant pool tags

Constant pool structures for types `CONSTANT_Class_info`, `CONSTANT_Fieldref_info`, `CONSTANT_Methodref_info`, `CONSTANT_InterfaceMethodref_info`, `CONSTANT_Integer_info`, `CONSTANT_NameAndType_info` and `CONSTANT_Utf8_info` are supported.

4.1.2.5 Fields

In `field_info` structures, the supported access flags are `ACC_PUBLIC`, `ACC_PRIVATE`, `ACC_PROTECTED`, `ACC_STATIC` and `ACC_FINAL`.

The remaining components of `field_info` structures are fully supported.

4.1.2.6 Methods

In method_info structures, the supported access flags are ACC_PUBLIC, ACC_PRIVATE, ACC_PROTECTED, ACC_STATIC, ACC_FINAL and ACC_ABSTRACT. The access flag ACC_NATIVE is supported for non-applet class files.

The remaining components of method_info structures are fully supported.

4.1.2.7 Attributes

The attribute_info structure is supported. The Code, ConstantValue, Exceptions and LocalVariableTable attributes are supported.

4.2 Bytecode Subset

4.2.1 Unsupported Bytecodes

lconst_<l>	fconst_<f>	dconst_<d>	ldc2_w2
lload	fload	dload	lload_<n>
fload_<n>	dload_<n>	laload	faload
daload	caload	lstore	fstore
dstore	lstore_<n>	fstore_<n>	dstore_<n>
lastore	fastore	dastore	castore
ladd	fadd	dadd	lsub
fsub	dsub	lmul	fmul
dmul	ldiv	fdiv	ddiv
lrem	frem	drem	lneg
fneg	dneg	lshl	lshr
lushr	land	lor	lxor
i2l	i2f	i2d	l2i
l2f	l2d	f2i	f2d
d2i	d2l	d2f	i2c
lcmp	fcmpl	fcmpg	dcmpl
dcmpg	lreturn	freturn	dreturn
monitorenter	monitorexit	multianewarray	goto_w
jsr_w			

4.2.2 Supported Bytecodes

nop	aconst_null	iconst_<i>	bipush
sipush	ldc	ldc_w	iload
aload	iload_<n>	aload_<n>	iaload
aaload	baload	saload	istore

astore	istore_<n>	astore_<n>	iastore
aastore	bastore	sastore	pop
pop2	dup	dup_x1	dup_x2
dup2	dup2_x1	dup2_x2	swap
iadd	isub	imul	idiv
irem	ineg	ior	ishl
ishr	iushr	iand	ixor
iinc	i2b	i2s	if<cond>
ificmp_<cond>	ifacmp_<cond>	goto	jsr
ret	tableswitch	lookupswitch	ireturn
areturn	return	getstatic	putstatic
getfield	putfield	invokevirtual	invokespecial
invokestatic	invokeinterface	new	newarray
anewarray	arraylength	athrow	checkcast
instanceof	wide	ifnull	ifnonnull

4.2.3 Static Restrictions on Bytecodes

A `class` file must conform to the following restrictions on the static form of bytecodes for it to be acceptable to a Java Card Virtual Machine.

4.2.3.1 ldc, ldc_w

The `ldc` and `ldc_w` bytecodes can only be used to load integer constants. The constant pool entry at *index* must be a `CONSTANT_Integer` entry.

4.2.3.2 lookupswitch

The value of the *npairs* operand must be less than 65536. The bytecode can contain at most 65535 cases.

4.2.3.3 tableswitch

The values of the *high* and *low* operands must both be less than 65536 (so they can fit in 16 bits). The bytecode can contain at most 65535 cases.

4.2.3.4 wide

The `wide` bytecode cannot be used to generate local indices greater than 127, and it cannot be used with any instructions other than `iinc`. It can only be used with an `iinc` bytecode to extend the range of the increment constant.

4.3 Exceptions

Java Card provides full support for the Java exception mechanism. Users can define, throw and catch exceptions just as in standard Java. Java Card also makes use of the standard exceptions and errors defined in *The Java Language Specification* [1]. An updated list of Java's standard exceptions is provided in the JDK documentation.

Not all of Java's standard exceptions are supported in Java Card. Exceptions related to unsupported features are naturally not supported. Class loader exceptions (the bulk of the checked exceptions) are not supported. And no exceptions or errors defined in packages other than `java.lang` are supported.

Note that some exceptions may be supported to the extent that their error conditions are detected correctly, but classes for those exceptions will not necessarily be present in the API.

The supported subset is described in Tables 4.3, 4.4 and 4.5.

4.3.1 Uncaught and Uncatchable Exceptions

In standard Java, uncaught exceptions and errors will cause the virtual machine to report the error condition and exit. In Java Card, uncaught exceptions or errors should cause the card to be muted. A virtual machine has the option of taking more drastic actions, such as blocking the card from further use.

Throwing a runtime exception or error which cannot be caught should also cause the card to be muted. Cards may also optionally take stricter actions in response to throwing such an exception.

4.3.2 Checked Exceptions

Exception	**Supported**	**Not Supported**
ClassNotFoundException		●
CloneNotSupportedException		●
IllegalAccessException		●
InstantiationException		●
InterruptedException		●
NoSuchFieldException		●
NoSuchMethodException		●

Table 4.3 support of checked exceptions

4.3.3 Runtime Exceptions

Runtime Exception	**Supported**	**Not Supported**
ArithmeticException	●	
ArrayStoreException	●	
ClassCastException	●	
IllegalArgumentException	●	
IllegalThreadStateException		●
NumberFormatException		●
IllegalMonitorStateException		●
IllegalStateException	●	
IndexOutOfBoundsException	●	
ArrayIndexOutOfBoundsException	●	
StringIndexOutOfBoundsException		●
NegativeArraySizeException	●	
NullPointerException	●	
SecurityException	●	

Table 4.4 Support of runtime exceptions

4.3.4 Errors

Error	Supported	Not Supported
LinkageError	•	
ClassCircularityError	•	
ClassFormatError	•	
ExceptionInInitializerError	•	
IncompatibleClassChangeError	•	
AbstractMethodError	•	
IllegalAccessError	•	
InstantiationError	•	
NoSuchFieldError	•	
NoSuchMethodError	•	
NoClassDefFoundError	•	
UnsatisfiedLinkError	•	
VerifyError	•	
ThreadDeath		•
VirtualMachineError	•	
InternalError	•	
OutOfMemoryError	•	
StackOverflowError	•	
UnknownError	•	

Table 4.5 Support of errors