Appendix B

JCOD Syntax

This chapter describes JCOD syntax, and how to encode class files using JCOD. Jcoder is a low-level assembler that accepts text in the JCOD format and produces a `.class` file for use with a Java Virtual Machine. Jcoder's primary use is as a tool for producing specialized tests for testing a JVM implementation.

Jcoder is called a low-level assembler because it provides very few higher abstractions beyond the class file format. Typically, there is a 1-1 relationship between constructs in the class file, and constructs in the JCOD format. All Java Spec come from Chapter 4 of the Java Virtual Machine Specification.

This chapter contains the following sections:

Syntax

The source text file can be free form (newlines are considered blanks) and may contain Java-style commenting. The first line of a JCOD file represents the name of the resulting file in the destination directory. This name does not affect the content of the resulting file. This line has two forms:

```
file FILENAME

or

class CLASSNAME
```

In the latter case, extension `.class` will be added to form FILENAME. Jcod's `-d option` allows you to define the destination directory. A list of structured data items follows the class name. The length (in bytes) of each item is determined by its representation.

Primitive Types

Types:

- Decimal Values
- Hexadecimal Values
- Unicode Values
- Constants Values

Decimal Values

Decimal values may be followed by an optional size indicator. Default values (no indicator) are 2 bytes. Decimal numbers may start with `#' sign. The `#' sign does not have an affect on the number, but may simply be used to indicate decimal values.

Optional Size Indicators

```
b (1 byte)
s (2 bytes)
i (4 bytes)
l (8 bytes)
```

Examples

```
17
17s
#17
```

Hexadecimal Values

Size of hexadecimal values is determined by the number of places the number holds. Two places is equal to 1 byte. A hexadecimal number may not hold more than 16 places (8 bytes). Leading zeros should be used to where appropriate.

Examples

```
0x17 (1 byte)
0x0017 (2 bytes)
```

Unicode Values

Unicode strings use the same syntax as strings in the Java programming language.

Pure unicode strings are bounded with character \' instead of \". Unlike ordinary strings, they are not prepended with two-byte integer representing actual string's length. As a result, they may exceed 64Kbyte limit.

Constant Values

Each constant is 1 byte in size.

Constant Pool Tags:

```
Utf8 = 1;
int = 3;
float = 4;
long = 5;
double = 6;
class = 7;
String = 8;
Field = 9;
Method = 10;
InterfaceMethod = 11;
NameAndType = 12;
```

Types for StackMap attribute:

```
B = 0;
I = 1;
F = 2;
D = 3;
L = 4;
N = 5;
IO = 6;
O = 7;
NO = 8;
```

`Types for access flags typically are encoded in 2 bytes:`

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declared `public`; may be accessed from outside its package.
ACC_PRIVATE	0x0002	Declared `private`; usable only within the defining class.
ACC_PROTECTED	0x0004	Declared `protected`; may be accessed within subclasses.

Note - access flags are defined separately for Class, Field, Methods etc. You should look

`at the VM spec to see the flags that are applicable for a given entity.`

Structured Data Types

Types:

- Record
- Open List
- Closed List
- Array
- Byte Array
- Attribute
- Component

Record

A record is a sequence of simple data types separated by blanks or commas. Records may be limited by a semicolon or right curly brace `}'.

Example

```
class #5
1,2,3,4
```

Open List

An open list is sequence of records separated by semicolons.

Example

```
class #5; // #1
class #12; // #2
Method #2 #0; // #3
```

Closed List

Open list enclosed in curly braces `{ }'.

Example

```
{
 0x0001;// access
 #14; // name_cpx
 #15; // sig_cpx
}
```

Array

An integer in square brackets followed by a closed list. The integer represents the length of the array and occupies 2 bytes in the resulting class file. If omitted, the integer will represent the number of records in the following closed list. If the right square bracket is followed by letter "b", then the length occupies 1 byte. Such arrays are used in JavaCard .cap files.

Example

```
[]
{
 // Constant Pool
 ; // first element of constant pool is always empty
 class #5; // #1
 class #12; // #2
 Method #2 #0; // #3
 NameAndType #14 #0; // #4
 Utf8 "empty"; // #5
}
```

In this example, Jcoder will set the length of the array to 6 because the first semicolon is counted as an element delimiter, even though the first element occupies 0 bytes.

Byte Array

A byte array is defined through the use of the "Byte" keyword followed by length of the array and the array itself. By default, the length of the array occupies 4 bytes in the resulting `.class` file. If the right square bracket is followed by letter "b", then the length occupies 1 byte, letter "s" sets the size of length to 2 bytes. If the length is omitted, the number of bytes in the following closed list is used.

Example

```
Bytes[5]
{
 0x2AB70003B1;
};
```

Attribute

An attribute is defined through the use of the "Attr" keyword. This keyword is followed by a 2 byte integer index in the constant pool and a 4 byte integer representing the length of the variable part of the attribute in bytes. These two values should be inclosed in parenthesis and separated by commas. If the second value is omitted (length), the actual number of bytes in the following closed list is used.

Example

```
Attr(#9, 2)
{ // SourceFile at 0x000000DE
 #13;
} // end SourceFile
```

Component

A Component is very similar to Attribute. The only difference is that first value in parenthesis (used to be index in attribute and tag in component) occupies 1 byte and the second value (length) occupies 2 bytes. Component are useful to represent JavaCard .cap files.

Example

```
Component(3, 17) { // Applet
         [1]b { // applets
                          { // applet 0
                                  Bytes[13]b {
                                          0xA00000006203010A;
                                          0x0103010681;
                                  };
                                  142; // install method offset
                          };
                 };
};
```

Structured Data Type Rules

It is possible to define a length of an array, byte array or attribute that differs from the actual length of array, byte array or attribute. In this case, a warning is generated but the your defined length will be used.

General Class Structure

Class File Spec	JCOD

```
ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-
1];
    u2 access_flags;
    u2 this_class;
    u2 super_class;
    u2 interfaces_count;
    u2 interfaces[interfaces_count];
    u2 fields_count;
    field_info fields[fields_count];
    u2 methods_count;
    method_info methods[methods_count];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

[4.1]

```
class Foo
{
  0xCAFEBABE; // magic
  0; // minor version
  50; // major version
  [v] { // Constant Pool
    ; // first element is empty
    ...
  } // Constant Pool

  0x0021; // access flags
  #5;// this_cpx
  #6;// super_cpx

  [w] { // Interfaces
    ...
  } // Interfaces

  [x] { // fields
    ...
  } // fields

  [y] { // methods
      ...
  } // methods

  [z] { // Attributes
      ...
  } // Attributes
} // end class Foo
```

Class Attributes

The following examples describe the attributes that may appear in a classes attributes list.

Class File Spec	JCOD

`SourceFile_attribute {`
 	
` //index to (UTF8): "SourceFile"`
`  u2 attribute_name_index;`
`  u4 attribute_length; //always 2`
 	
` //index to (UTF8): file name`
`  u2 sourcefile_index;`
`}`
 	
[4.7.10]
	`class Foo {`
	`  ...`
	`  [1] { // Attributes`
	`  Attr(#9, 2){ // SourceFile at 0x000000DE`
	`  #13;  // index to UTF8: filename`
	`  } // end SourceFile attr`
	`  } // end Attributes`
	`  ...`
	`} // class Foo`

`Deprecated_attribute {`
 	
` //index to (UTF8): "Deprecated"`
`  u2 attribute_name_index;`
`  u4 attribute_length; //always 0`
`}`
 	
[4.7.15]
	`class Foo {`
	`  ...`
	`  [1] { // Attributes`
	`  Attr(#8, 0){ // Deprecated`
	`  } // end Deprecated attr`
	`  } // end Attributes`
	`  ...`
	`} // class Foo`

```
Synthetic_attribute {

 //index to (UTF8): "Synthetic"
 u2 attribute_name_index;
 u4 attribute_length; //always 0
}
```

[4.7.8]

```
class Foo {
   ...
     [1] { // Attributes
               Attr(#8, 0){ // Synthetic
               } // end Synthetic attr
           } // end Attributes
   ...
} // class Foo
```

```
Signature_attribute {

 //index to (UTF8): "Signature"
 u2 attribute_name_index;
 u4 attribute_length;

 //index to (UTF8): signature
 u2 signature_index;
}
```

[4.7.9]

```
class Foo {
   ...
     [1] { // Attributes
               Attr(#8, 2){ // Signature
                 #30; //cpx_signature
               } // end Signature attr
           } // end Attributes
   ...
} // class Foo
```

```
BootstrapMethods_attribute {
 //index to (UTF8): "BootstrapMethods"

 u2 attribute_name_index;

 u4 attribute_length;

 u2 bootstrap_method_count;

 {

   //index to CONSTANT_MethodHandle

   u2 bootstrap_method_ref;

   //number of arguments in Method

   u2 bootstrap_argument_count;

   {

     // constant pool index

     u2 bootstrap_argument;

   } arguments[number_of_arguments];

 } bootstrap_methods
        [number_of_bootstrap_methods];

}
```

[4.7.23]

```
class Foo {
   ...
     [1] { // Attributes
               Attr(#8, 6){ // BootstrapMethods
                  [1] { // BootstrapMethodEntries
                        #12; //cpx_method_handle
                         [1] { // Arguments
                              #14; //cpx_arg1
                            } // end Arguments
                     } // end BootstrapMethods entries
               } // end BootstrapMethods attr
           } // end Attributes
   ...
} // class Foo
```

```
InnerClasses_attribute {

 //index to (UTF8): "InnerClasses"
 u2 attribute_name_index;
 u4 attribute_length;
 u2 number_of_classes;
 {

 //index to (Class:) Class Reference
 u2 inner_class_info_index;

 //index to (Class:) Class Reference
 u2 outer_class_info_index;

 //index to (UTF8): name
 u2 inner_name_index;

 // Access Flags (Long)
 u2 inner_class_access_flags;
 } classes[number_of_classes];
}
```

[4.7.6]

```
class Foo {
   ...
     [1] { // Attributes
              Attr(#8, 8){ // InnerClasses
                 [1] { // InnerClassEntries
                       #12; //cpx_inner_class_info
                       #13; //cpx_outer_class_info
                       #14; //cpx_inner_class_name
                       01L; //inner class access
                     } // end InnerClass entries
              } // end InnerClass attr
         } // end Attributes
   ...
} // class Foo
```

```
EnclosingMethod_attribute {
 u2 attribute_name_index;
 u4 attribute_length;

 //index to (Class:) Class Reference
 u2 class_index
 //index to (NameAndType):
 // Method name/type ref

 u2 method_index;
}
```

[4.7.7]

```
class Foo {
   ...
     [1] { // Attributes
              Attr(#8, 4){ // EnclosingMethod
                 #10; //cpx_class
                 #42; //cpx_method
              } // end EnclosingMethod attr
         } // end Attributes
   ...
} // class Foo
```

```
SourceDebugExtension_attribute {
 u2 attribute_name_index;
 u4 attribute_length;
 u1 debug_extension[attribute_length];
}
```

[4.7.11]

```
class Foo {
   ...
     [1] { // Attributes
              Attr(#8, 2){ // SourceDebugExtension
                 Bytes[5] {
                    0x2AB70003B1;
                 };
              } // end SourceDebugExtension attr
         } // end Attributes
   ...
} // class Foo
```

```
RuntimeVisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.16]

```
class Foo {
   ...
     [1] { // Attributes
              Attr(#8, X){ // RuntimeVisibleAnnotations
                 [n] { // annotations
                    annotation_1
                    annotation_2
                     ...
                    annotation_n
                 } // end annotations
              } // end RuntimeVisibleAnnotations attr
           } // end Attributes
   ...
} // class Foo
```

```
RuntimeInvisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.17]

```
class Foo {
   ...
     [1] { // Attributes
              Attr(#8, X){ // RuntimeInvisibleAnnotations
                 [n] { // annotations
                    annotation_1
                    annotation_2
                     ...
                    annotation_n
                 } // end annotations
              } // end RuntimeInvisibleAnnotations attr
           } // end Attributes
   ...
} // class Foo
```

The Constant Pool and Constant Elements

A variety of information is stored in the constant pool of a class file.

Class File Spec	JCOD
[4.4]	

```
ClassFile {
 ...
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-1];
 ...


}
```

```
class Foo {
 ...
  [] { // Constant Pool
    ; // first element is empty
    cp_info_1; // #1
    cp_info_2; // #2
    cp_info_3; // #3
     ...
    cp_info_n; // #n
  } // Constant Pool

  ...
} // end class Foo
```

Constant Declarations

The following examples describe the various constant declarations that may appear in a constant pool.

Class File Spec	JCOD

```
...
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-1]
    {

      cp_info {
         u1 tag;
         u1 info[];
      }

 ...
    }
 ...
```

[4.4]

```
 ...
  [] { // Constant Pool
    ; // first element is empty
    Method #6 #15; // #1
    Field #16 #17; // #2
    String #18; // #3
    Method #19 #20; // #4
    class #21; // #5
    class #22; // #6
    Utf8 ""; // #7
    Utf8 "()V"; // #8
    Utf8 "Code"; // #9
    Utf8 "LineNumberTable"; // #10
    Utf8 "main"; // #11
    Utf8 "([Ljava/lang/String;)V"; // #12
    Utf8 "SourceFile"; // #13
    Utf8 "HelloWorld.java"; // #14
    NameAndType #7 #8; // #15
    class #23; // #16
    NameAndType #24 #25; // #17
    Utf8 "Hello, world!"; // #18
    class #26; // #19
    NameAndType #27 #28; // #20
    Utf8 "HelloWorld"; // #21
    Utf8 "java/lang/Object"; // #22
    Utf8 "java/lang/System"; // #23
    Utf8 "out"; // #24
    Utf8 "Ljava/io/PrintStream;"; // #25
    Utf8 "java/io/PrintStream"; // #26
    Utf8 "println"; // #27
    Utf8 "(Ljava/lang/String;)V"; // #28
  } // Constant Pool

   ...
```

```
CONSTANT_Class (Reference)
 [7] // tag value for ClassRef
 [cp_index] //index to (UTF8):
 //  Class name
```

[4.4.1]

```
class #26; // Class ref, name = const #26
 ...
 Utf8 "java/io/PrintStream"; // #26
```

```
CONSTANT_Fieldref (Reference)
 [9] // tag value for FieldRef
 [cp_index] //index to (Class): Class reference
 [cp_index] //index to (NameAndType):
 // Field name/type ref
```

[4.4.2]

```
Field #16 #17; // #2
 ...
 class #23; // #16
 NameAndType #24 #25; // #17
```

```
CONSTANT_Methodref  (Reference)
 [10] // tag value for ClassRef
 [cp_index] //index to (Class): Class reference
 [cp_index] //index to (NameAndType):
 // Method name/signature ref
```

[4.4.2]

```
Method #19 #20; // #4
 ...
 class #26; // #19
 NameAndType #27 #28; // #20
```

```
CONSTANT_InterfaceMethodref  (Reference)
 [11] // tag value for ClassRef
 [cp_index] //index to (Class): Class reference
 [cp_index] //index to (NameAndType):
 // Method name/signature ref
```

[4.4.2]

```
Interface #19 #20; // #4

...

class #26; // #19

NameAndType #27 #28; // #20
```

CONSTANT_String (Reference)
[8] // tag value for StringRef
[cp_index] //index to (UTF8)
 	
[4.4.3]
	String #18; // #3
	...
	Utf8 "Hello, world!"; // #18

CONSTANT_Integer (Value) [3] // tag value for Integer [u4] //bytes (Integer) [4.4.4]	Integer 31; // = 31
CONSTANT_Float (Value) [4] // tag value for Integer [u4] //bytes (Integer) [4.4.4]	Float 3.14; // = 3.14f
CONSTANT_Long (Value) [5] // tag value for Long [u4] //high-bytes (Long) [u4] //low-bytes (Long) [4.4.5]	Long 0x7FFL; // = 0x7ff0000000000000L
CONSTANT_Double (Value) [6] // tag value for Long [u4] //high-bytes (Long) [u4] //low-bytes (Long) [4.4.5]	Double 3.14; // = 3.14D

CONSTANT_NameAndType (Reference)
[12] // tag value for NameAndTypeRef
[cp_index] //index to (UTF8): Name
[cp_index] //index to (UTF8):
// Type or Signature
 	
[4.4.6]
	NameAndType #27 #28; // #20
	...
	Utf8 "println"; // #27
	Utf8 "(Ljava/lang/String;)V"; // #28

CONSTANT_UTF8 (Value) [1] // tag value for UTF8 [u2] //string length (in bytes) bytes[u2] // byte array [4.4.7]	Utf8 "main"; // #11

CONSTANT_MethodHandle (Value)
[u1] // subtag value
[u2] // index to:
// (MethodRef|InterfaceMethod|Field)
 	
[4.4.8]
	...
	[] { // Constant Pool
	; // first element is empty
	...
	Utf8 "()V"; // #11
	Method #17 #19; // #12
	...
	// #16 // reference with subtag=7b to
	// Method Test.methName:"()V"
	MethodHandle 7b #12;
	
	class #18; // #17
	Utf8 "Test"; // #18
	NameAndType #20 #11; // #19
	Utf8 "methName"; // #20
	...
	} // Constant Pool

CONSTANT_MethodType (Value) [cp_index] //index to (UTF8): // method type [4.4.9]	MethodType "()V"; // #11

CONSTANT_InvokeDynamic (Value)
[u1] // offset value in
// BootstrapMethods table
[cp_index] //index to (NameAndType)
 	
[4.4.10]
	...
	[] { // Constant Pool
	; // first element is empty
	...
	// #1 // reference to record at
	// index 0 in BootstrapMethods attribute,
	// and to NameAndType #10
	// (methName:"(I)I")
	
	InvokeDynamic 0s #10;
	
	Utf8 "methName"; // #6
	...
	NameAndType #6 #21; // #10
	...
	Utf8 "(I)I"; // #21
	} // Constant Pool

Fields and Field Elements

Class File Spec	JCOD

ClassFile {
...
u2 fields_count;
field_info fields[fields_count];
...
}
 	
[4.5]
	class Foo {
	...
	[n] { // fields
	field_info_1; // #1
	field_info_2; // #2
	field_info_3; // #3
	...
	field_info_n; // #n
	} // fields
	...
	} // end class Foo

```
field_info {
    u2 access_flags;

 //index to (UTF*): field name
    u2 name_index;

 //index to (UTF8): field type
    u2 descriptor_index;

 //integer attribute_info
    u2 attributes_count; attributes
[attributes_count];
}
```

```
[n] { // fields
     ...
      { // field_info Bar
        0x0001; // access_flags
        #7; // name_cpx
        #8; // desc_cpx
        1   // Attribute_count
        [x] { // Attributes
              ...
        } // end Attributes
    } // end Field Bar
  ...
 } // fields
```

[4.5]

Field Attributes

The following examples describe the various attributes that may exist in a field attributes list.

Class File Spec	JCOD

`ConstantValue_attribute {`
 	
` //index to (UTF8): "ConstantValue"`
`    u2 attribute_name_index;`
`    u4 attribute_length; //always 2`
 	
` //index to (String|Integer|Long|Float|Double)`
`    u2 constantvalue_index;`
`}`
 	
[4.7.2]
	`...`
	`  { // Field Bar`
	`      ...`
	`      [1] { // Attributes`
	`           Attr(#7, 2){ // ConstantValue`
	`             #8; // attr_value_cpx`
	`           } // end ConstantValue attr`
	`        } // end Attributes`
	`       ...`
	`   } // end Field Bar`
	`...`

`Synthetic_attribute {`
 	
` //index to (UTF8): "Synthetic"`
`    u2 attribute_name_index;`
`    u4 attribute_length; //always 0`
`}`
 	
[4.7.8]
	`...`
	`  { // Field Bar`
	`    ...`
	`      [1] { // Attributes`
	`            Attr(#8, 0){ // Synthetic`
	`            } // end Synthetic attr`
	`          } // end Attributes`
	`     ...`
	`   } // end Field Bar`
	`...`

`Deprecated_attribute {`
` //index to (UTF8): "deprecated"`
`    u2 attribute_name_index;`
`    u4 attribute_length; //always 0`
`}`
 	
[4.7.15]
	`...`
	`  { // Field Bar`
	`    ...`
	`      [1] { // Attributes`
	`            Attr(#8, 0){ // Deprecated`
	`            } // end Deprecated attr`
	`          } // end Attributes`
	`   } // end Field Bar`
	`...`

```
Signature_attribute {
 //index to (UTF8): "Signature"
 u2 attribute_name_index;
 u4 attribute_length;
 //index to (UTF8): signature
 u2 signature_index;
}
```

[4.7.9]

```
   ...
     { // Field Bar
       ...
         [1] { // Attributes
               Attr(#8, 2){ // Signature
                  #32; //cpx_signature
               } // end Signature attr
           } // end Attributes
       ...
     } // end Field Bar
   ...
```

```
RuntimeVisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.16]

```
...
     { // Field Bar
       ...
         [1] { // Attributes

 // RuntimeVisibleAnnotations
               Attr(#8, X){
                  [n] { // annotations
                     annotation_1
                     annotation_2
                      ...
                     annotation_n
                  } // end annotations
               }
 // end RuntimeVisibleAnnotations attr
           } // end Attributes
       ...
     } // end Field Bar
   ...
```

```
RuntimeInvisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.17]

```
...
     { // Field Bar
       ...
         [1] { // Attributes

 // RuntimeInvisibleAnnotations
               Attr(#8, X){
                  [n] { // annotations
                     annotation_1
                     annotation_2
                      ...
                     annotation_n
                  } // end annotations
               }
 // end RuntimeInvisibleAnnotations attr
           } // end Attributes
       ...
     } // end Field Bar
  ...
```

Methods and Method Elements

Class File Spec	JCOD

```
ClassFile {
    ...
    u2 methods_count;
    method_info methods[methods_count];
    ...
}
```

[4.6]

```
class Foo {
  ...
  [n] { // methods
    method_info_1; // #1
    method_info_2; // #2
    method_info_3; // #3
     ...
    method_info_n; // #n
  } // end methods
  ...
} // end class Foo
```

```
method_info {
    u2 access_flags;

 //index to (UTF8): Name
    u2 name_index;

 //index to (UTF8): Method Signature
    u2 descriptor_index;

 //integer attribute_info
    u2 attributes_count; attributes
[attributes_count];
}
```

[4.6]

```
class Foo {
  ...
  [n] { // methods
    ...
       {  // method info Bar
         0x0001; // access_flags
         #7; // name_cpx
         #8; // desc_cpx
         1   // Attribute_count
         [x] { // Attributes
         ...
         } // end Attributes
       } // end method Bar
     ...
  } // end methods
  ...
} // end class Foo
```

Method Attributes

The following examples describe the various attributes that may exist in a method attributes list.

Class File Spec	JCOD

`Exceptions_attribute {`
 	
` //index to (UTF8): "Exceptions"`
`  u2 attribute_name_index;`
`  u4 attribute_length; //integer`
`  u2 number_of_exceptions; //integer`
`  u2 exception_index_table`
`[number_of_exceptions];`
` //index to (ConstantClassInfo)`
`}`
 	
[4.7.5]
	`...`
	`  { // Method Bar`
	`  ...`
	`  [1] { // Attributes`
	`  ...`
	
	`// Exceptions Attr`
	`  Attr(#7, 6){`
	`  [2] {  // numb exceptions`
	`  #30; // exception 1`
	`  #31; // exception 2`
	`  }`
	`  }`
	`  ...`
	`  }`
	`  } // end Method Bar`
	`...`

`Synthetic_attribute {`
 	
` //index to (UTF8): "Synthetic"`
`  u2 attribute_name_index;`
`  u4 attribute_length; //always 0`
`}`
 	
[4.7.8]
	`...`
	`  { // Method Bar`
	`  ...`
	`  [1] { // Attributes`
	`  Attr(#8, 0){ // Synthetic`
	`  } // end Synthetic attr`
	`  } // end Attributes`
	`  ...`
	`  } // end Method Bar`
	`...`

```
Deprecated_attribute {

 //index to (UTF8): "deprecated"
    u2 attribute_name_index;
    u4 attribute_length; //always 0

}
```

[4.7.15]

```
   ...
     { // Method Bar
       ...
         [1] { // Attributes
               Attr(#8, 0){ // Deprecated
               } // end Deprecated attr
             } // end Attributes
        ...
     } // end Method Bar
   ...
```

```
Signature_attribute {

 //index to (UTF8): "Signature"
 u2 attribute_name_index;
 u4 attribute_length;

 //index to (UTF8): signature
 u2 signature_index;
}
```

[4.7.9]

```
  ...
     { // Method Bar
       ...
         [] { // Attributes
               Attr(#8){ // Signature
                  #32; //cpx_signature
               } // end Signature attr
             } // end Attributes
        ...
     } // end Method Bar
   ...
```

```
Code_attribute {
 //index to (UTF8): "Code"
    u2 attribute_name_index;
    u4 attribute_length; //integer
    u2 max_stack; //integer
    u2 max_locals; //integer
    u4 code_length; //long integer
    u1 code[code_length]; //byte array
    u2 exception_table_length; //integer
    { u2 start_pc; //integer
     u2 end_pc; //integer
     u2 handler_pc; //integer
     u2 catch_type; //index to (ConstantClass)
    } exception_table[exception_table_length];
    u2 attributes_count; //integer
    attribute_info attributes[attributes_count];
}
```

[4.7.3]

```
   ...
     { // Method Bar
       ...
     [] { // Attributes
          ...
         Attr(#11) { //Code
            1; // max_stack
            1; // max_locals
            Bytes[5] {
                0x2AB70003B1;
            };
            [] { // Traps
            } // end Traps
            [] { // Attributes
               ...
            } // end Attributes
          } // end Code Attr
          ...
       } // end Attributes
     } // end Method Bar
   ...
```

```
RuntimeVisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.16]

```
   ...
     { // Method Bar
       ...
         [1] { // Attributes

// RuntimeVisibleAnnotations
           Attr(#13) {
             [] { // annotations
                  {  //  annotation
annotation_1
annotation_2
...
annotation_n
                  }  //  annotation
                }
              }
// end RuntimeVisibleAnnotations
           ...
         } // end Attributes
        ...
     } // end Method Bar
   ...
```

```
RuntimeInvisibleAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 num_annotations;
    annotation annotations[num_annotations];
}
```

[4.7.17]

```
   ...
     { // Method Bar
       ...
         [1] { // Attributes

 // RuntimeInvisibleAnnotations
            Attr(#13) {
              [] { // annotations
                   {  //  annotation
                      annotation_1
 annotation_2
 ...
 annotation_n
                   }  //  annotation
                 }
               }
 // end RuntimeInvisibleAnnotations
            ...
           } // end Attributes
         ...
      } // end Method Bar
   ...
```

```
RuntimeVisibleParameterAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u1 num_parameters;
    {       u2 num_annotations;
        annotation annotations[num_annotations];
    }       parameter_annotations[num_parameters];
}
```

[4.7.18]

```
   ...
     { // Method Bar
       ...
         [1] { // Attributes
// RuntimeInvisibleParameterAnnotations
              Attr(#8, X){
                 [n] { // annotations
                    annotation_1
                    annotation_2
                     ...
                    annotation_n
                 } // end annotations
               }
// end RuntimeVisibleParameterAnnotations attr
            } // end Attributes
          ...
      } // end Method Bar
   ...
```

```
RuntimeInvisibleParameterAnnotations_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u1 num_parameters;
    {       u2 num_annotations;
        annotation annotations[num_annotations];
    }       parameter_annotations[num_parameters];
}
```

[4.7.19]

```
   ...
     { // Method Bar
       ...
         [1] { // Attributes

 // RuntimeInvisibleParameterAnnotations
              Attr(#8, X){
                 [n] { // annotations
                    annotation_1
                    annotation_2
                     ...
                    annotation_n
                 } // end annotations
               }
// end RuntimeInvisibleParameterAnnotations attr
            } // end Attributes
          ...
      } // end Method Bar
   ...
```

```
AnnotationDefault_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    element_value default_value;
}
```

[4.7.22]

```
...
  { // Method Bar
   ...
     [] { // Attributes
          Attr(#8){ // AnnotationDefault
             element_value
             } // end annotations
          } // end AnnotationDefault attr
       } // end Attributes
   ...
  } // end Method Bar
...
```

Code Attributes

The following examples describe the various attributes that may exist in a code attribute.

Class File Spec	JCOD
`LineNumberTable_attribute { u2 attribute_name_index; u4 attribute_length; u2 line_number_table_length; { u2 start_pc; u2 line_number; } line_number_table[line_number_table_length]; }` [4.7.12]	`... { // Code Attribute (Method Bar) ... [] { // Attributes Attr(#11) { // LineNumberTable [] { // LineNumberTable 0 // start_pc 2; // line_number } } // end LineNumberTable } // Attributes ... } // end Code Attribute ...`
`LocalVariableTable_attribute { u2 attribute_name_index; u4 attribute_length; u2 local_variable_table_length; { u2 start_pc; u2 length; u2 name_index; u2 descriptor_index; u2 index; } local_variable_table[local_variable_table_length]; }` [4.7.13]	`... { // Code Attribute (Method Bar) ... [] { // Attributes // LocalVariableTable Attr(#8) { [] { // LocalVariableTableEntry 0 //start_pc 1 //length #1 //name_cpx 1 //index } } // end LocalVariableTable } // Attributes ... } // end Code Attribute ...`

```
LocalVariableTypeTable_attribute {
 u2 attribute_name_index;
 u4 attribute_length;
 u2 local_variable_type_table_length;
 { u2 start_pc;
 u2 length;
 u2 name_index;
        u2 signature_index;
 u2 index;
 } local_variable_type_table
[local_variable_type_table_length];
}
```

[4.7.14]

```
...
     { // Code Attribute (Method Bar)
      ...
        [1] { // Attributes
// LocalVariableTypeTable
              Attr(#9) {
              [] {
//LocalVariableTypeTableEntry
                    0  //start_pc
                    1  //length
                   #14 //name_cpx
                   #17 //signature_cpx
                    4  //index
                  }
              }
// end LocalVariableTypeTable
         } // Attributes
       ...
      } // end Code Attribute
   ...
```

```
StackMapTable_attribute {
 u2 attribute_name_index;
 u4 attribute_length;
 u2 number_of_entries;
 stack_map_frame entries[number_of_entries];
}
```

[4.7.4]

```
 ...
     { // Code Attribute (Method Bar)
      ...
        [] { // Attributes
              Attr(#11) { // StackMapTable
              [] { // StackMapTable
 stack_map_info_1
 ...
 stack_map_info_n
                    }
              } // end StackMapTable
         } // Attributes
       ...
      } // end Code Attribute
   ...
```

StackMapInfo Frames

The following examples describe Stack Map Frames that may appear in a Stack Map Attribute.

Class File Spec	JCOD

`union stack_map_frame {`
`    same_frame;`
`    same_locals_1_stack_item_frame;`
`    same_locals_1_stack_item_frame_extended;`
`    chop_frame;`
`    same_frame_extended;`
`    append_frame;`
`    full_frame;`
`}`
 	
[4.7.4]	

```
same_frame {
    u1 frame_type = SAME; /* 0-63 */
}
```

[4.7.4]

```
  ...
    { // Code Attribute (Method Bar)
     ...
       [] { // Attributes
             Attr(#11) { // StackMapTable
             [] { // StackMapTable
                    ...
                     63; //same_frame (offset =
63)
                    ...
                  }
             } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
  ...
```

```
same_locals_1_stack_item_frame {
    u1 frame_type = SAME_LOCALS_1_STACK_ITEM;/* 64-
127 */
    verification_type_info stack[1];
}
```

[4.7.4]

```
 ...
    { // Code Attribute (Method Bar)
     ...
       [] { // Attributes
             Attr(#11) { // StackMapTable
             [] { // StackMapTable
                    ...

 //same_locals_1_stack_item_frame

 // (offset = 65 - 64 = 1)
                     65;
 // verification type
info

                    verification_info_1;
                    ...
                  }
             } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
 ...
```

```
same_locals_1_stack_item_frame_extended {
    u1 frame_type = SAME_LOCALS_1_STACK_ITEM_EXTENDED;
                                                /* 247
*/
    u2 offset_delta;
    verification_type_info stack[1];
}
```

[4.7.4]

```
...
    { // Code Attribute (Method Bar)
     ...
       [] { // Attributes
             Attr(#11) { // StackMapTable
             [] { // StackMapTable
                    ...

//same_locals_1_stack_item_frame_extended
                    247;
                    43L; // (offset = 43)

 // verification type info

                    verification_info_1;
                    ...
                  }
             } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
 ...
```

```
chop_frame {
    u1 frame_type = CHOP;                    /* 248-
250 */
    u2 offset_delta;
}
```

[4.7.4]

```
...
     { // Code Attribute (Method Bar)
      ...
        [] { // Attributes
              Attr(#11) { // StackMapTable
              [] { // StackMapTable
                    ...

 //chop_frame (last 1 local absent)
                    249;
                    43L; // (offset = 43)
                    ...
                   }
              } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
 ...
```

```
same_frame_extended {
    u1 frame_type = SAME_FRAME_EXTENDED;   /* 251 */
    u2 offset_delta;
}
```

[4.7.4]

```
...
     { // Code Attribute (Method Bar)
      ...
        [] { // Attributes
              Attr(#11) { // StackMapTable
              [] { // StackMapTable
                    ...
 //same_frame_extended
                    251;
                    43L; // (offset = 43)
                    ...
                   }
              } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
 ...
```

```
append_frame {
     u1 frame_type = APPEND;               /* 252-254
*/
     u2 offset_delta;
     verification_type_info locals[frame_type - 251];
}
```

[4.7.4]

```
...
     { // Code Attribute (Method Bar)
      ...
        [] { // Attributes
              Attr(#11) { // StackMapTable
              [] { // StackMapTable
                    ...
                    252; //append_frame
                    43L; // (offset = 43)
// verification type info

// numb verif info = 252 - 251 = 1

                    verification_info_1;
                    ...
                   }
              } // end StackMapTable
         } // Attributes
       ...
     } // end Code Attribute
 ...
```

```
union verification_type_info {
    Top_variable_info;                // = 0
    Integer_variable_info;            // = 1
    Float_variable_info;              // = 2
    Long_variable_info;               // = 3
    Double_variable_info;             // = 4
    Null_variable_info;               // = 5
    UninitializedThis_variable_info;  // = 6
    Object_variable_info;             // = 7 + cpx
    Uninitialized_variable_info;      // = 8 + u2
offset
}
```

[4.7.4]

Annotations and Annotation Elements

The following examples describe Annotations and Annotation Elements that may appear in an Annotations Attribute.

Class File Spec	JCOD

`annotation {`
`  u2 type_index;`
`  u2 num_element_value_pairs;`
`  {  u2 element_name_index;`
`  element_value value;`
`  }  element_value_pairs`
`[num_element_value_pairs]`
`}`
 	
[4.7.16]	

`element_value {`
`  u1 tag;`
`  union {`
`  u2 const_value_index;`
`  {  u2 type_name_index;`
`  u2 const_name_index;`
`  }  enum_const_value;`
`  u2 class_info_index;`
`  annotation annotation_value;`
`  {  u2 num_values;`
`  element_value values[num_values];`
`  }  array_value;`
`  }  value;`
`}`
 	
[4.7.16.1]
	`...`
	`  { // Method Bar`
	`  ...`
	`  [1] { // Attributes`
	`  Attr(#13) { //`
	`RuntimeVisibleAnnotations`
	`  [] { // annotations`
	`  {  //  annotation`
	`  #14;`
	`  [] { // element_value_pairs`
	`  }  //  element_value_pairs`
	`  }  //  annotation`
	`  }`
	`  } // end RuntimeVisibleAnnotations`
	`  ...`
	`  } // end Attributes`
	`  ...`
	`  } // Method Bar`