## Table of Contents

Eclipse Foundation Specification License .................................................. 1  
  Disclaimers ................................................................................. 2  
1. Goal ....................................................................................... 3  
2. Terminology ........................................................................ 4  
3. Approach ............................................................................. 5  
  3.1. Single Translation ............................................................... 5  
  3.2. Multiple Translations ............................................................ 5  
  3.3. Diagram ........................................................................... 6  
4. Scope .................................................................................... 7  
  4.1. Variables .......................................................................... 7  
  4.2. Multi-Level Source View ....................................................... 7  
  4.3. Finding Source Files .............................................................. 7  
  4.4. Multiple Source Files per Class File ................................... 7  
5. Source Map Format ................................................................. 8  
  5.1. General Format ................................................................. 8  
  5.2. Header ............................................................................. 8  
  5.3. StratumSection ................................................................. 8  
  5.4. FileSection ...................................................................... 9  
  5.5. LineSection ..................................................................... 9  
  5.6. VendorSection .................................................................. 11  
  5.7. EndSection ...................................................................... 11  
  5.8. Embedded Source Maps .................................................... 12  
  5.9. SMAP Syntax .................................................................. 12  
6. SMAP Resolution ..................................................................... 15  
  6.1. LineInfo Composition Algorithm ....................................... 16  
  6.2. Resolution Example ........................................................... 17  
7. JPDA Support .......................................................................... 21  
8. SourceDebugExtension Support .................................................. 22  
  8.1. SourceDebugExtension Access ........................................... 22  
  8.2. SourceDebugExtension Class File Attribute ..................... 22  
9. Example .................................................................................. 24  
  9.1. Input Source ..................................................................... 24  
  9.2. Language Processor ........................................................... 24  
  9.3. Post Processor .................................................................. 26  
  9.4. Debugging ................................................................. 26
Eclipse Foundation Specification License

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] Eclipse Foundation, Inc. <<url to this license>>"

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright © 2018, 2020 Eclipse Foundation. This software or document includes material copied from or derived from Jakarta ® Debugging Support for Other Languages https://jakarta.ee/specifications/debugging/2.0/"
Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the copyright holders or the Eclipse Foundation may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.
Chapter 1. Goal

A mechanism is needed by which programs executed under the Java™ virtual machine but written in languages other than the Java programming language, can be debugged with references to the original source (for example, source file and line number references).

Constraints:

• No change to the Java programming language.
• Optional change to Java programming language compiler.
• No change to JPDA clients (debuggers, ...) for basic functionality. With the exception being, tools that arbitrarily prohibit non-Java programming language source.
• Minimal change to Java virtual machine.
• No change to the Java platform class libraries.
## Chapter 2. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>final-source</td>
<td>The final form of source. This source will be compiled into a class file. Typically, final-source is Java programming language source.</td>
</tr>
<tr>
<td>translated-source</td>
<td>Source that will be translated by a language-processor into another language or form.</td>
</tr>
<tr>
<td>language-processor</td>
<td>Converts translated-source to final-source or to different translated-source.</td>
</tr>
<tr>
<td>class file</td>
<td>A collection of bytes in Java virtual machine class file format.</td>
</tr>
<tr>
<td>compiler</td>
<td>Compiler which converts final-source to class files. For example, javac is a compiler.</td>
</tr>
<tr>
<td>post-processor</td>
<td>Takes a class file and a Source Map File as input; generates a class file as output. Inserts a SourceDebugExtension attribute.</td>
</tr>
<tr>
<td>Source Map SMAP</td>
<td>Specifies the mapping of source between one or more sets of input source and the output source. See: Source Map Format.</td>
</tr>
<tr>
<td>Source Map File SMAP-file</td>
<td>Specifies the mapping created by a language-processor between translated-source input and the resultant output source. It is a Source Map stored in a file in Source Map Format.</td>
</tr>
<tr>
<td>stratum</td>
<td>A view of a class from a particular, named, programming language level.</td>
</tr>
<tr>
<td>JPD A</td>
<td>The Java Platform Debugger Architecture.</td>
</tr>
<tr>
<td>JDI</td>
<td>The Java Debug Interface, which is the high-level Java programming language interface of the Java Platform Debugger Architecture. See: JDI Specification.</td>
</tr>
<tr>
<td>SourceDebugExtension attribute</td>
<td>A Java virtual machine class file attribute that holds information about the source. In this case, the information is a Source Map in Source Map Format. The source is mapped to final-source (output source). The Source Map has potentially multiple sets of input source (strata). See: SourceDebugExtension Class File Attribute</td>
</tr>
</tbody>
</table>
Chapter 3. Approach

3.1. Single Translation

A language-processor translates translated-source to final-source (the case of translation from one translated-source to another translated-source, is addressed below). It also creates a second output, an SMAP-file, whose format is described in Source Map Format. This file describes the mapping between input source and output source (e.g. line number and source file).

The final-source generated by the language-processor is compiled by the compiler.

The post-processor takes the class file generated by the compiler and the SMAP-file as input. A SourceDebugExtension attribute containing the SMAP in the SMAP-file is added to the class file and the new class file is written.

Optionally, the compiler may take both final-source and the SMAP-file as input, and perform both compilation and installation of the SourceDebugExtension.

When the resultant program is debugged using a debugging tool based on the Java Debug Interface (JDI) of JPDA, the final-source line number information is converted to the specified language view (strata).

3.2. Multiple Translations

A language-processor might translate source into source which will become input to another language-processor, and so on. Eventually, after possibly many translated-source forms, final-source is produced. Each translation produces SMAP information. This information must be preserved and placed in context, so that each stratum can be mapped to the final-source.

A language-processor checks for an SMAP-file in a location parallel to that of the input source. For example, if the source repository is a file system and the input source is located at path name.extension then path name.extension.smap will be checked for an SMAP. The input SMAPs will be copied into the generated SMAP. See Embedded Source Maps for specifics on the embedding of input SMAPs.

In the case of multiple translations, the post-processor must resolve the embedded SMAPs. See SMAP Resolution.

Note that final-source need not be Java programming language source, as compilers for other languages may directly generate class files, including the SourceFile and LineNumberTable class file attributes. SMAPs and the mechanism presented here are still useful for handling multiple translations.

A programming language implementor, directly generating class files might also choose to generate SMAPs (thus functioning as both language-processor and compiler) since the SMAP is useful for describing source configurations (such as multiple source files per class file) which cannot be
represented with the `SourceFile` and `LineNumberTable` attributes. In this case, the input is translated-source and the final-source is represented in the attributes but is never generated.

### 3.3. Diagram

This diagram demonstrates data flow. The particular case shown has two levels of translation, with file inclusion on the second level (as is the case in the example in [SMAP Resolution](#)).

Where `TS` is translated-source and `FS` is final-source.
Chapter 4. Scope

4.1. Variables

The complexity of mapping semantics (like variable and data views) across languages is such that this feature has not been included in this version of the specification and will be considered for a future version.

4.2. Multi-Level Source View

The ability to choose the source level to view is addressed in this specification. These are referred to as strata.

4.3. Finding Source Files

Currently, final-source is found by combining the follow elements:

- A source path
- The package name converted to a directory path
- The source file name from a JDI call (derived from the SourceFile class file attribute)

Since existing debuggers use this mechanism (the only way for an existing debugger to find translated-source) each aspect must be addressed:

- The source path must be set-up to include translated-source directories
- Source must be placed in a directory corresponding to the package
- The JDI call must return the translated-source name.

For debuggers written against the new APIs, a new method has been added which returns the source path - this makes the translated-source directory structure flexible.

4.4. Multiple Source Files per Class File

When an inclusion mechanism is used, a class file will contain source from multiple translated-source files. The SourceFile attribute of class files only associates one source file with a class file which is one reason the approach of simply rewriting the SourceFile and LineNumberTable attributes had to be abandoned. The SMAP allows a virtually unlimited number of source files per stratum.
Chapter 5. Source Map Format

A Source Map (SMAP) describes a mapping between source positions in an input language (translated-source) and source positions in a generated output language. A view of the source through such a mapping is called a stratum. The SMAP-file contains an unresolved SMAP. The SourceDebugExtension class file attribute, when used as described in this document, contains an SMAP. The SMAP stored in a SourceDebugExtension attribute must be resolved, and thus will have no embedded SMAPs and will have the final-source language as the output language.

An SMAP consists of a header and one or more sections of mapping information.

There are currently seven types of section: stratum sections, file sections, line sections, vendor sections, end sections, and open and close embedded sections. New section types may be added in the future - to facilitate this, any unknown sections must be ignored without error.

The semantics of each section is discussed below. For clarity, an informal description of the syntax of each section is included in the discussion. See the formal SMAP syntax for syntax questions.

5.1. General Format

The SMAP consists of lines of Unicode text, with a concrete representation of UTF-8. Line termination is with line-feed, carriage-return or carriage-return followed by line-feed. Because SMAPs are included in class files, size of the SMAP was an important constraint on the format chosen for them.

5.2. Header

The first line of an SMAP is the four letters SMAP which identifies it as an SMAP. The next line is the name of the generated file. This name is without path information (and thus if the generated file is final-source, the name should match the SourceFile class file attribute). The last line of the header is the default stratum for this class. The default stratum is the stratum used when a debugger does not explicitly specify interest in another stratum. In an unresolved SMAP the default stratum can be unspecified (blank line). In a resolved SMAP the default stratum must be specified. A specified stratum must either be one represented with a stratum section or Java which indicates the standard final-source information should be used by default.

5.3. StratumSection

An SMAP may map more than one translated-source to the output source (the output source is final-source if the SMAP is in a SourceDebugExtension). A view of the source is a stratum (whether viewed as translated-source or final-source). Each translated-source language should have its own stratum section with a unique stratum name. The final-source stratum (named “Java”) is created automatically and should not have a stratum section. The stratum section should be followed by a file section and a line section which will be associated with that stratum.
The format of the section is simply the stratum section marker *S followed by the name of the stratum. The section ends with a line termination. One FileSection and one LineSection (in either order) must follow the StratumSection (before the next StratumSection or the EndSection). One or more VendorSections may follow a StratumSection. There must be at least one StratumSection.

5.4. FileSection

The file section describes the translated-source file names. Each line maps a file ID to a source name and, optionally, to a source path. File IDs are used only in the LineSection. The source name is the name of the translated-source. The source path is the path to the translated-source, the "/" symbol is translated to the local file separator. In the case where the source repository is a file system, source name is the file name (without directory information) and source path is a path name (often relative to one of the compilation source paths). For example: Bar.foo would be a source name, and here/there/Bar.foo would be a source path. The first file line denotes the primary file.

The format of the file section is the file section marker *F on a line by itself, followed by file information. File information has two forms, source name only and source name / source path. The source name only form is one line: the integer file ID followed by the source name. The source name / source path form is two lines: a plus sign +, file ID, and source name on the first line and the source path on the second. The file ID must be unique within the file section. A FileSection may only occur after a StratumSection. The FileName must have at least one character. The AbsoluteFileName, if specified, must have at least one character.

For example:

```
*F
  + 1 Foo.xyz
  here/there/Foo.xyz
  2 Incl.xyz
```

declares two source files. File ID #1 has source name Foo.xyz and source path here/there/Foo.xyz. File ID #2 has source name Incl.xyz and a source path to be computed by the debugger.

5.5. LineSection

The line section associates line numbers in the output source with line numbers and source names in the input source.

The format of the line section is the line section marker *L on a line by itself, followed by the lines of LineInfo. Each LineInfo has the form:

```
InputStartLine # LineFileID , RepeatCount : OutputStartLine , OutputLineIncrement
```
where all but

```
InputStartLine : OutputStartLine
```

are optional.

A range of output source lines is mapped to a single input source line. Each LineInfo describes RepeatCount of these mappings. OutputLineIncrement specifies the number of lines in the output source range; this line increment is applied to each mapping in the LineInfo. The source file containing the input source line is specified by LineFileID via the FileSection.

More precisely, for each \( n \) between zero and

```
RepeatCount - 1
```

the input source line number

```
InputStartLine + n
```

maps to the output source line numbers from

```
OutputStartLine + (n * OutputLineIncrement)
```

through

```
OutputStartLine + ((n + 1) * OutputLineIncrement) - 1
```

If absent, RepeatCount and OutputLineIncrement default to one. If absent, LineFileID defaults to the most recent value (initially zero).

The first line of a file is line one. RepeatCount is greater than or equal to one. Each LineFileID must be a file ID present in the FileSection. InputStartLine is greater than or equal to one. OutputStartLine is greater than or equal to one. OutputLineIncrement is greater than or equal to zero. A LineSection may only occur after a StratumSection.

For example:
Creates this mapping:

<table>
<thead>
<tr>
<th>Input Source</th>
<th>Output Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>Begin Line</td>
</tr>
<tr>
<td>123</td>
<td>207</td>
</tr>
<tr>
<td>130</td>
<td>210</td>
</tr>
<tr>
<td>131</td>
<td>211</td>
</tr>
<tr>
<td>132</td>
<td>212</td>
</tr>
<tr>
<td>140</td>
<td>250</td>
</tr>
<tr>
<td>160</td>
<td>300</td>
</tr>
<tr>
<td>161</td>
<td>302</td>
</tr>
<tr>
<td>162</td>
<td>304</td>
</tr>
</tbody>
</table>

Note that multiple LineInfo may map multiple input source lines to a single output source line, when such a LineSection is being used to map output source lines to input source lines, a first matching LineInfo rule applies.

Note also that multiple LineInfo may map a single input source line to a multiple, possibly disjoint, output source lines, when such a LineSection is being used to map input source lines to output source lines, a first matching LineInfo rule again applies.

### 5.6. VendorSection

The vendor section is for vendor specific information.

The format is *V on the first line to mark the section. The second line is the vendor ID which is formed by the same rules by which unique package names are formed in the Java language specification, third edition (§7.7) Unique Package Names.

### 5.7. EndSection

The end section marks the end of an SMAP, it consists simply of a *E marker. The end section must be the last line of an SMAP.
5.8. Embedded Source Maps

The OpenEmbeddedSection marks the beginning and CloseEmbeddedSection the end of a set of EmbeddedSourceMaps. These SMAPs correspond to the input source for a language-processor. The stratum of the language-processor is indicated on both sections. These sections must not occur in a resolved SMAP.

The format is the *O marker and the name of the output stratum on the first line. This is followed by the set of embedded SMAPs. The embedded SMAPs are included "whole" - from the SMAP to the EndSection *E marker - inclusive. Finally, the *C marker and the name of the output stratum on the last line terminates the embedded SMAPs.

5.9. SMAP Syntax

SMAP:  
  Header { Section } EndSection

Header:  
  ID OutputFileName DefaultStratumId

ID:  
  SMAP CR

OutputFileName:  
  NONASTERISKSTRING CR

DefaultStratumId:  
  NONASTERISKSTRING CR

Section:  
  StratumSection
  FileSection
  LineSection
  EmbeddedSourceMaps
  VendorSection
  FutureSection

EmbeddedSourceMaps:  
  OpenEmbeddedSection { SMAP } CloseEmbeddedSection

OpenEmbeddedSection:  
  *O StratumID CR

CloseEmbeddedSection:  
  *C StratumID CR
StratumSection:
  *S StratumID CR

StratumID:
  NONASTERISKSTRING

LineSection:
  *L CR { LineInfo }

LineInfo:
  InputLineInfo : OutputLineInfo CR

InputLineInfo:
  InputStartLine, RepeatCount
  InputStartLine

OutputLineInfo:
  OutputStartLine, OutputLineIncrement
  OutputStartLine

InputStartLine:
  NUMBER
  NUMBER # LineFileID

LineFileID:
  FileID

RepeatCount:
  NUMBER

OutputStartLine:
  NUMBER

OutputLineIncrement:
  NUMBER

FileSection:
  *F CR { FileInfo }

FileInfo:
  FileID FileName CR
  + FileID FileName CR AbsoluteFileName CR

FileID:
  NUMBER

FileName:
  NONASTERISKSTRING
Where \( \{x\} \) denotes zero or more occurrences of \( x \). And where the terminals are defined as follows (whitespace is a sequence of zero or more spaces or tabs):

<table>
<thead>
<tr>
<th>Terminals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONASTERISKSTRING</td>
<td>Any sequence of characters (excluding the terminal carriage-return or newline) which does not start with &quot;*&quot;. Leading whitespace is ignored.</td>
</tr>
<tr>
<td>NUMBER</td>
<td>Non negative decimal integer. The number is terminated by the first non-digit character. Leading and trailing whitespace is ignored.</td>
</tr>
<tr>
<td>CR</td>
<td>a line terminator: carriage-return, carriage-return followed by new-line or new-line.</td>
</tr>
<tr>
<td>OTHERCHAR</td>
<td>Any character (other than carriage-return, new-line, space or tab) not already used as a section header (not S,F,L,V,O,C or E).</td>
</tr>
<tr>
<td>VENDORID</td>
<td>A sequence of characters that identifies a vendor. The name is formed by the same rules that unique package names are formed in the Java language specification. Leading and trailing whitespace is ignored. The terminal carriage-return or new-line is excluded.</td>
</tr>
</tbody>
</table>
Chapter 6. SMAP Resolution

Before the SMAP in a SMAP-file can be installed into the SourceDebugExtension attribute it must be resolved into an SMAP with no embedded SMAPs and with final-source as the output source. A set of embedded SMAPs is specific to a stratum and is resolved in the context of the matching StratumSection in the outer SMAP. The resolved SMAP includes StratumSections computed from each set of embedded SMAPs as well as the unchanged StratumSections of the outer SMAP. If embedded SMAPs are nested, the inner-most is resolved first.

The structure of an SMAP with embedded SMAPs is as follows:

```
SMAP
...
*O B
SMAP
...
*S A
...
*E
*C B
...
*S B
...
*E
```

The structure is a set of embedded SMAPs (for a stratum, here named B), an outer StratumSection (for B), and an embedded SMAP with a StratumSection (for a stratum, here named A). Note that: there may be many sets of embedded SMAPs, many embedded SMAPs within the set of embedded SMAPs, and many StratumSections within an SMAP. A StratumSection maps source information from its stratum to an output stratum. Thus, the embedded StratumSection maps stratum A to stratum B. We know it is mapped to stratum B because the set of embedded SMAPs for stratum B corresponds to the input for the language-processor for B. The outer StratumSection maps stratum B to its output stratum (let’s call this stratum C), if the shown SMAP is the outer-most SMAP then stratum C is the final-source stratum. The purpose of resolution is to create a non-embedded StratumSection for A which maps to C (all StratumSections within an SMAP must map to the same output stratum, in a resolved outer-most SMAP all StratumSections will map to the final-source stratum). This is done by composing the mapping in the embedded StratumSection (from A to B) with the mapping in the outer StratumSection (from B to C). Since there may be many embedded StratumSections for A, these sections must be merged.

A StratumSection is computed for each stratum present in the embedded SMAPs. The computed StratumSection is the merge of each embedded StratumSection, for that stratum. Line number information is composed with the line number information of the outer StratumSection (note that the embedded StratumSections cannot be for the same stratum as the outer StratumSection). Specifically, a computed StratumSection consists of a merged FileSection, a composed LineSection, and direct copies.
of any VendorSections or unknown sections. The merged FileSection includes each unique FileInfo, with FileIDs reassigned to be unique. The composition the LineSections is described in the algorithm below.

6.1. LineInfo Composition Algorithm

The following pseudo-code sketches the algorithm for resolving LineInfo in embedded SMAPs. LineInfo resolution is by composition - discussed above. An embedded LineInfo which maps stratum $A$ to stratum $B$ is composed with an outer LineInfo which maps stratum $B$ to stratum $C$ to create a new resolved LineInfo which maps stratum $A$ to stratum $C$.

The SMAPs and their components are marked by subscript:

- Embedded SMAP - level$_E$
- Outer StratumSection - level$_O$
- Resolved computed StratumSection - level$_R$

The inputs and outputs of the algorithm are LineInfo tuples. Line information is represented in this algorithm in its LineInfo format which is discussed in the LineSection. This algorithm is invoked for each LineInfo$_E$ in each embedded SMAP.
ResolveLineInfo:

\[
\text{InputStartLine}_E \# \text{LineFileID}_E, \text{RepeatCount}_E : \text{OutputStartLine}_E, \text{OutputLineIncrement}_E
\]

as follows:

if RepeatCount$_E > 0$ then {
  for each LineInfo$_0$ in the stratum of the embedded SMAP:
    InputStartLine$_0 \# \text{LineFileID}_0, \text{RepeatCount}_0 : \text{OutputStartLine}_0, \text{OutputLineIncrement}_0$
    which includes OutputStartLine$_E$
    that is, InputStartLine$_0 + N == \text{OutputStartLine}_E$
    for some offset into the outer input range $N$ where $0 \leq N < \text{RepeatCount}_0$
  and for which LineFileID$_0$ has a sourceName matching the embedded SMAP’s OutputFileName {
    compute the number of outer mapping repetitions which can be applied
    available := RepeatCount$_0 - N$
    compute the number of embedded mapping repetitions which can be applied
    completeCount := floor(available / OutputLineIncrement$_E$) min RepeatCount$_E$;
    if completeCount > 0 then {
      output resolved LineInfo
      InputStartLine$_E \# \text{uniquify(LineFileID}_0), \text{completeCount} :$
      (\text{OutputStartLine}_0 + (N \times \text{OutputLineIncrement}_0)),
      (\text{OutputLineIncrement}_E \times \text{OutputLineIncrement}_0);$
      ResolveLineInfo
      (InputStartLine$_E + \text{completeCount}) \# \text{LineFileID}_E, (\text{RepeatCount}_E - \text{completeCount}) :$
      (\text{OutputStartLine}_E + \text{completeCount} \times \text{OutputLineIncrement}_E), \text{OutputLineIncrement}_E$
    } else {
      output resolved LineInfo
      InputStartLine$_E \# \text{uniquify(LineFileID}_0), 1 :$
      (\text{OutputStartLine}_0 + (N \times \text{OutputLineIncrement}_0)), \text{available} ;$
      ResolveLineInfo
      InputStartLine$_E \# \text{LineFileID}_0, 1 :$
      (\text{OutputStartLine}_E + \text{available}), (\text{OutputLineIncrement}_E - \text{available});$
      ResolveLineInfo
      (InputStartLine$_E + 1) \# \text{LineFileID}_E, (\text{RepeatCount}_E - 1):$
      (\text{OutputStartLine}_E + \text{OutputLineIncrement}_E), \text{OutputLineIncrement}_E$
    }
  }
}

where \text{uniquify} converts a LineFileID$_E$ to a corresponding LineFileID$_R$

### 6.2. Resolution Example

The following example demonstrates resolution with this algorithm. The \text{general example} will provide context before walking through this example. In this example, \text{Incl.bar} is included by \text{Hi.bar}, but each is the result of a prior translation.
If the unresolved SMAP (in `Hi.java.smap`) is as follows:

<table>
<thead>
<tr>
<th>SMAP</th>
<th>Outer Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi.java</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
</tr>
<tr>
<td><strong>O Bar</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMAP</th>
<th>OpenEmbeddedSection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi.bar</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
</tr>
<tr>
<td><strong>S Foo</strong></td>
<td></td>
</tr>
<tr>
<td><strong>F</strong></td>
<td></td>
</tr>
<tr>
<td>1 Hi.foo</td>
<td></td>
</tr>
<tr>
<td><strong>L</strong></td>
<td></td>
</tr>
<tr>
<td>1#1,5:1,2</td>
<td></td>
</tr>
<tr>
<td><strong>E</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMAP</th>
<th>Embedded SMAP (Hi.bar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl.bar</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
</tr>
<tr>
<td><strong>S Foo</strong></td>
<td></td>
</tr>
<tr>
<td><strong>F</strong></td>
<td></td>
</tr>
<tr>
<td>1 Incl.foo</td>
<td></td>
</tr>
<tr>
<td><strong>L</strong></td>
<td></td>
</tr>
<tr>
<td>1#1,2:1,2</td>
<td></td>
</tr>
<tr>
<td><strong>E</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>C Bar</strong></th>
<th>CloseEmbeddedSection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S Bar</strong></td>
<td></td>
</tr>
<tr>
<td><strong>F</strong></td>
<td></td>
</tr>
<tr>
<td>1 Hi.bar</td>
<td></td>
</tr>
<tr>
<td>2 Incl.bar</td>
<td></td>
</tr>
<tr>
<td><strong>L</strong></td>
<td></td>
</tr>
<tr>
<td>1#1:1</td>
<td></td>
</tr>
<tr>
<td>1#2,4:2</td>
<td></td>
</tr>
<tr>
<td>3#1,8:6</td>
<td></td>
</tr>
</tbody>
</table>

| **Outer StratumSection** |                                |

---

6.2. Resolution Example

Jakarta Debugging Support for Other Languages
The merged level_	extsubscript{r} FileSection is (in stratum 	exttt{Foo}):

\begin{itemize}
  \item Hi.foo
  \item Incl.foo
\end{itemize}

The computation proceeds as follows:

<table>
<thead>
<tr>
<th>LineInfo\textsubscript{E}</th>
<th>LineInfo\textsubscript{E} recursion 1</th>
<th>LineInfo\textsubscript{E} recursion 2</th>
<th>matching outer LineInfo\textsubscript{O}</th>
<th>resolved LineInfo\textsubscript{R}</th>
<th>discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1#1,5:1,2</td>
<td></td>
<td></td>
<td>1#1</td>
<td>1#1,1:1,1</td>
<td>ResolveLineInfo is called for 1#1,5:1,2 (from the first embedded SMAP - OutputFileName is 	exttt{Hi.bar}). 1#1:1 is found as the outer StratumSection LineInfo\textsubscript{O} with InputStartLine\textsubscript{O} of 1 and LineFileIDO has a sourceName matching 	exttt{Hi.bar}. \texttt{N} is 0, and completeCount is 0, thus the else branch is taken. available is 1 and thus output is 1#1,1:1,1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1#1,1:2,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The remaining half of the initial LineInfo\textsubscript{O} mapping then must be resolved recursively, but there is no match and it is ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2#1,4:3,2</td>
<td>3#1,8:6</td>
<td></td>
<td>2#1,4:6,2</td>
<td></td>
<td>The remaining mappings are also handled recursively. There is a matching LineInfo\textsubscript{O}. \texttt{N} is 0, and completeCount is 4, thus the if branch is taken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6#1,0:11,2</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
<td>The recursive resolve descends deeper but does nothing since all of RepeatCount\textsubscript{E} has been mapped. The first LineInfo\textsubscript{E} is now resolved. Since it had only one LineInfo\textsubscript{E} the first SMAP is also resolved.</td>
</tr>
</tbody>
</table>

Jakarta Debugging Support for Other Languages
### 6.2. Resolution Example

<table>
<thead>
<tr>
<th>$\text{LineInfo}_E$</th>
<th>$\text{LineInfo}_E$ recursion 1</th>
<th>$\text{LineInfo}_E$ recursion 2</th>
<th>matching outer $\text{LineInfo}_o$</th>
<th>resolved $\text{LineInfo}_R$</th>
<th>discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1#1,2:1,2</td>
<td></td>
<td></td>
<td>1#2,4:2</td>
<td>1#2,2:2,2</td>
<td>Now for the second SMAP (OutputFileName is <code>Incl.bar</code>). FileID$_E$ 1 in this SMAP is <code>Incl.foo</code> which corresponds to the remapped FileID$_R$ 2. So the matching $\text{LineInfo}_o$ is 1#2,4:2. $N$ is 0, and $\text{completeCount}$ is 2, so the if branch is taken.</td>
</tr>
<tr>
<td>3#1,0:5,2</td>
<td></td>
<td>n/a</td>
<td></td>
<td></td>
<td>The recursive resolve does nothing since all the maps have been handled. Resolution is complete.</td>
</tr>
</tbody>
</table>

The resultant resolved SMAP is:

```
SMAP
Hi.java
Java
*S Foo
*F
1 Hi.foo
2 Incl.foo
*L
1#1,1:1,1
2#1,4:6,2
1#2,2:2,2
*S Bar
*F
1 Hi.bar
2 Incl.bar
*L
1#1:1
1#2,4:2
3#1,8:6
*E
```
Chapter 7. JPDA Support

The Java Platform Debugger Architecture in the Java 1.4 release was extended in support of debugging other languages. The new APIs and APIs with comments changed to include reference to strata are listed below:

<table>
<thead>
<tr>
<th>New APIs</th>
<th>APIs with Changed Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSourceDebugExtension</td>
<td></td>
</tr>
<tr>
<td>JDWP - ReferenceType (2) Command Set</td>
<td></td>
</tr>
<tr>
<td>SourceDebugExtension Command (12)</td>
<td></td>
</tr>
<tr>
<td>JDWP - VirtualMachine (1) Command Set</td>
<td></td>
</tr>
<tr>
<td>SetDefaultStratum Command (19)</td>
<td></td>
</tr>
<tr>
<td>JDI - VirtualMachine interface</td>
<td></td>
</tr>
<tr>
<td>void setDefaultStratum(String stratum)</td>
<td></td>
</tr>
<tr>
<td>String getDefaultStratum()</td>
<td></td>
</tr>
<tr>
<td>JDI - ReferenceType interface</td>
<td></td>
</tr>
<tr>
<td>String sourceNames(String stratum)</td>
<td>String sourceName()</td>
</tr>
<tr>
<td>String sourcePaths(String stratum)</td>
<td></td>
</tr>
<tr>
<td>List allLineLocations(String stratum, String sourceName)</td>
<td>List allLineLocations()</td>
</tr>
<tr>
<td>List locationsOfLine(String stratum, String sourceName, int lineNumber)</td>
<td>List locationsOfLine(int lineNumber)</td>
</tr>
<tr>
<td>List availableStrata()</td>
<td></td>
</tr>
<tr>
<td>String defaultStratum()</td>
<td></td>
</tr>
<tr>
<td>String sourceDebugExtension()</td>
<td></td>
</tr>
<tr>
<td>JDI - Method interface</td>
<td></td>
</tr>
<tr>
<td>List allLineLocations(String stratum, String sourceName)</td>
<td>List allLineLocations()</td>
</tr>
<tr>
<td>List locationsOfLine(String stratum, String sourceName, int lineNumber)</td>
<td>List locationsOfLine(int lineNumber)</td>
</tr>
<tr>
<td>JDI - Location interface</td>
<td></td>
</tr>
<tr>
<td>class comment (strata defined)</td>
<td></td>
</tr>
<tr>
<td>int lineNumber(String stratum)</td>
<td>int lineNumber()</td>
</tr>
<tr>
<td>String sourceName(String stratum)</td>
<td>String sourceName()</td>
</tr>
<tr>
<td>String sourcePath(String stratum)</td>
<td></td>
</tr>
<tr>
<td>String sourcePath()</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 8. SourceDebugExtension Support

Debugger applications frequently need debugging information about the source that exceeds what is delivered by the existing Java™ Virtual Machine class file attributes (SourceFile, LineNumber, and LocalVariable). This is particularly true for debugging the source of other languages. In a distributed environment side files may not be accessible, the information must be directly associated with the class.

The solution is the addition of a class file attribute which holds a string. The string contains debugging information in a standardized format which allows for evolution and vendor extension.

8.1. SourceDebugExtension Access

This string is made opaquely accessible at the three layers of the Java Platform Debugger Architecture (JPDA):

<table>
<thead>
<tr>
<th></th>
<th>GetSourceDebugExtension(jclass clazz, char **sourceDebugExtensionPtr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JVMDI</td>
<td>GetSourceDebugExtension(jclass clazz, char **sourceDebugExtensionPtr)</td>
</tr>
<tr>
<td>JDWP</td>
<td>SourceDebugExtension Command (12) in the ReferenceType (2) Command Set</td>
</tr>
<tr>
<td>JDI</td>
<td>String sourceDebugExtension() in the ReferenceType interface</td>
</tr>
</tbody>
</table>

8.2. SourceDebugExtension Class File Attribute

Java virtual machine class file attributes are described in section 4.7 of the The Java Virtual Machine Specification. The definition of the added attribute is in the context of The Java Virtual Machine Specification:
The `SourceDebugExtension` attribute is an optional attribute in the `attributes` table of the `ClassFile` structure. There can be no more than one `SourceDebugExtension` attribute in the `attributes` table of a given `ClassFile` structure.

The `SourceDebugExtension` attribute has the following format:

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

The items of the `SourceDebugExtension_attribute` structure are as follows:

**attribute_name_index**

The value of the `attribute_name_index` item must be a valid index into the `constant_pool` table. The `constant_pool` entry at that index must be a `CONSTANT_Utf8_info` structure representing the string "SourceDebugExtension".

**attribute_length**

The value of the `attribute_length` item indicates the length of the attribute, excluding the initial six bytes. The value of the `attribute_length` item is thus the number of bytes in the `debug_extension[]` item.

**debug_extension[]**

The `debug_extension` array holds a string, which must be in UTF-8 format. There is no terminating zero byte. The string in the `debug_extension` item will be interpreted as extended debugging information. The content of this string has no semantic effect on the Java Virtual Machine.
Chapter 9. Example

The example below shows how the process described above would apply to a tiny JSP program.

9.1. Input Source

The input consists of two JSP files, the first is Hello.jsp:

```html
1. <HTML>
2. <HEAD>
3. <TITLE>Hello Example</TITLE>
4. </HEAD>
5. <BODY>
6. < %@ include file="greeting.jsp" %>
7. </BODY>
8. </HTML>
```

The second JSP file is the included file greeting.jsp:

```html
1. Hello There!<P>
2. Goodbye on <%= new Date() %>
```

9.2. Language Processor

When a JSP compiler (the language-processor) compiles these files it will produce two outputs - a Java programming language source file and a SMAP-file. The generated Java programming language source file is HelloServlet.java:

```java
1. import jakarta.servlet.*;
2. import jakarta.servlet.http.*;
3. public class HelloServlet extends HttpServlet {
4.     public void doGet(HttpServletRequest request,
5.                     HttpServletResponse response)
6.         throws ServletException, IOException {
7.             response.setContentType("text/html");
8.             PrintWriter out = response.getWriter();
```
The generated SMAP-file is `HelloServlet.java.smap`:

```
SMAP
HelloServlet.java
JSP
*S JSP
*F
1 Hello.jsp
2 greeting.jsp
*L
1#1,5:10,2
1#2,2:20,2
7#1,2:24,2
*E
```

A couple things are interesting to note about this SMAP — the user has chosen to make JSP the default
stratum (perhaps by a command line option) and even though there are ten lines of input source and 29 lines of generated source, only three LineInfo lines describe the transformation: the first and last are for the lines before and after the include (respectively) and the middle is for the included file greeting.jsp.

The three LineInfo lines describe these mappings:

1#1,5:10,2
Hello.jsp: line 1 -> HelloServlet.java: lines 10, 11
line 2 -> lines 12, 13
line 3 -> lines 14, 15
line 4 -> lines 16, 17
line 5 -> lines 18, 19

1#2,2:20,2
greeting.jsp: line 1 -> HelloServlet.java: lines 20, 21
line 2 -> lines 22, 23

7#1,2:24,2
Hello.jsp: line 7 -> HelloServlet.java: lines 24, 25
line 8 -> lines 26, 27

9.3. Post Processor

Next HelloServlet.java is compiled by a Java programming language compiler (for example javac) producing the class file HelloServlet.class. Then the post-processor is run. It takes HelloServlet.class and HelloServlet.java.smap as input. It creates a SourceDebugExtension attribute whose content is the SMAP in HelloServlet.java.smap and rewrites HelloServlet.class with this attribute.

9.4. Debugging

Now the program is run under the control of a debugger (which is a client of JDI). Let’s say we are stepping through this code and the debugger has just received a JDI StepEvent for the line that is just about to output <BODY>. The debugger’s code might look like this (the StepEvent is in the variable stepEvent):

```java
Location location = stepEvent.location();
String sourceName = location.sourceName("Java");
int lineNumber = location.lineNumber("Java");
displaySource(sourceName, lineNumber);
```

where displaySource is a debugger routine that displays a source location. Because the Java stratum has been specified sourceName would be HelloServlet.java, the lineNumber would be 19 and the
displayed line would be:

```java
out.println("<BODY>");
```

However, if `sourceName` and `lineNumber` were derived as follows:

```java
String sourceName = location.sourceName("JSP");
int lineNumber = location.lineNumber("JSP");
```

Since the JSP stratum has been specified, `sourceName` would be `Hello.jsp`, the `lineNumber` would be 5 and the displayed line would be:

```html
<BODY>
```

This occurs because the `SourceDebugExtension` attribute was stored when the VM read `HelloServlet.class` and it was retrieved with the `SourceDebugExtension` JDWP command which in turn caused the JVMDI function call `GetSourceDebugExtension`. The SMAP in the `SourceDebugExtension` was parsed which provided the above transformation of source location. Specifically, the line:

```
1#1,5:10,2
```

is the basis of this transformation - which refers to FileId #1

```
1 Hello.jsp
```

and whence the `sourceName` information. Since the default stratum specified in the SMAP is JSP, the code:

```java
String sourceName = location.sourceName();
int lineNumber = location.lineNumber();
```

would have the same effect. Since this is the form code would have taken before these extensions were introduced, existing debuggers can be utilized if they are run under the new implementation of JDI.