jMIBC™

Version 1.0

Java-based MIB Compiler

User's Guide

jSNMP Enterprises

jMIBC™ User's Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Credits and Acknowledgments

Specials thanks to the following people who contributed to the success of this project:

Amanda Goldner, Jim Pickering, Beckie White, Wes Strickland, and Jim Mortensen.

Contents

jMIBC™

Java-based MIB Compiler

User's Guide

Introduction

This manual provides the reader with information about the use of jMIBC, jSNMP Enterprises' Java-based Management Information Base (MIB) Compiler. It contains both installation and usage information as well as examples.

Description of jMIBC

SNMP Object Identifiers (OID) are difficult to remember. Users may prefer to make a SNMP request with the name associated with an OID instead of the OID's dotted decimal notation (e.g., ifAdminStatus.1 instead of 1.3.6.1.2.1.2.2.1.7.1). jMIBC takes a series of MIB files and produces a dictionary file that is used by the jSNMP Enterprise 3.x SnmpMIBService to translate OIDs to/from common names, to retrieve an OID's status, access, type, 'abstract' type, and description, and to translate OID enumerated values. Note that beginning with jSNMP Enterprise 3.2, it is no longer necessary to precompile MIBs to dictionary files with jMIBC, as MIBs can be loaded at runtime with the new jMIBC.loadMib() method. For a complete example that loads a MIB at runtime, see SnmpV1GetSysInfo.java in the jSNMP Enterprise examples directory.

jMIBC is written in Java and is therefore platform independent, thus it will run on any machine. jMIBC is a self-contained package and requires no other runtime libraries to run. The grammar of the MIB definition has been loosened within jMIBC so that a greater number of MIB files can be parsed without modification. jMIBC is a two pass compiler. In the first pass, it will pick up all definitions. In the second pass, all OIDs will be resolved and all SYNTAX types will be resolved to their lowest denominator.

Target Audience

jMIBC is targeted to users of jSNMP Enterprise 3.x. Using jMIBC to build dictionary files will give users the opportunity to take advantage of the SnmpMIBService interface in jSNMP Enterprise 3.x to translate OIDs.

Users should be familiar with the concept of OIDs for referencing data. In addition, users should be comfortable with MIB structures for describing the data to be managed.

Installation

Requirements

jMIBC requires a JDK™ 1.1.x or later compatible development and runtime environment.

Installation

jMIBC is a JAR packaged application. The JAR file can be installed anywhere on a system.

Using jMIBC

Using with JDK1.1

If you are using JDK 1.1, you will need to set the CLASSPATH environment variable to run jMIBC. For example, this could be accomplished with the following command:

```
>set CLASSPATH=C:\jSNMP\jMIBC.jar;%CLASSPATH%
```

for Win32 systems, or

```
>export CLASSPATH=/jSNMP/jMIBC.jar:$CLASSPATH
```

for Unix systems.

Running jMIBC can be accomplished with the following command:

```
>java jMIBC [options] <file>
```

Using with JDK1.2 and JDK1.3

For JDK1.2 and JDK1.3, you can follow the instructions for JDK1.1, or you can run jMIBC without setting the `CLASSPATH` environment variable. For example, this can be accomplished with the following command:

```
>java –jar jMIBC.jar [options] <file>
```

The `–jar` flag tells the Java interpreter that the application is packaged in the JAR file format.

Options

Name	Definition	Notes	Example
-b definition	Predefines an OID. This allows the user to define an OID without including all dependent MIB files.	Definitions must be in order of dependency.	-bmib-2=mgmt.1
-d	Allow duplicate definitions of identifiers or types without printing an error. No comparisons are made. Also allows syntax definitions to be undefined without printing an error.	If not specified, if an object is redefined and the definitions differ, an error is reported. In any case, the latest definition is the one that is accepted.	-d
-i mibfilename	Include a MIB file for definition but do not include the contents of the file in the output.	The –i option must precede each file that is to be included.	-iSNMPv2-CONF.my
-o outputfilename	The file name where the output will be written.	If not specified, output is written to standard output.	-oIF-MIB.jmib
-smi type	Defines a type to be "intrinsic."	The "smi" option defines a type to be valid.	-smiDisplayString
<file>	A list of MIB files to be processed separated by a space.	A single MIB file may contain multiple definitions but a single definition cannot span across multiple files.	

Options may appear in any order on the command line with the exception of the dependency order.

Input Files

To use jMIBC, you must first have one or more MIB files. MIB files can be obtained at various locations and are often part of an RFC document. Before any MIB derived from a RFC can be used as input to jMIBC, the RFC must be edited to remove the extraneous text leaving only the ASN.1 definitions. jMIBC will fail if a RFC has not been edited.

The full path name of the file must be specified on the command line or the file must exist in the current directory. jMIBC does not recognize the concept of an include path in which to look for the MIB files.

Restrictions

MIB files must be syntactically correct. If parsing errors are encountered, the MIB file must be corrected, though jMIBC does relax the following syntax rules:

- allow underscores in object names
- allow enumerations that begin with upper case letters

The following are syntax rules that jMIBC cannot handle causing it to terminate and generate an error:

- type names that begin with a lower case letter
- identifiers that begin with an upper case letter
- enumerations that begin with a number

jMIBC Examples

Complete MIB Definition

The following example illustrates compiling the SNMPv2 MIB file IF-MIB.my into the dictionary file IF-MIB.jmib:

```
>java -jar jMIBC.jar IF-MIB.my -o IF-MIB.jmib
```

The following example illustrates compiling the SNMPv1 MIB file BRIDGE-MIB.my into the dictionary file BRIDGE-MIB.jmib:

```
>java -jar jMIBC.jar BRIDGE-MIB.my -o BRIDGE-MIB.jmib
```

> *NOTE The MIB compiler will look for the files referenced in the IMPORT statement in the MIB file in the directory where the input file exists. In the examples above, the current directory is the directory that is searched to resolve names.*

Forcing Include Files for a MIB Definition

There are cases where the MIB compiler will not be able to find the files that are referenced in an IMPORT statement in the MIB file. In these cases, the dependent MIB files will need to be included using the `–i` option.

The following example illustrates compiling the SNMPv2 MIB file IF-MIB.my into the dictionary file IF-MIB.jmib by including all dependent MIBs via the `-i` option:

```
>java -jar jMIBC.jar -iSNMPv2-SMI.my -iSNMPv2-TC.my \
-iSNMPv2-CONF.my -iSNMPv2-MIB.my -iIANAifType-MIB.my IF-MIB.my \
-o IF-MIB.jmib
```

The following example illustrates compiling the SNMPv1 MIB file BRIDGE-MIB.jmib into the dictionary file BRIDGE-MIB.jmib by including all dependent MIBs via the `-i` option:

```
>java -jar jMIBC.jar -iRFC1155-SMI.my -iRFC1213-MIB.my \
BRIDGE-MIB.my -o BRIDGE-MIB.jmib
```

> *NOTE: if the –o <filename> option is not specified, output will be directed to standard output*

Partial MIB Definition

The following two examples illustrate compiling the SNMPv1 MIB file BRIDGE-MIB.my onto standard output by **not** including all dependent MIB files via the `-i` option. This method would most likely be used when not all the MIB files are available.

```
>java -jar jMIBC.jar -bmib-2=mgmt.1 -iRFC1155-SMI.my \
BRIDGE-MIB.my
```

```
>java -jar jMIBC.jar –d -bmib-2=mgmt.1 BRIDGE-MIB.my
```

The first example shows that the dependent MIB file RFC1213-MIB has not been included via the `-i` option. However, the necessary OID definition for `mib-2` has been specified with the `-b` option. The second example is identical to the first except for the exclusion of the RFC1155-SMI.my MIB and the inclusion of the `-d` option. The `-d` option is necessary to suppress the error about the missing `Counter` type that would have been defined if the MIB file RFC1155-SMI.my had been included.

Output

jMIBC output is composed of `name=value` pairs. Each name has a mapping to an OID. In addition, each name will have a type associated with it. Valid types are `regPt`, `OID`, `Table`, `Row`, `Enum`, and any valid defined ASN.1 SYNTAX.

regPt

A `regPt` is a registration point and is the type associated with a simple OID identified by the OBJECT IDENTIFIER declaration. For example, `ifMIBObjects` is defined as an `OBJECT IDENTIFIER` in the SNMPv2 MIB file IF-MIB.my and the jMIBC file IF-MIB.jmib.

MIB definition

```
ifMIBObjects OBJECT IDENTIFIER ::= { ifMIB 1 }
```

jMIBC output

```
ifMIBObjects=1.3.6.1.2.1.31.1
ifMIBObjects.TYPE=regPt
```

Table

An object is declared a `Table` if its `SYNTAX` is declared to be `SEQUENCE OF`. In the following example, `ifXTable` is defined with a `SYNTAX` of `SEQUENCE OF IFXEntry`.

MIB definition

```
ifXTable         OBJECT-TYPE
    SYNTAX       SEQUENCE OF IfXEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "removed for brevity"
    ::= { ifMIBObjects 1 }
```

jMIBC output

```
ifXTable=1.3.6.1.2.1.31.1.1
ifXTable.TYPE=Table
ifXTable.ACCESS=not-accessible
ifXTable.STATUS=current
ifXTable.DESCRIPTION=removed for brevity
```

Row

An object is declared a `Row` if its `SYNTAX` is declared to be of a type that resolves to an object that has been declared a `SEQUENCE`. In the following example, `ifXEntry` is defined with a `SYNTAX` of `IfXentry` and `IfXEntry` is defined with a `SYNTAX` of `SEQUENCE`:

MIB definition

```
ifXEntry         OBJECT-TYPE
    SYNTAX       IfXEntry
    MAX-ACCESS   not-accessible
```

```
    STATUS      current
    DESCRIPTION "removed for brevity"
    AUGMENTS    { ifEntry }
    ::= { ifXTable 1 }

IfXEntry ::=
    SEQUENCE {
        ifName                  DisplayString,
        ifInMulticastPkts       Counter32,
        ifInBroadcastPkts       Counter32,
        ifOutMulticastPkts      Counter32,
        ifOutBroadcastPkts      Counter32,
        ifHCInOctets            Counter64,
        ifHCInUcastPkts         Counter64,
        ifHCInMulticastPkts     Counter64,
        ifHCInBroadcastPkts     Counter64,
        ifHCOutOctets           Counter64,
        ifHCOutUcastPkts        Counter64,
        ifHCOutMulticastPkts    Counter64,
        ifHCOutBroadcastPkts    Counter64,
        ifLinkUpDownTrapEnable  INTEGER,
        ifHighSpeed             Gauge32,
        ifPromiscuousMode       TruthValue,
        ifConnectorPresent      TruthValue,
        ifAlias                 DisplayString,
        ifCounterDiscontinuityTime TimeStamp
    }
```

jMIBC output

```
ifXEntry=1.3.6.1.2.1.31.1.1.1
ifXEntry.TYPE=Row
ifXEntry.ACCESS=not-accessible
ifXEntry.STATUS=current
ifXEntry.DESCRIPTION=removed for brevity
```

Enum

Enumerations are used to translate names to enumerated values and vice-versa. The following example illustrates an enumeration declared within the object definition:

MIB definition

```
ifAdminStatus OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),
                down(2),
                testing(3)
            }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION "removed for brevity"
    ::= { ifEntry 7 }
```

jMIBC output

```
ifAdminStatus=1.3.6.1.2.1.2.2.1.7
ifAdminStatus.TYPE=Enum
```

```
ifAdminStatus.ABSTRACTTYPE=INTEGER
ifAdminStatus.ACCESS=read-write
ifAdminStatus.STATUS=current
ifAdminStatus.DESCRIPTION=removed for brevity
ifAdminStatus.up=1
ifAdminStatus.1=up
ifAdminStatus.down=2
ifAdminStatus.2=down
ifAdminStatus.testing=3
ifAdminStatus.3=testing
```

Items will also be declared `Enum` if the `SYNTAX` for the item eventually resolves to an enumerated type. The following example illustrates this type of enumeration, where `RowStatus` is defined in the MIB file SNMPv2-TC.my:

MIB definition

```
ifStackStatus  OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION "removed for brevity"
::= { ifStackEntry 3 }

RowStatus ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION  "removed for brevity"
    SYNTAX       INTEGER {
                     active(1),
                     notInService(2),
                     notReady(3),
                     createAndGo(4),
                     createAndWait(5),
                     destroy(6)
                 }
```

jMIBC output

```
ifStackStatus=1.3.6.1.2.1.31.1.2.1.3
ifStackStatus.TYPE=Enum
ifStackStatus.ABSTRACTTYPE=RowStatus
ifStackStatus.ACCESS=read-create
ifStackStatus.STATUS=current
ifStackStatus.DESCRIPTION=removed for brevity
ifStackStatus.active=1
ifStackStatus.1=active
ifStackStatus.notInService=2
ifStackStatus.2=notInService
ifStackStatus.notReady=3
ifStackStatus.3=notReady
ifStackStatus.createAndGo=4
ifStackStatus.4=createAndGo
ifStackStatus.createAndWait=5
ifStackStatus.5=createAndWait
ifStackStatus.destroy=6
ifStackStatus.6=destroy
```

Objects

All other objects are defined with the type specified by the SYNTAX after the SYNTAX has been completely resolved. If an item is declared via a SMI definition, the type will be used directly. In this example, ifTestId has been defined to be an object with a SYNTAX of TestAndIncr where TestAndIncr has been defined as an INTEGER. Therefore, ifTestId will be resolved to an INTEGER.

For example, an object is declared an OID if its SYNTAX is declared to be OBJECT IDENTIFIER. In the following example, ifTestType resolves to an OBJECT IDENTIFIER, as AutonomousType is defined as an OBJECT IDENTIFIER in the MIB file SNMPv2-TC.my:

MIB definition

```
ifTestType       OBJECT-TYPE
    SYNTAX       AutonomousType
    MAX-ACCESS   read-write
    STATUS       deprecated
    DESCRIPTION "removed for brevity"
::= { ifTestEntry 3 }

AutonomousType ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION "removed for brevity"
    SYNTAX       OBJECT IDENTIFIER
```

jMIBC output

```
ifTestType=1.3.6.1.2.1.31.1.3.1.3
ifTestType.TYPE=OID
ifTestType.ABSTRACTTYPE=AutonomousType
ifTestType.ACCESS=read-write
ifTestType.STATUS=deprecated
ifTestType.DESCRIPTION=removed for brevity
```

Error Messages

jMIBC emits all errors to standard error. When an error is encountered, jMIBC will terminate. Action must be taken by the user in order to continue processing. The following are possible errors:

Error: Parsing file <file>

This message indicates that there was an error parsing the input file. The file name where the error is detected is printed on the error line. Additional details of the error will follow and typically include the line number and column where the error was encountered. The following error message indicates that an enumeration begins with a digit:

```
Encountered "1" at line 8036, column 25.
Was expecting one of:
    <LCASEFIRST_IDENT_TKN> ...
    <UCASEFIRST_IDENT_TKN> ...
```

The following example illustrates an error where the RFC file was not edited to remove the extraneous text:

```
Encountered "Working" at line 1, column 9.
Was expecting one of:
    "{" ...
    "DEFINITIONS" ...
    "FORCE-INCLUDE" ...
    "EXCLUDE" ...
```

Error: Unknown syntax of <type> for <name> in <file>.

This message will be displayed if an object has a syntax type that has not been declared: The recommended way to handle this error is to include all dependent MIB files. However, this error can also be suppressed via the `-d` option.

Error: <name> from <file> is undefined.
Please define using the –b option or include all dependent MIB files.

This message will be displayed if a reference to an OID name is encountered that has not been found in the file. The recommended way to correct this problem is to include all the dependent MIB files using the `-i` option in the correct order. The dependent MIB files can be determined by looking at the `IMPORT` definitions and including all the MIB files referenced in the `FROM` clause. However, using the `-d` option will suppress the lack of definition and jMIBC will continue.

Warning Messages

jMIBC emits all warnings to standard error. When a warning is encountered, jMIBC will report the warning and continue. The user should determine if this is an acceptable condition and if not, correct the problem. The following are possible warnings:

```
Warning: Redefinition of <name> in <file>
         Previously found in <file>.
```

This warning indicates that a type, identifier, or OID has been redefined. The last definition found will be used. This warning will only occur if the definitions differ. The recommended way to handle this is to either remove the redefinition or make the definitions identical. However, using the `-d` option will disable the object comparison.

Additional Resources

For more information we suggest:

Format of ASN.1 definitions	ISO/IEC 8824:1998 Specification of Abstract Syntax Notation One (ASN.1) ISO/IEC 8825:1998 Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)
jSNMP Enterprises' collection of MIBs and jMIBC dictionary files	http://www.jsnmp.com/MIBs/
jSNMP Enterprise™ Home Page	http://www.jsnmp.com/products.html
jSNMP Enterprise™ FAQ	http://www.jsnmp.com/docs/jSNMP/jSNMPEnterpriseFAQ.pdf