SysInfo 10 H66 Java Programmer's Guide

COLLABORATORS

	TITLE : SysInfo 10 H66 Java Programmer's Guide	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 15, 2016	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Introduction

1.1 Related Documentation

- SysInfo Release Notes
- SysInfo Installation Guide
- SysInfo Reference Manual
- SysInfo Java Reference Manual

1.2 Target Audience

The target audience for this guide is anybody who wishes to take utilize one of the Application Programming Interfaces (API) provided by MagniComp™'s SysInfo™ software. The API's are intended for in depth use such as embedding SysInfo™ into a third-party application.

1.3 Overview

This document provides information on SysInfo™'s API's. There are multiple ways of programming with SysInfo™ including:

- Writing your own parser to read the field deliminated output from **mcsysinfo --encode report**
- Use the SysInfo™ C API. This option is deprecated due to the complexity placed upon a third-party to implement and sustain.
- Use the SysInfo™ Java API.

This document covers the SysInfo™ Java API which is the recommended API.

Chapter 2

Overview

This chapter will discuss how to use the SysInfo™ Java Application Programming Interface (API). This API provides full access to all SysInfo™ data classes and requires no user-supplied parsing of SysInfo™ data. Each SysInfo™ data class is provided as a discrete Java object. The API functions by executing SysInfo™ and parsing the XML output which is then returned to the caller as Java objects.

The API is fully cross platform compatible on all platforms supported by SysInfo™ including Linux, Mac, Unix, and Windows. A Java application developed with the SysInfo™ Java API can be built once and installed on any SysInfo™ supported platform without the need to recompile on each platform.

The full API is documented in the SysInfo Java Reference Manual.

2.1 Requirements

The SysInfo™ Java API requires the following:

- The SysInfo™ "Full runtime environment" distribution named `mcsysinfo-version-platform` or the "Runtime environment without any user interface" distribution named `mcsysinfo-noui-version-platform` Both are available for download from www.magnicomp.com.
- The SysInfo™ "Software Developer's Kit" (SDK) distribution named `mcsysinfo-sdk-version-platform`. This is available for download from www.magnicomp.com.
- Java SE 1.6 or later available from www.java.com.

2.2 How to Run SysInfo™ Java API

In order to execute and run the SysInfo™ Java API you need the SysInfo Java Software Developer's Kit distribution available from www.magnicomp.com. Inside this kit is the file `mcsysinfo.jar` which contains the SysInfo Java API.

To execute an example which uses the SysInfo™ Java API you run:

```
java -classpath mcsysinfo.jar example.java
```

You can replace `example.java` with your own JAR file or java source file.

Chapter 3

Hello World Example

Here is a simple Java program which will print "Hello World" along with the hostname of the system it is run on.

```
/*
 * Simple "hello world" style SysInfo example.
 */
package example1;

import com.magnicomp.sysinfo.v2.SysInfo;
import com.magnicomp.sysinfo.v2.core.SysInfoCore2;
import com.magnicomp.sysinfo.common.SysInfoException;

public class HelloWorld {
  public static void main(String[] args) {
    SysInfoCore2 sysInfo = null;

    try {
      // Create SysInfo object
      SysInfo si = new SysInfo();
      // Get the SysInfo object
      sysInfo = si.get();
    } catch (SysInfoException e) {
      System.err.printf("SysInfo Exception: %s", e.traceBack());
      System.exit(1);
    }

    System.out.printf("Hello World, my HostName is %s\n",
        sysInfo.getGeneral().getHostName().getValue());
  }
}
```

When this program is executed the output will look like:

```
Hello World, my HostName is superfly.acme.com
```

In the next subsections we will breakdown each component of this code.

3.1 Hello World Imports

The import section of the code contains the following:

```
import com.magnicomp.sysinfo.v2.SysInfo;
import com.magnicomp.sysinfo.v2.core.SysInfoCore2;
import com.magnicomp.sysinfo.common.SysInfoException;
```

The class `com.magnicomp.sysinfo.v2.SysInfo` is the primary class used to get SysInfo data. The SysInfo class data is returned in SysInfoCore2 objects.

It is important to note the `v2` component of the classname packages. The `v2` indicates the class is part of the SysInfo Java API version 2. This version is not the same as the SysInfo product release version. The SysInfo Java API version may change over time when significant changes are needed. The API versioning scheme is in place to allow users of this API to minimize the changes necessary when moving from release to release of the SysInfo product.

The class `com.magnicomp.sysinfo.common.SysInfoException` is SysInfo's primary exception class. It provides traceback methods to allow the API user to determine where a fault occured in SysInfo.

3.2 Hello World Creation

The next part of our example contains these calls:

```
1 SysInfoCore2 sysInfo = null;
2 SysInfo si = new SysInfo();
```

Line 1 creates a `SysInfoCore2` variable named `sysInfo`. The `SysInfoCore2` class is the parent object for each of SysInfo's data class objects. This variable will later be assigned to the parent object containing the data class objects.

Line 2 creates a new `SysInfo` object. Since no arguments are given to `SysInfo()` the API will look for the SysInfo runtime product in the default location of `/opt/sysinfo` on Linux/Unix/Mac and `C:\Program Files\MagniComp\SysInfo` on Windows. If the SysInfo runtime product is not found a `SysInfoException` will be thrown. Later sections will discuss the multiple means of specifing the location of the SysInfo runtime product.

3.3 Hello World Get

The line

```
1 sysInfo = si.get();
```

will execute SysInfo and returned the requested data in a `SysInfoCore2` object identified by `sysInfo`. Since no SysInfo arguments were specified in the `new SysInfo()` call and no call was made to `setCmdOptions()` the default SysInfo data classes `General` and `DevInfo` will be retrieved.

3.4 Hello World Print

The lines

```
1 System.out.printf("Hello World, my HostName is %s\n",
2   sysInfo.getGeneral().getHostName().getValue());
```

will print the "Hello World" string along with the value for the system hostname.

Chapter 4

Show General, Device, NetIf Example

It's time for a more complex example which we will call "Show General, Device, NetIf". In this example we will show how to specify more parameters to tell the API what specific data we want and how to use the resulting data.

This example should be called with a single command line argument of **LocalMethod1**, **LocalMethod2**, or **RemoteMethod1**. Each of these names triggers a different example method of retrieving the same classes of data from either the local system or a remote system.

Here is the full listing

```
/*
 * Example of how to call SysInfo java API to retrieve data.
 * The code in this file can be freely used.
 */
package example1;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.magnicomp.sysinfo.v2.SysInfo;
import com.magnicomp.sysinfo.v2.core.DevInfo;
import com.magnicomp.sysinfo.v2.core.NetIf;
import com.magnicomp.sysinfo.v2.core.NetIfAddr;
import com.magnicomp.sysinfo.v2.core.SysInfoCore2;
import com.magnicomp.sysinfo.v2.core.TypeAddrtype;
import com.magnicomp.sysinfo.v2.core.TypeString;
import com.magnicomp.sysinfo.common.SysInfoException;

public class ShowGeneralDeviceNetIf {
  public static void main(String[] args) {
    // Variable which will contain SysInfo data
    SysInfoCore2    sysInfo = null;
    // Top level directory where SysInfo is installed.
    String        sysInfoDir = "/opt/sysinfo";
    // List of classes to retrieve
    String        classes = "general,device,netif";
    // Hostname of remote system to get data from
    String        remoteHostName = "dune";

    if (args.length <= 0) {
      System.out.printf("First argument must be one of LocalMethod1, LocalMethod2,  ↩
          RemoteMethod1\n");
      System.exit(1);
    }
    String method = args[0];
```

```
    if (method.equalsIgnoreCase("LocalMethod1")) {
      /*
       * Retrieve the SysInfo data specified by classes.
       */
      sysInfo = getLocalMethod1(sysInfoDir, classes);
    } else if (method.equalsIgnoreCase("LocalMethod2")) {
      /*
       * Retrieve the SysInfo data specified by classes.
       * This does the same thing as LocalMethod1.
       */
      sysInfo = getLocalMethod2(sysInfoDir, classes);
    } else if (method.equalsIgnoreCase("RemoteMethod1")) {
      /*
       * Retrieve the SysInfo data specified by classes
       * from host remoteHostName.
       */
      sysInfo = getRemoteMethod1(remoteHostName, classes);
    }

    // Print some of the General class data
    print("GENERAL", sysInfo.getGeneral().getHostName());
    print("GENERAL", sysInfo.getGeneral().getOsName());
    print("GENERAL", sysInfo.getGeneral().getOsVersion());
    print("GENERAL", sysInfo.getGeneral().getOsDistName());
    print("GENERAL", sysInfo.getGeneral().getOsDistVersion());
    print("GENERAL", sysInfo.getGeneral().getSystemModel());
    print("GENERAL", sysInfo.getGeneral().getSystemManufacturerFull());

    /*
     * Print some of the Device data attributes
     * for all the devices found.
     */
    List<DevInfo> devList = sysInfo.getDevInfos();
    Iterator<DevInfo> devIter = devList.iterator();
    while (devIter.hasNext()) {
      DevInfo devInfo = devIter.next();
      System.out.printf("\n");
      print("DEVICE", devInfo.getName());
      print("DEVICE", devInfo.getVendor());
      print("DEVICE", devInfo.getModel());
      print("DEVICE", devInfo.getSerial());
      print("DEVICE", devInfo.getCapacity());
    }

    /*
     * Print some of the NetIf data attributes
     * for all the network interfaces found.
     */
    List<NetIf> ifList = sysInfo.getNetIves();
    Iterator<NetIf> ifIter = ifList.iterator();
    while (ifIter.hasNext()) {
      NetIf nif = ifIter.next();
      System.out.printf("\n");
      print("NETIF", nif.getName());
      print("NETIF", nif.getMACaddr());
      print("NETIF", nif.getSpeed());
      List<NetIfAddr> addrList = nif.getAddresses();
      Iterator<NetIfAddr> addrIter = addrList.iterator();
      while (addrIter.hasNext()) {
        NetIfAddr addr = addrIter.next();
        printAddr("NETIF ADDR", addr.getAddrType());
        print("NETIF ADDR", addr.getHostAddr());
```

```
      print("NETIF ADDR", addr.getNetMask());
      print("NETIF ADDR", addr.getBroadAddr());
    }
  }
}

/**
 * Get SysInfoCore2 data via the SysInfo API from the local system.
 * This is a simple method which uses one API
 * method call to specify the classes to retrieve.
 * @param sysInfoDir
 * @param classes Comma separated list of SysInfo classes
 * @return
 */
private static SysInfoCore2 getLocalMethod1(String sysInfoDir, String classes) {
  SysInfoCore2    sysInfo = null;

  try {
    SysInfo si = new SysInfo(sysInfoDir);
    // Specify what SysInfo data classes we should retrieve
    si.setDataClasses(classes);
    // Get the SysInfo object with data populated
    sysInfo = si.get();
  } catch (SysInfoException e) {
    System.err.printf("SysInfo Exception: %s", e.traceBack());
    System.exit(1);
  }

  return sysInfo;
}

/**
 * Get SysInfoCore2 data via the SysInfo API from the local system.
 *
 * This method passes command line arguments to the SysInfo API
 * which specify what data to retrieve.  It does the same thing
 * as getMethod1 but it shows how any SysInfo CLI option can
 * be specified via the API.
 *
 * @param sysInfoDir
 * @param classes Comma separated list of SysInfo classes
 * @return
 */
 private static SysInfoCore2 getLocalMethod2(String sysInfoDir, String classes) {
 SysInfoCore2    sysInfo = null;

  try {
    /*
     * Create arguments to give SysInfo.  See the mcsysinfocli
     * Reference Manual for details on all valid options.
     */
    List<String> siArgs = new ArrayList<String>();
    siArgs.add("--class");
    // Tell SysInfo what data classes to populate
    siArgs.add(classes);
    // Create SysInfo object
    SysInfo si = new SysInfo(sysInfoDir, siArgs);
    // OPTIONAL: Enable debugging
    si.setDebug(true);
    // Get the SysInfo object with data populated
    sysInfo = si.get();
  } catch (SysInfoException e) {
```

```
      System.err.printf("SysInfo Exception: %s", e.traceBack());
      System.exit(1);
    }

    return sysInfo;
  }

  /**
   * This method retrieves the SysInfo Data Classes specified in
   * classes from a remote system with a hostname of hostName.
   *
   * @param hostName Name of host to retrieve SysInfo data from
   * @param classes Comma separated list of SysInfo classes
   * @return
   */
    private static SysInfoCore2 getRemoteMethod1(String hostName, String classes) {
    SysInfoCore2    sysInfo = null;

    try {
      SysInfo si = new SysInfo(null, hostName);
      // Specify what SysInfo data classes we should retrieve
      si.setDataClasses(classes);
      // Get the SysInfo object with data populated
      sysInfo = si.get();
    } catch (SysInfoException e) {
      System.err.printf("SysInfo Exception: %s", e.traceBack());
      System.exit(1);
    }

    return sysInfo;
  }

  private static void print(String prefix, TypeString info) {
    if (info == null)
      return;
    System.out.printf("%s %s=\"%s\"\n", prefix,
        info.getLabel(), info.getValue());
  }

  private static void printAddr(String prefix, TypeAddrtype info) {
    if (info == null)
      return;
    System.out.printf("%s %s=\"%s\"\n", prefix,
        info.getLabel(), info.getValue());
  }
}
```

When this program is executed the output will look something like:

```
GENERAL Host Name="superfly.acme.com"
GENERAL OS Name="Linux"
GENERAL OS Version="2.6.34.7-56.fc13.x86_64"
GENERAL OS Distribution Name="Fedora"
GENERAL OS Distribution Version="13"
GENERAL System Model="OptiPlex 330"
GENERAL System Manufacturer (Full)="Dell Inc."

DEVICE Name of Device="radeon/1/0/0"
DEVICE Vendor="ATI"
DEVICE Model="RV610 video device [Radeon HD 2400 PRO]"

DEVICE Name of Device="sda"
```

```
DEVICE Vendor="WDC"
DEVICE Model="WD2500AAJS-75VWA0"
DEVICE Serial Number="WD-WCRAY7055940"
DEVICE Capacity="233 GB"

DEVICE Name of Device="sr0"
DEVICE Vendor="HL-DT-ST"
DEVICE Model="DVD+-RW GSA-H73N"
DEVICE Serial Number="07/06/27 7U0"

DEVICE Name of Device="bios0"
DEVICE Vendor="Dell Inc."
DEVICE Model="A06"

DEVICE Name of Device="sysboard0"
DEVICE Vendor="Dell Inc."
DEVICE Model="0KP561"
DEVICE Serial Number="..CN7802813G1G."

DEVICE Name of Device="enclosure0"
DEVICE Vendor="Dell Inc."
DEVICE Model="Desktop"
DEVICE Serial Number="GXBM6G1"

DEVICE Name of Device="cpu0"
DEVICE Vendor="Intel"
DEVICE Model="Core2 Duo Desktop E4600"

DEVICE Name of Device="memory0"
DEVICE Model="Synchronous DDR2"
DEVICE Capacity="2.0 GB"

DEVICE Name of Device="memory1"
DEVICE Model="Synchronous DDR2"
DEVICE Capacity="2.0 GB"

NETIF Name of Interface="lo"
NETIF ADDR Address Type="INTERNET"
NETIF ADDR Host Address="127.0.0.1"
NETIF ADDR Network Mask="255.0.0.0"
NETIF ADDR Broadcast Address="127.255.255.255"
NETIF ADDR Address Type="INET_6"
NETIF ADDR Host Address="::1"
NETIF ADDR Network Mask="128"

NETIF Name of Interface="eth0"
NETIF Current MAC Address="00:1e:c9:38:d2:99"
NETIF Link Speed="100 Mb"
NETIF ADDR Address Type="INTERNET"
NETIF ADDR Host Address="192.5.10.7"
NETIF ADDR Network Mask="255.255.255.0"
NETIF ADDR Broadcast Address="192.5.10.255"
NETIF ADDR Address Type="INET_6"
NETIF ADDR Host Address="fe80::21e:c9ff:fe38:d299"
NETIF ADDR Network Mask="64"
```

4.1 Imports

The imports section of this example contains:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.magnicomp.sysinfo.v2.SysInfo;
import com.magnicomp.sysinfo.v2.core.DevInfo;
import com.magnicomp.sysinfo.v2.core.SysInfoCore2;
import com.magnicomp.sysinfo.v2.core.TypeString;
import com.magnicomp.sysinfo.common.SysInfoException;
```

The first lines import some standard Java classes used in the example. The `com.magnicomp.sysinfo...` classes introduce several new classes not used in the Hello World example.

The `import com.magnicomp.sysinfo.v2.core.DevInfo` class is used to later iterate through a list of DevInfo objects. The `import com.magnicomp.sysinfo.v2.core.TypeString` class is used to allow for some generic handling of the returned data later discussed in the `print` method.

4.2 Setup

The first portion of the `main` class defines a variable to specify the location where the SysInfo runtime product is installed on the local system:

```
String sysInfoDir = "/opt/sysinfo";
```

This value is only used when retrieving SysInfo data from the local system. It is ignored when retrieving SysInfo data from a remote system.

The next variable defines a comma separated list of SysInfo data classes we want to retrieve.

```
String classes = "general,device,netif";
```

The variable declaration:

```
String remoteHostName = "dune";
```

defines the name of the remote system to retrieve SysInfo data from. You should change this to be whatever hostname you want to use in your environment. This value is not used when retrieving data from the local system.

4.3 Get Methods

There are multiple ways of retrieving SysInfo data using the API. The next section of code determines which retrieval method should be used for demonstration purposes:

```
String method = args[0];
if (method.equalsIgnoreCase("LocalMethod1")) {
  /*
   * Retrieve the SysInfo data specified by classes.
   */
  sysInfo = getLocalMethod1(sysInfoDir, classes);
} else if (method.equalsIgnoreCase("LocalMethod2")) {
  /*
   * Retrieve the SysInfo data specified by classes.
   * This does the same thing as LocalMethod1.
   */
  sysInfo = getLocalMethod2(sysInfoDir, classes);
} else if (method.equalsIgnoreCase("RemoteMethod1")) {
  /*
```

```
   * Retrieve the SysInfo data specified by classes
   * from host remoteHostName.
   */
  sysInfo = getRemoteMethod1(remoteHostName, classes);
}
```

The first command line argument specified is used to determine which method is choosen.

4.3.1 Get Local Method 1

The `getLocalMethod1` method retrieves data using the most simple API calls from the local system by running the SysInfo Command Line Interface (CLI). The core code is:

```
SysInfoCore2 sysInfo = null;

SysInfo si = new SysInfo(sysInfoDir);
// Specify what SysInfo data classes we should retrieve
si.setDataClasses(classes);
// Get the SysInfo object with data populated
sysInfo = si.get();

return sysInfo;
```

The call to `new SysInfo(sysInfoDir)` creates a new SysInfo object and tells it to look in the directory specified by `sysInfoDir` for the SysInfo CLI.

The call to `si.setDataClasses(classes)` specifies the list of SysInfo data classes to retrieve.

The call to `si.get()` does the actual retrieval of data.

4.3.2 Get Local Method 2

The `getLocalMethod2` method retrieves data from the local system by running the SysInfo Command Line Interface (CLI). It does the same thing as `getLocalMethod1` but shows how to specify options to the CLI. The core code is:

```
SysInfoCore2 sysInfo = null;

List<String> siArgs = new ArrayList<String>();
siArgs.add("--class");
// Tell SysInfo what data classes to populate
siArgs.add(classes);
// Create SysInfo object
SysInfo si = new SysInfo(sysInfoDir, siArgs);
// OPTIONAL: Enable debugging
//si.setDebug(true);
// Get the SysInfo object with data populated
sysInfo = si.get();

return sysInfo;
```

Any valid SysInfo CLI option can be specified by additional `siArgs.add(...)` calls.

It may be useful to sometimes enable debugging in the SysInfo Java API layer. To do this, you can use the following method:

```
si.setDebug(true);
```

Omitting this method will default to debugging being disabled.

4.3.3 Get Remote Method 1

The `getRemoteMethod1` method retrieves data from a remote system running the SysInfo Agent version 10 or later. The core code is:

```
SysInfoCore2 sysInfo = null;

SysInfo si = new SysInfo(null, hostName);
// Specify what SysInfo data classes we should retrieve
si.setDataClasses(classes);
// Get the SysInfo object with data populated
sysInfo = si.get();

return sysInfo;
```

The call to `SysInfo(null, hostName)` is what tells the SysInfo API to retrieve the information from a host called `hostName`.

4.4 Print General

The next section of code displays the results of the `General` data class data returned in the `sysInfo` object:

```
print("GENERAL", sysInfo.getGeneral().getHostName());
print("GENERAL", sysInfo.getGeneral().getOsName());
print("GENERAL", sysInfo.getGeneral().getOsVersion());
print("GENERAL", sysInfo.getGeneral().getOsDistName());
print("GENERAL", sysInfo.getGeneral().getOsDistVersion());
print("GENERAL", sysInfo.getGeneral().getSystemModel());
print("GENERAL", sysInfo.getGeneral().getSystemManufacturerFull());
```

Each line passes a `TypeString` object to the `print` method. All SysInfo data class's consist of attributes. Each attribute has a defined type. Each defined type is self-describing. That is, it contains not just a value but descriptive data like a description of what the value is and a brief "label" value suitable for display.

We have choosen to report only attributes which are of type `TypeString` in this example.

4.5 Print DevInfo

The final section of the `main` class displays the results of the `DevInfo` data class data returned in the `sysInfo` object:

```
List<DevInfo> devList = sysInfo.getDevInfos();
Iterator<DevInfo> devIter = devList.iterator();
while (devIter.hasNext()) {
  DevInfo devInfo = devIter.next();
  System.out.printf("\n");
  print("DEVICE", devInfo.getName());
  print("DEVICE", devInfo.getVendor());
  print("DEVICE", devInfo.getModel());
  print("DEVICE", devInfo.getSerial());
  print("DEVICE", devInfo.getCapacity());
}
```

Unlike the `General` data class which contains a single set of attributes, the `DevInfo` object is actually a Java `List` of `DevInfo` objects. This is because there are usually multiple devices on a system where-as there is usually only a single hostname as found in the `General` data class.

4.6 Print Method

The `print` method is writen as:

```
private static void print(String prefix, TypeString info) {
  if (info == null)
    return;
  System.out.printf("%s %s=\"%s\"\n", prefix,
      info.getLabel(), info.getValue());
}
```

This method prints the `prefix` string passed to it and then the Label and Value of the `TypeString` object passed to it. The `getLabel()` method returns a brief printable string describing what the object contains. The `getValue()` method returns the actual value as a String for the object. Additional type classes returned by `SysInfoCore2` are described in the SysInfo Java Reference Manual.