Guide Line Definition for JavaPOS for Windows Operating Systems

- Extensions to the Standard

Version	1.3
Date	Apr 18th 2016
Author	Peter Düllings, Wincor Nixdorf International GmbH
Contributions	…
Pages	30
Audience	- Application providers - Independent hardware vendors of peripheral devices - POS integrators

Table of Contents

1 Introduction

This document describes the requirements for JavaPOS Middleware installers for Windows operating systems.

Currently the JavaPOS standard provides in a good way the definition of the application programmer's interface to allow a hardware-independent access of peripheral devices [1].

However, during working with the standard some issues have been experienced which are not yet standardized or there are no defined Guide Lines available. The intention of this document is to provide a Guide Line which is a recommendation for application providers (ISV) as well as peripheral vendors (IHV). The document targets the following items in handling JavaPOS modules on Windows based operating[1] systems:

- Packaging und delivery of JavaPOS drivers
- Configuring of JavaPOS modules to the desired hardware
- Handling of additional functionality concerning firmware and device settings
- Handling of additional inventory and statistic data by JavaPOS drivers

1.1 Overview

As of today these Windows OS run on Windows Systems:

- Windows 7 Professional (x86/x64)
- POSReady 7 (x86/x64)
- Windows 8.1 Professional (x86/x64)
- Windows Embedded 8.1 Industry (x86/x64)
-

Beside the Windows versions listed above the latest Windows OS version 10 needs to be considered.

1.2 Audience

Providers of POS systems and peripherals (Independent Hardware Vendors[2]) shall deliver JavaPOS packages following this Guide Line, which are compatible to the Windows 7 Professional, POSReady 7, Windows 8.1 Professional, Windows Embedded 8.1 Industry, Windows 10 and Windows 10 successor versions and do not break the system integrity of the OS.

Application provider (Independent Software Vendor) shall use the JavaPOS modules in the described manner following this Guide Line.

[1] e.g. POS systems, cash systems, automated checkout systems, kiosk systems
[2] i.e. OEMs and ODMs

Application and POS user (customer) shall use the driver and the on top layered application in the hereby described manner.

1.3 Bibliography

[1] The Unified POS specification Version 1.13 from July 15th,2009
Downloadable from http://nrf-arts.org/content/unifiedpos

1.4 Abbreviations/Glossary

API	Application Programming Interface
ISV	Independent Software Vendor
IHV	Independent Hardware Vendor
ODM	Original Design Manufacturer
OEM	Original Equipment Manufacturer
POS	Point Of Service
JCL	JavaPOS Config Loader see also: http://sourceforge.net/projects/jposloader/
OS	Operating System

1.5 History

Version	Date	Author	Remarks
V1.0	2014-10-14	P. Düllings	First version
V1.1	2016-03-03	P. Düllings	-Windows PATH adjusted - javapos.config.properties AND javapos.config.xml - chapter 3 added: installation of additional features in a separate installer - msi installer preferred - log files to be written in UTF-8 mode
V1.2	2016-03-07	P. Düllings	Minor spelling errors corrected.
V1.3	2016-04-18	P. Düllings	- Some textual replacements, clarifications. - Serviceability data in inventory and monitoring (log) files updated and more precise described. - configurator allowing properties and XML as input (javapos.config.properties and javapos.config.xml)

2 Packaging and delivery of JavaPOS modules

This chapter describes the manner of how to package a JavaPOS module under Windows Operating Systems. There are several general Guide Line issues defined and examples shown.

2.1 Deliverables as MSI or as Self-Extractor

1. The Installer has to be provided either as MSI Installer (preferred) or as self-extracting EXE program.
2. After installation in the common Windows data location an entry for this software is added. Typically this is under “Control Panel” / “Add & Remove Software”. Also an uninstallation of the SW is to be provided.
3. The installer has to ensure that existing customized config files[3] are not deleted or modified.
4. The installer has an option to create a log file containing more detailed information about the individual actions during installation and uninstallation.
5. The installation can be done interactively allowing changing at least the installation directory and the *Program Group* name for the *Start Menu Folder*. ALL additional parameters may be changed interactively.
6. The GUI mode installer has an option to create at the end of the installation an automatic installation script file (ASCII / XML format) containing all selected installation parameters by the user. The automatic installation script may be used as input file for a silent installation. The major purpose of the automatic installation script file is to minimize the customization efforts for the user.
7. The installation can also be performed silently and unattended.
8. Silently means, no GUI and error messages do pop up during installation.
   Unattended means, no additional user interaction is needed to perform the installation.
   That is, all necessary parameters such as
   - installation directory
   - installation options (e.g. packages)
   - Program Group name for the Start Menu Folder
   - deleting log files
   - remove running system service provided by the JavaPOS installer package[4]
   - others

   can be passed via calling options to the installer or referencing to the automatic installation script file (see point 4.). No interaction with the user shall be required. The

[3] list of customized config files see chapter “Wincor Nixdorf JavaPOS configurator”. Customized config files are files containing customer specific settings.
[4] this option is required in case of an update installation to a new installation path

installer writes more detailed information into a log file which can also be specified via calling options.
The major purpose is to allow a remote, automatically installation or update of existing installation without any user interaction at the system.

9. The default installation path is
   %PROGRAMFILES%\javapos*<companyname-with-extensions>*
   <companyname-with-extensions> shall not contain white space characters.

   The installer has an option which allows changing the default installation path.
   For example:
   XYZ_JavaPOS_1_13_001>.exe /DIR=C;*<mydirectory>*
   In case of attended installations, the path
   %PROGRAMFILES%\javapos*<companyname-with-extensions>*
   is presented as default and may be modified interactively.
10. Windows 32 bit and Windows 64 bit:
    An installer has to be provided for Windows 32bit.
    Another separate installer has to be provided for Windows 64bit.
    Installing the 32 bit installer under Windows 32 bit shall install the software to
    %PROGRAMFILES%\javapos*<companyname-with-extensions>*
    Installing the 64 bit installer under Windows 64 bit shall install the software to
    %PROGRAMFILES%\javapos*<companyname-with-extensions>*
    Installing the 32 bit installer under Windows 64 bit shall install the software to
    %PROGRAMFILES(x86)%\javapos*<companyname-with-extensions>*
    (This may be the case when using a 32 bit JavaVM under Windows 64 bit)

11. In the Program Group of the Start Menu Folder additional icons are created during installation:
    - icon for uninstallation
    - icon for additional readme and PDF documentation
    - icons for additional test tools
12. System services provided with a JavaPOS installer package do not start automatically after the installation. Default startup type should be “automatic”, i.e. system service(s) will be started after reboot automatically.
    It should be always possible to disable automatic start of system service by a calling option of the installer.
    System services names and the “Display name” of system services are vendor specific and shall follow the naming convention:
    [*<companyname><service name>*]
    <companyname> shall not contain whitespaces and special characters. Typically only those characters as allowed under Windows for group names.

The “Description” of the system service should contain a short description of the system service itself.

13. Running system services provided with a JavaPOS installer package should be stopped and removed automatically in case of an update installation.
14. The JavaPOS SW must be installable and removable unattended, that is, no user interaction is needed. The major purpose is to allow a remote, automatically installation, update or removal of SW modules without any user interaction at the system.
15. It can be assumed that a package is installed under a user with administrator rights.
16. Adaptations dependent on connected peripheral device have to be performed in a system service scripts and to be performed during booting of the system – not during installation. If the JavaPOS middleware package needs to use a background service it needs to be installed as Windows operating system service following the standard Guide Lines for Windows Services.
    see also: http://en.wikipedia.org/wiki/Windows_service
17. Additional 3rd party components such as a communication API for RS232 interfaces or for USB devices have to be packaged in a separate installer package with respect to the defined license.

Note: recommended is the easy-to-use Freeware Tool **Inno Setup**:
http://www.jrsoftware.org/isinfo.php

2.2 Content of a JavaPOS Packages

Packages shall contain at minimum:

- One or more JAR files containing the Java dependent code
- If required for JavaPOS, one or more binary files containing additional features not available in the Java program code
- JavaPOS configuration files as XML files readable by the JCL.
- JavaPOS example configuration files
- Documentation describing each configuration parameter, their meanings and their possible range of settings. The documentation shall be provided either in HTML, PDF or TXT format.

Additional documentation about hardware may be desirable. However, usually such information is distributed by the vendor independently from the JavaPOS package.

2.3 Directory structure of JavaPOS installer packages

For Windows it is usually recommended to install software packages under directory

%PROGRAMFILES%\<*companyname*>

directory. The structure below is not really defined.

To let OEM JavaPOS packages work together with Window Nixdorf JavaPOS without further manual configuration, there is a determined directory structure required. Usually Wincor Nixdorf JavaPOS has the JavaPOS Service Control Layer and the JavaPOS Config Loader included. Therefore it is not necessarily required to have this additionally in an OEM JavaPOS Package of a peripheral device.

The following file structure shall be prepared after installation of the JavaPOS middleware of the peripheral device:

- %PDIR% is the selected installation directory of the JavaPOS package, e.g.
  %PROGRAMFILES%\javapos\<companyname-with-extensions>
- %DATADIR% is the predefined selected data directory, e.g.
  %PROGRAMDATA%\javapos\<companyname-with-extensions>

Documentation	%PDIR%\doc
JavaPOS JAR Files	%PDIR%\lib
Binary files (Dynamic link libraries DLL)	%PDIR%\bin
Binary files (executable)	%PDIR%\bin
Batch script , cmd scripts and other scripts	%PDIR%\bin
JavaPOS XML Configuration files (considered as read-only)	%PDIR%\xml
JavaPOS option files (used to JavaVM (considered as read-only, see section 4.1)	%PDIR%\joption
Configuration files (e.g. property files *.properties, *.ini files,*.conf etc.)	%PDIR%\config
Logging files	%DATADIR%\log
Installation helper files /helper scripts etc	%PDIR%\inst

2.4 User handling for JavaPOS drivers and Security aspects

For security reason Retail applications do not run under Administrator permission. Usually Retail application have an own Windows user created ("POS user") and especially defined for the Retail application.

Therefore, JavaPOS drivers cannot assume that they run under Administrator permissions or an own user assigned by the JavaPOS package.

Since under Windows operating system users can be member of one or more groups it allows to handle this security requirement by defining Guide Lines handling groups.

A JavaPOS package provider shall deliver the package in that way that - if the Retail application assigned Windows user is member of an own, vendor specific group - the Retail application can use the JavaPOS also as non-administrator.

The name of the group is vendor-specific and

- shall follow the naming convention **<*companyname*>dev**
  <companyname> shall not contain whitespaces and special characters. Typically only those characters as allowed under Windows for group names.
- is created by the JavaPOS package during installation, if not exists. If multiple JavaPOS packages have to be installed the first package creates the group
- remains also after uninstalling the JavaPOS package.
  (The reason is that otherwise the assignment of the POS user to the group may be lost when uninstalling all JavaPOS packages and reinstall the packages again.)

2.5 Architecture Guide Lines of the driver stack JavaPOS

1. To allow an easier modular packaging the JavaPOS deliverables should not contain possibly required Windows operating system drivers (those are typically files with extension *.sys).
   Therefore it is recommended to use for:
   - USB printers the standard USBPRINT.SYS driver which is part of the operating system.
   - USB devices which are designed as USB HID devices the standard USBHID.SYS driver which is part of the operating system.
   - for RS232 (serial devices) the standard operating system serial driver.
   - for USB devices which are not designed as USB HID devices possibly own OS system drivers.

   However, those drivers need to be delivered by the OEM / ODM as WHQL certified drivers for Windows 7 Professional, POSReady 7, Windows 8.1 Professional, Windows Embedded 8.1 Industry, Windows 10 and not as part of the JavaPOS package.

2. During installation process the installer must not assume that hardware is connected to the system. Instead, in a later phase when rebooting the system a connected peripheral hardware can be assumed

3 Packaging and Delivery of additional software modules

When providing additional features as described later in chapter "5 Features of JavaPOS packages" as separate Java-written or native modules and not as part of the JavaPOS device service (that is, provision of inventory , statistical and monitoring data, firmware download and parameter set download) those may be provided as a separate installer.

The guidelines are the same as described in chapter "2 Packaging and delivery of JavaPOS modules" - with the following exceptions:

- The default installation path shall be:
  %PROGRAMFILES%\<companyname-with-extensions>
- Under this installation path the same sub directory structure as described in chapter 2.3 shall be used.

4 Configuring JavaPOS

Configuration of JavaPOS has been done in the past in very different ways dependent on methods defined by the different vendors.
This creates a lot of efforts on integrator side, on peripheral hardware vendor side and also on customer side. Settings for XML configuration files, rules for combining the important so-called “CLASSPATH” and also the so-called BINARY-PATH created a lot of headache.
The approach herein introduced shall reduce the effort significantly.
This chapter is more or less not important for the JavaPOS provider but for the POS application provider. However, it helps also JavaPOS providers to get a better understanding for the requirements.

4.1 Wincor Nixdorf JavaPOS configurator

The JavaPOS configurator is a small Java program which collects all files and paths, parses all the XML files and creates the **summarized jpos.xml**[5] and **setenv.bat** file.

The source of the configurator is provided by Wincor Nixdorf to allow also other companies this type of configuration to get a wider acceptance in the Retail market for peripheral vendors and application vendors.

This chapter describes the recommended mechanism to consolidate the JavaPOS drivers of connected peripheral devices and their corresponding installation path, so that the Java Retail application does not need to include JavaPOS drivers within their installation package AND to ensure correct JavaPOS installation path settings.

The goal is to separate JavaPOS drivers and their delivered XML files from the POS application and to increase supportability and the ease of life cycle management of a JavaPOS enabled Windows system.

Currently, some Java Retail applications are using static classpath.xml and jpos.xml files or are using their own mechanism for defining and generating a jpos.xml file which defines the “JavaPOS Open Names” to access the POS peripherals through the JavaPOS API.

The new concept offers a mechanism which creates on the fly matching CLASSPATH, BINARY-PATH and a jpos.xml file containing all default JavaPOS XML entries. JavaPOS providers can add customized specific XML entries, properties of XML entries which are overtaken automatically by the Configurator and enabled automatically for the application.

Prerequisite is, of course, that all JavaPOS modules follow the here described Guide Lines. The application can use the output of this process and is independent of

- adding, removing or renaming of JAR files from JavaPOS providers
- adding, removing or renaming of XML files and their entries from JavaPOS providers

[5] The jpos.xml output file describes the whole JavaPOS configuration of the system.

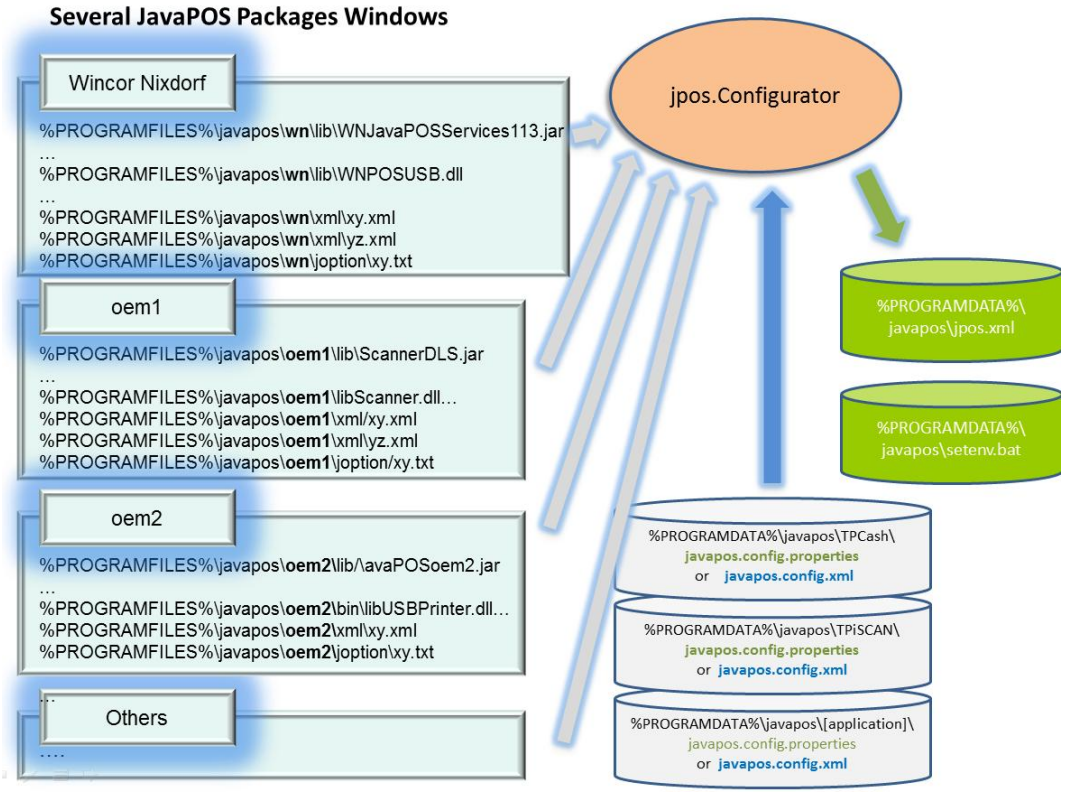
- adding, removing or renaming of native components such as dynamic link libraries shared objects or other executables from JavaPOS providers
- setting JavaPOS environment variables (so-called Java system properties)

The original XML file(s) installed by the JavaPOS package shall be considered as read-only for the application provider. Customized specific device settings may be set preferable in these config files by the application providers.

Application providers provide their configuration in one or more files named as **javapos.config.properties** or **javapos.config.xml** which is the glue code between JavaPOS and application. They are located at the directory path:

%PROGRAMMDATA%\javapos\<*companyname-with-extensions*>

To understand the process the following picture might help help ("oem1","oem2" and "wn" stands for <companyname-with-extension> of different JavaPOS delivering companies):



The customized config file(s) **javapos.config.properties** respectively **javapos.config.xml** are in a subdirectory with the name of the application below of
%PROGRAMDATA%\javapos

The configurator collects the existing customized config files and updates the resulting output file **jpos.xml** with the specific JPOSEntries in the customized config files. If on a system multiple applications use the JavaPOS layer each application can handle its own customized config file(s) in its own directory. If on a system multiple application use the

JavaPOS layer each application can handle its own javapos.config.* file. The jpos.xml output file describes the whole JavaPOS configuration of the system.

The output of the JavaPOS configurator is located in the directory

%PROGRAMMDATA%\javapos

and contains two files as follows:

- **"jpos.xml"** which contains a copy of the original XML JposEntries as provided by the different parties on the left side. Only those JposEntries are copied which are named in the **javapos.config.properties** respectively **javapos.config.xml** file(s)

- **"sentenv.bat"** batch script which defines some environment variables:
  - JAVAPOS_PATH:
    The variable contains all native paths which are needed to access native executables or batch scripts by some JavaPOS modules. This variable shall be used by the application starter BATCH script to expand the PATH variable.
  - JAVAPOS_LD_PATH:
    The variable contains all native paths which are needed to access the native dynamic link libraries by the JavaPOS modules. This variable shall be used by the application starter BATCH script to expand the PATH variable.
  - JAVAPOS_CLASSPATH:
    This variable contains all the collected JAR files and directories needed to have the several JavaPOS packages included. This variable shall be used by the application starter BATCH script to expand the CLASSPATH variable or the content may be passed to the JavaVM with the option –classpath.
  - JAVAPOS_OPTIONS:
    The variable contains additional options required by the device services and need to be transferred to the JavaVM. Application starter BATCH script shall use the content to pass immediately as first argument to the JavaVM:

    (If JavaPOS providers needs to have JavaVM option (usually transferred with the –D option), they can provide the options in a textfile located in the directory /opt/[cname]/javapos/joption *(see section 2.3). Such textfile may have as content "-DWNJavaPOS.debug=true". Each line must have one option, lines starting with "#" are comments.)*

  - JAVAPOS_JAVA_HOME:
    This variable identifies the absolute path of the installed JavaVM as used by the JavaPOS configurator. For example the Java runtime engine can be called by using the program $JAVAPOS_JAVA_HOME/bin/java.
    The home path is taken directly from the Java runtime system property *"java.home"*.
    Of course, different applications running simultaneously may use different Java runtime engines. The value in this variable is related to the JavaVM as used for running the configurator.

  - JAVAPOS_JAVA_VERSION:
    This variable identifies the current version of the JavaVM runtime engine as

used by the JavaPOS configurator.
The version is taken directly from the Java runtime system property *"java.version"*.
Of course, different applications running simultaneously may use different Java runtime engines. The value in this variable is related to the JavaVM as used for running the configurator.

Application providers have only to include the script "setenv.bat" with the BATCH command "call" (it calls another BATCH file and returns). Then the automatically created jpos.xml and the necessary CLASSPATH and binary PATH can be used by the application.
Example for an application starter batch script:

```
#some other lines of script
Call %PROGRAMDATA%\javapos\setenv.bat
set PATH=%JAVAPOS_PATH%;%PATH%
set PATH=%LD_LIBRARY_PATH%;%PATH%

java %JAVAPOS_OPTIONS% <additional options> -classpath \
%JAVAPOS_CLASSPATH% com.applprovider.MyPOSApplication
```

The configurator is to be executed always in case

- the application provider has updated the customized config files (javapos.config.properties or javapos.config.xml)
- A JavaPOS update of a package has been installed.

The customized config files have to be updated only in case

- An open name of a JposEntry in delivered XML files has been changed in a JavaPOS update
- An existing property is modified in the customized config file, the JposEntry default value of the original JavaPOS config file has been changed or a NEW JposEntry has been added to the original JavaPOS config file and needs to be modified for the customer.

The right place for calling the configurator is typically during each reboot of the POS system.

4.2 The customized config property file javapos.config.properties

This file contains the application specific device settings. The file is a so-called property file. That is, the file may contain comments (lines starting with the comment symbol "#") and additional lines containing <*name*> < *value*> pairs.

Mainly the file contains two types of information

1. A list of used JposEntries in the desired configuration. The JposEntries are specified by their names (JposEntry property "logicalDeviceName").
   Additionally, the entry can be mapped to a name which is more comfortable to the application or is used by default from the application.
   Syntax:
   jpos.names=name1,<name2>,<name3>,<name4>,…

jpos.name.<name1> =<originalname1>
jpos.name.<name2> =<originalname2>
jpos.name.<name3> =<originalname3>
jpos.name.<name4> =<originalname4>
…

Example:

```
# this is a comment
jpos.names=printer1,scanner1,scale1,apats-1,ic15e-1
jpos.name.printer1=WN_TH250_COM
jpos.name.scanner1=DLS-Gryphon-GD4135-USB-Scanner
jpos.name.scale1=WN_SCALE_SOFTWARE_WINDOWS
jpos.name.apats-1=CashChanger_C6020
jpos.name.ic15e-1=CashChanger_iCash15e
```

…
The application uses the open names (logical device names) "printer1", "scanner1","scale1,"apats-1",ic15e-1" whereas the names "WN_ND77_COM", "*DLS-Gryphon-GD4135-USB-Scanner", "WN_SCALE_SOFTWARE_WINDOWS", "CashChanger_C6020", "CashChanger_iCash15e"* are the original names as defined in the vendor delivered original XML files.

2. Definition of values for properties for a JposEntry which deviate from the original settings as defined in the vendor delivered original XML files.
   This is used for example if a RS232 device is connected at another COM port as defined in the original XML file. Or, if the printer is used in another country and a XML setting defining the country code is to be adapted.
   Syntax:
   jposentry.<original-openname>.<propertyname>=<newvalue>

   Example:

```
# this is a comment
jposentry.DLS-Gryphon-GD4135-USB-Scanner.beepDuration=2

#example: due to higher resolution the scale window needs to be centered
jposentry.WN_SCALE_SOFTWARE_WINDOWS.xPositionVD=400
jposentry.WN_SCALE_SOFTWARE_WINDOWS.yPositionVD=400

# we want to use the scale in UK
jposentry.WN_SCALE_SOFTWARE_WINDOWS.weightUnit=POUND

# the printer is connected at COM1 instead of COM2
jpos.entry.WN_TH250_COM.portName=COM1

# the cashchanger is connected at COM4 instead of COM1
jpos.entry.CashChanger_iCash15e.portName=COM4
```

4.3 The customized config XML file javapos.config.xml

Another possibility of specifying the JavaPOS configuration can be done in XML syntax. The previous example would be represented in XML syntax as follows (**javapos.config.xml**):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JposEntries PUBLIC "-//JavaPOS//DTD//EN" "jpos/res/jcl.dtd">
<JposEntries>
        <JposEntry logicalName="printer1">
                <prop name="originalOpenname" value="WN_TH250_COM" type="String"/>
                <prop name="portName" value="COM1" type="String"/>
        </JposEntry>
        <JposEntry logicalName="scanner1">
                <prop name="originalOpenname" value="DLS-Gryphon-GD4135-USB-Scanner" type="String"/>
                <prop name="beepDuration" value="2" type="String"/>
        </JposEntry>
        <JposEntry logicalName="scale1">
                <prop name="originalOpenname" value="WN_SCALE_SOFTWARE_WINDOWS" type="String"/>
                <prop name="xPositionVD" value="400" type="String"/>
                <prop name="yPositionVD" value="400" type="String"/>
                <prop name="weightUnit" value="POUND" type="String"/>
        </JposEntry>
        <JposEntry logicalName="ic15e-1">
                <prop name="originalOpenname" value="CashChanger_iCash15e" type="String"/>
                <prop name="port" type="String" value="COM4" />
        </JposEntry>
        <JposEntry logicalName="apats-1">
                <prop name="originalOpenname" value="CashChanger_C6020" type="String"/>
        </JposEntry>
</JposEntries>
```

4.4 Automatic configuration

In a later step an extension to this configurator could be a type of automatic configuration during boot of the system. It is conceivable to allow a process parsing operating system resources and checking which peripherals are connected and generating as output a customized config input property of XML file e.g. **javapos.config.properties** or **javapos.config.xml** file. A later service script may then run the configuration and creates the proper xml files for the JavaPOS layer.

5 Features of JavaPOS packages

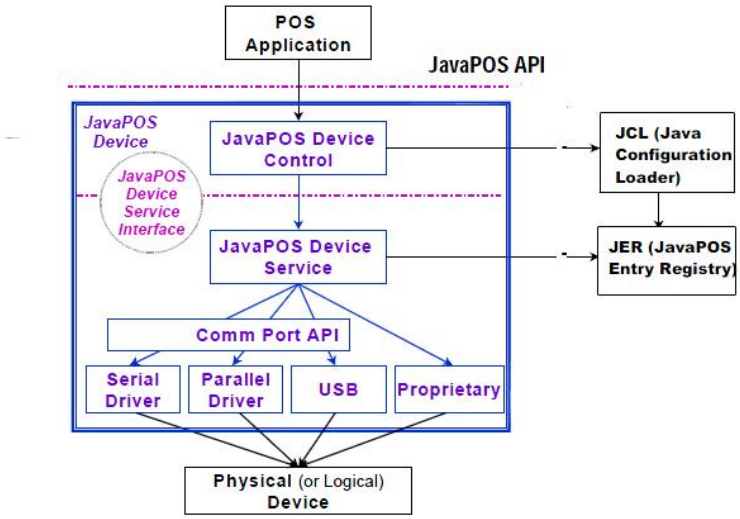
The main target of JavaPOS modules is to offer peripheral functionality in a harmonized way to POS providers. The idea is, for example, if printer hardware is changed the application has nothing to do than only exchanging the open name of the JavaPOS layer.

Due to the fact that in the JavaPOS API some functionalities are defined as optional controlled by so-called capabilities this target cannot be reached completely.

5.1 Programming API for Java application

The API follows the document [1] which describes UnifiedPOS and therefore the API JavaPOS for Java Technology in version 1.13.

The whole software stack has to follow the architecture as shown in the picture.



Since the JavaPOS Device Control layer is typically hardware independent the POS integrator has to deliver the Control layer and may extend functionality such as trace and diagnostic features. However, the JavaPOS API (and JavaPOS Device Service Interface) must not be changed. If new device categories not yet defined by the standard are needed they have to be clearly marked as extension. However, those extended categories shall follow the same architecture.
The JavaPOS Device Service layer and layers below have to be provided by peripheral vendor.

More detailed information about concept, architecture and API are described in the document[1].

5.2 Providing of Inventory and Statistical Information of the device

JavaPOS packages shall deliver inventory and statistical information of the connected device. This can be done either

a) separately

or

b) simultaneously

when the application runs with JavaPOS. The latter method is the better one.

5.2.1 Providing data separately

This can be done by a process running when the application has no access to the device.

There are two possibilities for the JavaPOS provider:

a) Providing a separate Java module collecting the data from the device and returning them as an XML string (see section 5.2.2)
b) Providing the method ***retrieveStatistics()*** in the JavaPOS module following the description in [1].

In both cases the POS integrator needs to perform a process of using the module and writing the retrieved date to an output file. The location of the output file shall be configurable. This process will be started when the retail application is down e.g. in a defined time frame during night or after the “day end” procedure when the POS application typically performs a systems reboot.

5.2.2 Providing data when the application runs

Even better is the way of providing these data when running the application since then provided statistical information are more up-to-date. Also the application has not to be shut down for collecting inventory and statistical data e.g. during night.

For this case the JavaPOS has to provide the data via a separate interface. This interface can be used simultaneously to the POS application by another service process.

The JavaPOS module shall perform a way of writing into output files by considering the defined requirements.

Providing output files

The information shall be provided by a separate ASCII output file or a separate XML output file with some requirements:

- The location and the name of the log file must be configurable (e.g. by a property file)

- A mechanism for making sure that the output file for inventory and statistical information does not increase infinite during long usage of the JavaPOS module.
- If an ASCII file is offered, the file shall contain per line one field following the format <fieldname>=<value>.
  A description shall be provided describing the meaning of the fields.
- If an XML file is offered the format shall follow the format as described for the JavaPOS ***retrieveStatistics()*** method (please see [1]).
  A description shall be provided describing the meaning of the fields.

5.2.3 Type of information

The following inventory information should be provided by peripheral hardware vendors:

#	Field meaning	Recommended fieldname[6]
1	ConfigurationEntry: Openname of the device driver as used by the application	CONF_OPENNAME
2	ConfigurationEntry: Java Service Class as used in the Device Service	CONF_SERVICECLASS
3	ConfigurationEntry: JavaPOS Device category as defined by the standard (e.g. JavaPOS_POSPrinter, JavaPOS_Scanner, JavaPOS_LineDisplay, etc the categories as defined in [1]	CONF_CATEGORY
4	ConfigurationEntry: Description of the JavaPOS Device Service as defined by the standard[1]	CONF_DESCRIPTION
5	ConfigurationEntry: Version of the JavaPOS Device Service, usually such as 1013XXX	CONF_DRIVER_VERSION
6	ConfigurationEntry: Description of the Device Service, usually delivered by the property getDeviceServiceDescription() of the JavaPOS device service	CONF_DRIVER_DESCRIPTION
7	ConfigurationEntry: Connection information such as serial connection settings or USB connection settings. Examples: “connected to RS232,/dev/ttyS1,9600,e,8,1,r” “connected to USB vid:pid=VVVV:PPPP (vendor- and product id in hexadecimal)	CONF_CONNECTION
8	Firmware Date in Format YYYY-MM-DD If multiple firmware dates are available they have to be separated by a comma in form <componentname1>=<date1>,	DEVICE_FIRMWARE_DATE

[6] Naming conventions for fields:
Configuration entries start with CONF_, statistical entries from the device start with STAT_, statistical counters start with STAT_C_, statistical time counters start with STAT_T_, statistical DATE start with STAT_D_ in format YYYY:MM:DD [HH:mm] or YYYY:CCC

	<componentname2>=<date2>,...	
9	Firmware Version in device individual manner may also contain different version numbers of SUB modules such as BOOT ID, FLASH ID, etc. If multiple firmware versions are available they have to be separated by a comma in form <componentname1>=<version1>, <componentname2>=<version2>,...	DEVICE_FIRMWARE_VERSION
10	Serial number(s) of device or device and subcomponents.(for USB devices those are usually offered by a standard USB descriptor) this is read via an String descriptor; If multiple serial numbers are available they have to be separated by a comma in form <componentname1>=<no1>, <componentname2>=<no2>,...	DEVICE_SERIAL_NUMBER
11	Date of Device Production in Format YYYY-MM-DD or YYYY-CCC (ccc is the calendar week). If multiple production dates are available they have to be separated by a comma in form <componentname1>=<date1>, <componentname2>=<date2>,...	DEVICE_PRODUCTION_DATE

Additional inventory fields may be for example:

a) Scanner configuration (e.g. enabled barcodes , enabled beep)
b) Scales (e.g. max weight, min weight, unit (e.g. kg or lb)

The following statistical information should be provided by peripheral hardware vendors:

a) Number of total operating hours, number of operating hours since last power cycle (recommended field names STAT_T_POWER_ON, STAT_T_POWER_ON_TOT)
b) Number of total error situations of the device, number of error situations of the device since last power cycle.

The following statistical information should be provided by peripheral hardware vendors dependent on the device category, examples:

a) For all devices printers:
number of total lines printed, number of lines printed since last power cycle, number of paper cuts, number of paper cuts since last power cycle (recommended fieldnames: STAT_C_LF_TOT, STAT_C_LF, STAT_C_CUT_TOT, STAT_C_CUT)

b) For scanners:
number of good reads, number of bad reads
(recommended fieldnames: STAT_C_READ_GOODREAD, STAT_C_BADREAD)

c) For scales:
number of good weighing's, number of under load conditions, number of overload weighing's

5.3 Providing of Monitoring Information of the device

5.3.1 Providing LOG files

Monitoring information shall be offered in a separate LOG file which is written by the JavaPOS module when the application is running.

Requirements to the LOG file:

- The location, the name of LOG file(s), LOG levels of ALL log files, LOG file settings must be configurable in a single property file (ASCII format).
- LOG file name shall follow the naming convention
  [<YYYY-MM-DD_HH-MM_filename>]
  <filename> shall not contain whitespaces and special characters.
- A mechanism for making sure that the log file does not increase infinite has to be performed by the provider.
- A mechanism should ensure that LOG files are created on daily basis. In case a LOG file exceeds the MAX file size limit a new LOG file should be created automatically.
- Every new created LOG file should start with a unique "header"[7] providing information like:
  PRODUCT-NAME
  VERSION-NR
  BUILD-ID
  USED DEVICES
  (see chapter."4.2 The customized config property file javapos.config.properties")
- Each monitoring event creates exactly one line in the LOG file. If a text contains a line feed it has to appear in the file as %0A (Note: a real "%" character has to appear as "%25"). *(This is for further tools analyzing the LOG file and parsing each line in a proper way.)*
  The format of a LOG file line has to follow the definition in section 6.1.
- A log entry is to be provided when an exceptional situation of the device occurs and also when the situation is again changed to OK / OPERATIONAL.

[7] target group for these header information is typically 1st and 2nd Level Service help desk

- Initially when JavaPOS starts during the setDeviceEnabled(true) a set OK/OPERATIONAL monitoring entries have to be performed to give information about the different initial states.
  Additionally, the JavaPOS methods **open(),close(), claim(), release()** and ***setDeviceEnabled(true)***, ***setDeviceEnabled(false)*** shall create a monitoring entry of type HINT.
- The log file shall be written in data format UTF-8.

Each LOG entry has the following form and is exactly one line – means it is terminated by the new line character:

```
dateTime|severity|computerName|componentName|instanceName|
        errorCode|errorCodeExtended|errortext [|additionalText]
```

The fields are separated by the „|“ (Pipe) character.

An example of such a LOG file:

```
25.10.2005 11:50:52.234|1|kasse103|JavaPOS.POSPrinter| WN_TH230_USB|0|11| Printer cover has been opened
25.10.2005 11:58:12.001|3|kasse103|JavaPOS.POSPrinter| WN_TH230_USB|108|0| error: printer is offline and does not respond!
```

5.3.2 Type of information

The following monitoring information (the event in case a state changing occurs) should be provided by peripheral hardware vendors:

a) Device has been disconnected or is reconnected again
b) Device has been changed to an inoperable state or changed again back to an operable state.

The following monitoring information should be provided by peripheral hardware vendors (this is a list of examples, of course, even more information of the peripheral device as provided by the hardware is recommended):

a) For POSPrinters:
   THE PAPER IS EMPTY, NEAR EMPTY OR OK
   THE PRINTERS COVER HAS BEEN OPENED OR CLOSED
   THE TONER IS EMPTY, NEAR EMPTY OR AGAIN FULL
   THE INK CASSETTE IS EMPTY, NEAR EMPTY OR OK

b) CashChangers:
   FOREIGN_MATERIAL_DETECTED
   SHUTTER_ERROR
   RETRACT_NOT_POSSIBLE
   CASHIN_CAPACITY_REACHED
   SEPARATION_NOT_POSSIBLE
   CASHOUT_INSEPARABLE_ITEMS_ROLLER1

c) Scanners:

d) Scales:
   TARE WEIGHT HAS BEEN CHANGED
   ZERO SCALE HAS BEEN PERFORMED

5.4 Firmware and configuration data download

POS peripheral providers shall provide in their JavaPOS packages a type of firmware download which can be used standalone and **remotely without interactions**.

The download process can be started by operating system based services (e.g. scheduled task) and in defined time frames, for example, in the night.

The application provider makes sure that before starting the firmware download process any access to the peripheral devices by the application is omitted.

There are several possibilities:

a) Either by implementing ***updateFirmware()*** of the JavaPOS Module.

b) Or by providing an additional Java module exposing one Java-API call following the design guide lines of the ***updateFirmware()*** method of JavaPOS.

c) Or by providing a separate program which can be used in Windows batch scripts unattended without user interactions.

In case a) and b) is offered, the Integration Provider has to provide a program calling the ***updateFirmware()*** method of the JavaPOS Module or - for case b) - the appropriate method of the additional Java module as delivered by the vendor. This program can be used in Windows batch scripts unattended without user interactions.

For case a) and b) there should be a Java-API call in a corresponding JAR file which transfers the file name and starts the download process. When download is finished the call returns. During the download it is possible to receive events with a parameter containing the current percentage of download state. The Caller program may register to the firmware download module and receive the events to get current status information of how much of the download is performed. When sending the event with a value 100% the download is finished.

Downloading the firmware will be rejected if the given file does not match to the connected hardware or the firmware does not match with the minimum hardware revision level expected from the firmware.

The downloading configuration data, e.g. for scanners printers, scales, cash changers there should be provided a similar way as for the firmware download. The module should be able to download a predefined configuration, e.g. for a scanner in the same manner. For scanners, such configuration items are mainly the same as to be configured by scanning manually configuration labels from pages from the Scanner Guide. For printers, such parameters may be some settings which are persistently stored in the device and described in the manual of the printer.

6 Appendix

6.1 Fields in a Monitoring LOG file

Each LOG entry has the following form and is exactly one line – means it is terminated by the new line character:

```
dateTime|severity|computerName|componentName|instanceName|
errorCode|errorCodeExtended|errortext [|additionalText]
```

The fields are separated by the „|“ (Pipe) character. The field *additionalText* can optionally also in the line.

An example of such a LOG file:

```
25.10.2005 11:50:52.234|1|kasse103|JavaPOS.POSPrinter| WN_TH230_USB|0|11| Printer cover has been opened
25.10.2005 11:58:12.001|3|kasse103|JavaPOS.POSPrinter| WN_TP07_USB|108|0| error: printer is offline and does not respond!
```

6.1.1 Field <dateTime> of LOG file

This parameter defines the time when the monitoring event has been created. Of course, prerequisite for a valid date and time is that the clock is set with the right time.

The form shall be as “DD.MM.YYYY HH:mm:ss.msec”

Code example for Java:

```
/** Returns the current date by "DD.MM.YYYY HH:MM:SS.ms"*/
public static String getDateString()
{
   java.util.Calendar c = Calendar.getInstance(); // new Calendar.();
   int day = c.get(Calendar.DAY_OF_MONTH), month = c.get(Calendar.MONTH), year= c.get(Calendar.YEAR);
   int hour = c.get(Calendar.HOUR_OF_DAY), minute = c.get(Calendar.MINUTE), sec= c.get(Calendar.SECOND);


   String sMsec = "000" + ( (int)(System.currentTimeMillis() %1000) );
   sMsec = sMsec.substring(sMsec.length()-3);

   return formatNum(day,2)+"."+formatNum(month+1,2)+"."+formatNum(year,2)+" "
      +formatNum(hour,2)+":"+formatNum(minute,2)+":"+formatNum(sec,2) + “.”+ sMsec;
}
public static String formatNum( int val, int numOfDigits)
{
   String x="00000000000000000000000";
   String xval="" +val;
   if (xval.length()>= numOfDigits) return xval;
   x= x+xval;
   return x.substring(x.length() -numOfDigits);
}
```

6.1.2 Field <severity> of LOG file

The severity field defines the relevance of an entry.

Constant	Value	Meaning
JavaCIMAdapter.HINT	0	simple informative hint: (no action necessary)
JavaCIMAdapter.NOTICE	1	simple informative, more important hint: (no action necessary) *JavaPOS StatusUpdate Events shall sent always with this type of serverity*
JavaCIMAdapter.WARNING	2	a warning information for example wrong configuration (possibly action needed)
JavaCIMAdapter.ERROR	3	The information represents a real error which has been occurred. (usually action needed) *(JavaPOS ErrorEvents are sent as error notifications.)* Example: printer is offline, paper-end, device not responding
JavaCIMAdapter.FATALERROR	4	important error with bigger impact. for example device broken, device in state where a really a technician has to exchange a module (urgent action needed)

6.1.3 Field <computerName> of LOG file

This parameter defines the name of the host. This is either the host name or the IP address of the client. Under Java the name can be generated from the package java.net.

6.1.4 Field <componentName> of LOG file

This field defines the type of device. Typically the UPOS standardized device category names pre-fixed by “JavaPOS_” is used.

Examples:

JavaPOS_POSPrinter
JavaPOS_MSR
JavaPOS_Scanner
JavaPOS_Scale
JavaPOS_Keylock
JavaPOS_POSKeyboard
JavaPOS_CashDrawer
JavaPOS_CashChanger

6.1.5 Field <instanceName> of LOG file

This field shall contain the used open name (logical devicename) from the POS application provider of the JavaPOS device.

6.1.6 Field <errorCode> of LOG file

This field shall contain a typical error code as defined by the JavaPOS standard in [1].

ErrorCode This section lists the general meanings of the error code property of an **ErrorEvent** or a **JposException**. In general, the property and method descriptions in later chapters list error codes only when specific details or information are added to these general meanings. The error code is set to one of the following values:		
JPOS_E_CLOSED	101	An attempt was made to access a closed JavaPOS Device.
JPOS_E_CLAIMED	102	An attempt was made to access a Physical Device that is claimed by another Device Control instance. The other Control must release the Physical Device before this access may be made. For exclusive-use devices, the application will also need to claim the Physical Device before the access is legal
JPOS_E_NOTCLAIMED	103	An attempt was made to access an exclusive-use device that must be claimed before the method or property set action can be used. If the Physical Device is already claimed by another Device Control instance, then the status JPOS_E_CLAIMED is returned instead.
JPOS_E_NOSERVICE	104	The Control cannot communicate with the Service, normally because of a setup or configuration error.
JPOS_E_DISABLED	105	Cannot perform this operation while the Device is disabled.
JPOS_E_ILLEGAL	106	An attempt was made to perform an illegal or unsupported operation with the Device, or an invalid parameter value was used.
JPOS_E_NOHARDWARE	107	The Physical Device is not connected to the system or is not powered on
JPOS_E_OFFLINE	108	The Physical Device is off-line.
JPOS_E_NOEXIST	109	The file name (or other specified value) does not exist.
JPOS_E_EXISTS	110	The file name (or other specified value) already exists.
JPOS_E_FAILURE	111	The Device cannot perform the requested procedure, even though the Physical Device is connected to the system, powered on, and on-line.
JPOS_E_TIMEOUT	112	The Service timed out waiting for a response from the Physical Device, or the Control timed out waiting for a response from the Service.
JPOS_E_BUSY	113	The current Device Service state does not allow this request. For example, if asynchronous output is in progress, certain methods may not be allowed.
JPOS_E_EXTENDED	114	A device category-specific error condition occurred. The error condition code is available by calling **getErrorCodeExtended**.

6.1.7 Field <errorCodeExtended> of LOG file

This parameter specifies the extended code if needed. For JavaPOS modules this code is represented by the *errorCodeExtended* field in case of error messages of the device service. For warning or hints this parameter may contain detailed information represented by a number individually set by the device service. For OPOS modules, this parameter is represented by the member *ResultCodeExtended* in case of an erroneous action done via the OPOS interface.

So, this parameter is more or less a place holder for additional enumerated status or error codes which are needed because the range of monitoring code does not provide the needed granularity of the possible messages.

6.1.8 Field <errorText> of LOG file

This parameter may contain error describing text - specifying in the language English some detailed information. For JavaPOS modules this parameter is represented by the *errorMessage* field in case the device service throws a *JposException*.

The text may contain special characters. Those with be converted to a %XX form (e.g. the line feed character is converted to %0A) to avoid mismatch in a line due to special characters which have a special meaning in the LOG file.

If this text is not needed the field has to be left empty.

6.1.9 Field <additionalText> of LOG file

This parameter may contain addition describing text or additional data record in their string representing presentation.

As same as for the field ***monitoringText*** also here special characters will be converted in the same manner.

If this text is not needed the field has to be left empty.

7 Conclusion /Feedback

The document describes some additional Guide Lines which are not yet covered by the standard.

Of course, the document is subject of change and is dependent on the input from users, integrators, JavaPOS providers.

Feel free to give any suggestions, hints, feedback to the following email account

retailswsupport@wincor-nixdorf.com

Please make sure that in the email subject contains the text

javaposextension feedback windows.