jPOS Programmer's Guide

Version: 1.6.1

Copyright © 2008 by Alejandro Revilla v
Foreword vi
1. The jPOS Project 1
1.1. About jPOS.org 1
1.2. jPOS License 1
1.3. Downloading jPOS 1
1.4. Directory Structure 2
1.5. Building jPOS 4
2. An ISO-8583 primer 7
2.1. International standard ISO 8583 7
2.2. Message structure 7
2.2.1. ISO-8583 fields 9
2.3. Wire protocol 12
2.4. Message flow 13
3. jPOS approach to ISO-8583 16
3.1. ISOMsg & Co. 16
3.2. Packing and unpacking 18
3.3. Creating custom packagers 20
3.4. Managing the wire protocol with ISOChannel 22
3.4.1. Filtered Channels 26
3.5. Accepting connections with ISOServer 27
3.6. Multiplexing an ISOChannel with a MUX 30
4. jPOS supporting classes 34
4.1. jPOS' Logger 34
4.2. NameRegistrar 37
4.3. Configuration 39
4.4. SystemMonitor 40
4.5. Profiler 42
4.6. DirPoll 43
4.7. ThreadPool 45
5. Implementing Custom Packagers 46
5.1. GenericPackager 47
6. Channel Implementations 51
6.1. TCP/IP Socket-based channels 51
6.2. LoopbackChannel 53
6.3. ChannelPool 54
6.4. SSL 54
7. jPOS Space 56
7.1. Overview 56
7.2. SpaceFactory 57
7.3. Using the space 58
7.4. SpaceTap 60
8. Q2 - second generation QSP 62
8.1. About Q2 62
8.2. Building Q2 62
8.3. Building Q2 Modules 62
8.4. Q2 Primer 62
8.5. Q2 Dynamic ClassLoading 71
8.6. Q2 Scripts 72
8.7. Q2 SpaceLet 73
9. Q2Mod jPOS 75
9.1. ChannelAdaptor 75

9.2. QMUX 78
9.3. QServer 79
9.4. DirPoll 80
9.5. TaskAdaptor 81
9.6. DailyTaskAdaptor 81
9.7. SMAdaptor 81
9.8. KeyStoreAdaptor 82
9.9. QExec 82
9.10. Jetty Integration 82
10. TransactionManager 83
10.1. Overview 83
10.2. Transaction Constants 83
10.3. Transaction Context 84
10.4. Context Recovery interface 85
10.5. AbortParticipant 85
10.6. TransactionManager 86
10.7. Integration with ISORequestListener 87
10.8. GroupSelector 87
10.9. Context implementation 89
11. UI Framework 93
11.1. Overview 93
11.2. UI configuration 94
11.3. menubar 95
11.4. UI Factories 97
11.4.1. JLabelFactory 100
11.4.2. JButtonFactory 101
11.4.3. TextFactory 101
11.4.4. HtmlFactory 102
11.4.5. PanelFactory 103
11.4.6. BorderLayoutFactory 103
11.4.7. GridLayoutFactory 104
11.4.8. HSplitFactory 104
11.4.9. VSplitFactory 105
11.4.10. JTabbedPaneFactory 105
11.4.11. JTreeFactory 106
11.4.12. LogListenerFactory 107
11.4.13. ISOMeterFactory 109
12. Filter Implementations 112
12.1. MD5Filter 112
12.2. MacroFilter 113
12.3. XSLTFilter 114
13. Legacy components 117
13.1. QSP - jPOS' component assembler 117
13.1.1. About QSP 117
13.1.2. Installing QSP 118
13.1.3. Configuration 120
13.1.4. <object> 120
13.1.5. <logger> 129
13.1.6. On-the-fly reconfiguration 132
13.1.7. <task> 132
13.1.8. <channel> 133
13.1.8.1. <filter> 135

13.1.9. <mux> 136
13.1.10. <server> 137
13.1.11. <panel> and <control-panel> 139
13.1.12. <thread-pool> 140
13.1.13. <sequencer> 141
13.1.14. <connection-pool> 142
13.1.15. <persistent-engine> 143
13.1.16. <dir-poll> 144
13.1.17. <daily-task> 146
13.1.18. <card-agent> 146
13.1.19. Security Module 147
13.1.20. Sequencers 148
13.1.21. BlockingQueue 149
13.1.22. LockManager 150
13.1.23. PersistentPeer 150
13.1.24. PersistentEngine 161
13.1.25. ConnectionPool 161
13.1.26. V24 162
Glossary 165
A. Getting involved 166
A.1. 166
B. Bibliography 167
C. ISOFieldPackagers 168
D. qsp-config.dtd 171
E. QTest and QTestMBean source code 174
F. ChangeLog 176
F.1. jPOS 1.4.9 176
F.2. jPOS 1.5.0 177
F.3. jPOS 1.5.1 178
F.4. jPOS 1.5.2 179
F.5. jPOS 1.6.0 179
F.6. jPOS 1.6.1 182
G. Software Copyright 183
H. GNU Affero General Public License version 3 184
I. Acknowledgments 194
Index 195

Copyright © 2008 by Alejandro Revilla

Foreword

This book is designed for developers that have to use jPOS for a given project and have to learn how to use it very quickly.

Although many old-timers from the financial interchange arena are using jPOS, we've identified a pattern where most of new users have to learn how to use jPOS, but also have to learn about credit card processing, ISO-8583, etc. So we'll try to address those items here as well.

This book covers the **jPOS 6** series. jPOS 6 is a major reorganization of our previous projects that provides a software development environment that enable users to work with jPOS, Q2 and jPOS-EE.

Chapter 1. The jPOS Project

1.1. About jPOS.org

The jPOS project is hosted by *jPOS.org* [http://jpos.org].

In order to stay up-to-date with jPOS news, you may want to regularly visit our *website* [http://jpos.org] and follow our blog [http://jpos.org/blog].

In addition, you may want to subscribe to our users' mailing list (jpos-users@googlegroups.com) [groups.google.com/group/jpos-users] and to the developers' mailing list (jpos-dev@yahoogroups.com) [groups.yahoo.com/group/jpos-dev]. or your can follow them over their RSS feeds:

- jPOS Users [http://groups.google.com/group/jpos-users/feed/msgs.xml]
- jPOS Developers [http://rss.gmane.org/gmane.comp.java.jpos.devel]

> **Tip**
> The jPOS activity can be tracked in the jpos-commits@groups.google.com [groups.google.com/group/jpos-commits] which also has an RSS feed [http://groups.google.com/group/jpos-users/feed/msgs.xml]. Changes are updated in our ChangeLog [http://jpos.org/wiki/ChangeLog].

1.2. jPOS License

jPOS is distributed under the GNU Affero General Public License version 3. (see Appendix H, *GNU Affero General Public License version 3* for details).

> **IMPORTANT NOTICE**
> If you don't plan to release your jPOS based application under a compatible license (see frequently asked questions http://www.fsf.org/licensing/licenses/agpl-3.0.html where you can find a license compatibility matrix) **you need to buy a commercial license** (you can contact http://jpos.org/contact?p=CL.Proguide).

1.3. Downloading jPOS

From time to time, we post new jPOS releases at *jPOS.org* (http://www.jpos.org). [http://www.jpos.org] but if you want to stay up to date with jPOS development and you want to get the latest features and bug fixes, we **strongly suggest** you to take the time to install a Subversion client and get the **latest and greatest** version from http://sourceforge.net/projects/jpos [http://sf.net/projects/jpos] SVN repository.

> **Tip**
> The repository has many branches and tags. Unless you are dealing with a legacy jPOS application, You want to get:

```
trunk/jpos6
```

using the command

```
svn co https://jpos.svn.sourceforge.net/svnroot/jpos/trunk/jpos6 jpos
```

.

1.4. Directory Structure

```
jpos6
    |-- COPYRIGHT                                                    (1)
    |-- CREDITS                                                      (2)
    |-- LICENSE                                                      (3)
    |-- README                                                       (4)
    |-- bin                                                          (5)
    |   |-- bsh
    |   |-- q2
    |   `-- release
    |-- build.xml                                                    (6)
    |-- build.properties                                             (7)
    |-- lib                                                          (8)
    |   |-- bsh-2.0b4.jar
    |   |-- jdbm-1.0.jar
    |   |-- jdom-1.0.jar
    |   `-- log4j-1.2.8.jar
    `-- modules                                                      (9)
        |-- ignore.list                                              (10)
        |-- include.list
        |-- jpos                                                     (11)
        |   |-- cfg
        |   |-- src
        |   `-- test
        |-- compat_1_5_2                                             (12)
        |   `-- src
        |-- q2                                                       (13)
        |   |-- deploy
        |   |-- lib
        |   `-- src
        |-- q2mod_jpos                                               (14)
        |   |-- src
        |   `-- test
        |-- security                                                 (15)
        |   `-- src
        `-- txnmgr                                                   (16)
            |-- src
            `-- test
```

(1) jPOS' Copyright notice as described in Appendix G, *Software Copyright*.

(2) A partial list of developers who have made jPOS possible.

(3) GNU General Public License as described in Appendix H, *GNU Affero General Public License version 3* .

(4) Information about how to build jPOS.

(5) Unix/Cygwin scripts that can be used to start the Q2 application.

(6) *Jakarta Ant* [http://ant.apache.org] configuration file. Try `ant -projecthelp` for details.

(7) Build configuration properties.

(8) The `lib` directory contains supporting libraries (jars) that are expected to be available in all jPOS applications.

This is a small list that is incremented by module-specific supporting libraries.

(9) jPOS is composed of **modules** that can be combined in order to build a jPOS application. Depending on your target application, some of those are required and some are optional.

jPOS build system can be extended with user-created modules.

See Section 1.5, “Building jPOS” for further details.

(10) The `ignore.list` as well as the `include.list` files are created the first time that `ant` is invoked. They can be used to customize the list of modules that are to be considered while building a given target jPOS application.

If both are empty, all modules in the `modules` directory are included as part of the build.

(11) The `jpos` module is the main module in the jPOS system containing -- among others -- the ISO-8583 support package (`org.jpos.iso`).

(12) The `compat_1_5_2` contains components that were available in legacy versions up to 1.5.2 (and actually the old development branch 1.5.3) but are probably going to be deprecated.

Tip

We recommend not using them in new applications but if you find them some of them useful, you may want to share with us your use case so we can consider them for re-inclusion in the main set of modules.

(13) Q2 is our JMX-MicroKernel. See Chapter 8, *Q2 - second generation QSP*.

(14) Q2 uses **services** that we call **QBeans**. QBeans can be used for many different purposes such as configuring and interconnecting jPOS components.

This module implements popular components such as the ChannelAdaptor, QServer, QMUX, etc.

(15) This module contains a JCE based `org.jpos.security.SMAdapter` implementation. [1]

(16) The Transaction Manager is a recent addition to the jPOS project that is rapidly gaining popularity among jPOS developers due to the way it fosters code reuse. See Chapter 10, *TransactionManager* for details.

[1] courtesy of Hani S. Kirollos with Egiptian Banks

1.5. Building jPOS

In order to build jPOS you need Apache Ant [http://ant.apache.org] version 1.6.5 or higher installed.

In order to run jPOS a JRE 1.5+ is required.

As an initial sanity check, you want to run:

```
ant -projecthelp
```

You should see an output like this:

```
Buildfile: build.xml

Main targets:

 clean      clean up build and dist directories
 compile    compile all
 dist       Builds source distribution
 jar        creates jpos.jar
 javadoc    generate javadocs
 run        invoke jPOS under Ant's JVM
 singlejar  creates a single jpos.jar including all supporting jars
 test       perform unit tests
 zip        create zip distribution
Default target: compile
```

A typical jPOS application has several runtime directories such as the `deploy` directory that is scanned by the Q2 application in order to start and stop its services, the `cfg` directory used to store runtime configuration and property files, `lib` directory containing supporting jars as well as other directories used for specific applications (such as the `webapps` used by web-enabled applications).

In addition, they have an `src` directory where the source code java files are located as well as a `test` directory which is the place for unit tests.

To recap, we have:

```
src
test
deploy
cfg
```

jPOS build subsystem **merges** these directories off the available modules and **flattens** them inside the automatically created **build** directory.

If we have the files:

```
module/mymoduleA/src/org/jpos/MyClassA.java
module/mymoduleA/lib/mydep1.jar
```

```
module/mymoduleB/src/org/jpos/MyClassB.java
```

```
module/mymoduleB/lib/mydep2.jar
```

and we call `ant` in order to build a jPOS application, as a first step ant will copy the files to the `build/src` directory and `build/lib` directory in order to end up with a directory structure like this:

```
build/src/org/jpos/MyClassA.java
build/src/org/jpos/MyClassB.java
build/lib/mydep1.jar
build/lib/mydep2.jar
```

Same goes for other directories such as `cfg`, and `webapps`.

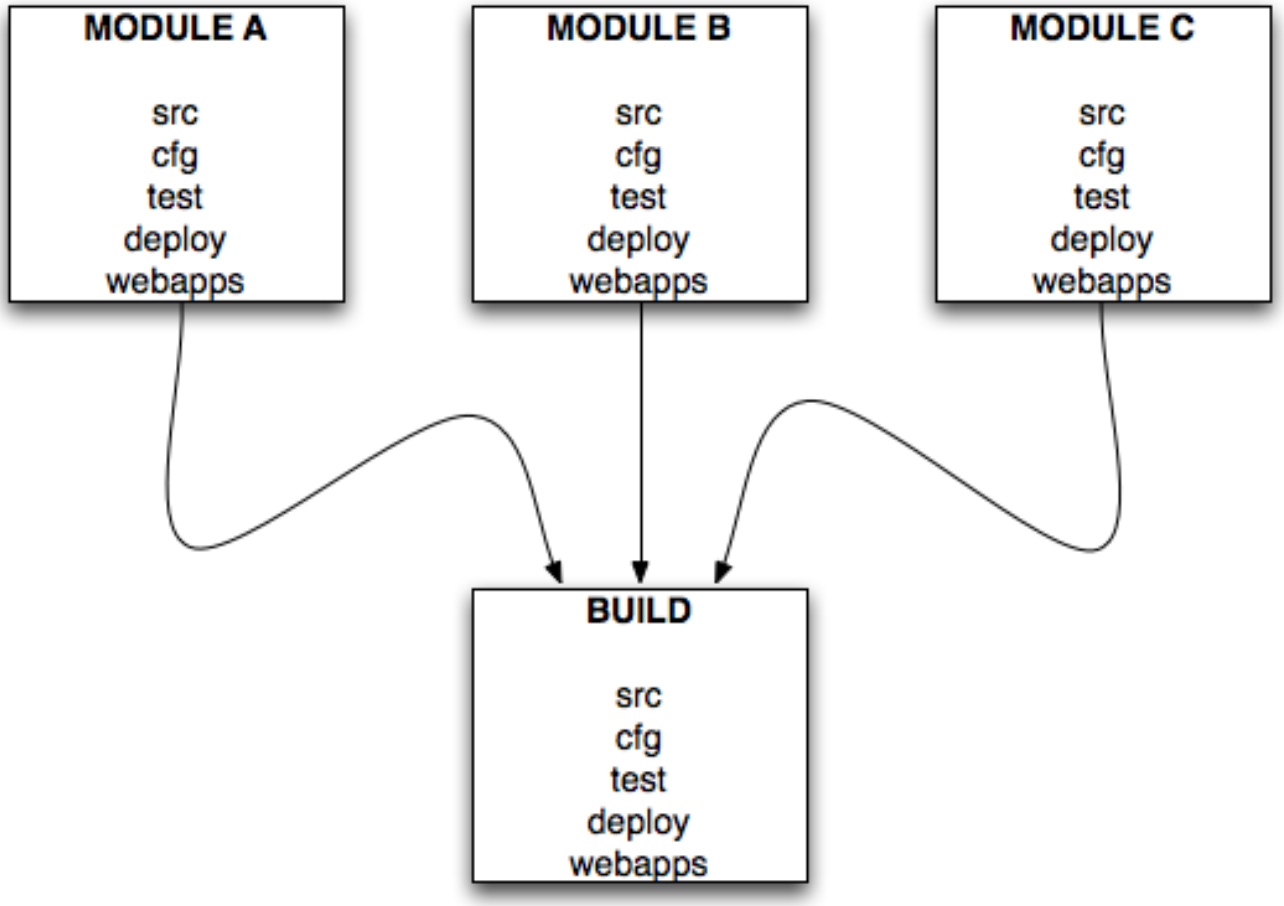


Figure 1.1. Build Merge Process

Tip

If you want to create a local copy of jPOS' javadocs, you can easily issue the following command:

```
ant javadoc
```

You can also browse the javadocs online at http://jpos.org/doc/javadoc/index.html

1. Using jPOS from your IDE

jPOS is IDE-agnostic. If you want to use it in your favorite IDE we recommend that you create the `jpos.jar`

using Ant once and then import it to it.

In order to be able to access the source code from your IDE, we recommend you to move the merged source tree in `build/src` into a directory accesible to your IDE.

2. Running jPOS

Depending on how you get to use jPOS, you can consider it a Library, a Framework or an Application.

While using it as a library, running jPOS boils down to running your application, which in turns will use jPOS.

When you use it as an Application/Framework what you are actually running is **Q2**, jPOS' container. (we cover Q2 in Chapter 8, *Q2 - second generation QSP*.

Running Q2 is as simple as calling:

```
java -jar jpos.jar
```

which produces output like this:

```
<log realm="Q2.system" at="Fri Nov 30 09:09:10 GMT-03:00 2007.775">
  <info>
    Q2 started, deployDir=/home/jpos/svn/jpos/jpos6/build/deploy
  </info>
</log>
```

You can run Q2 off the "build" directory (for testing purposes) but you want to move it to another destination for a production setup.

Tip

You may consider using the jPOS-EE SDK where jPOS becomes just another jPOS-EE module and so does your own application.

There's a unix shell script (that you can run in Windows inside Cygwin or you can use as a sample to create your own .BAT file) called `bin/q2` that can be used to start Q2. (it basically `cd build; java -jar q2.jar`).

In addition, there's an Ant task called "run" that you can use to run Q2 under Ant's control. See `ant -projecthelp` for details.

Chapter 2. An ISO-8583 primer

This chapter contains general information about the ISO-8583 International Standard.

ISO-8583 support is an important part of jPOS. This section outlines ISO-8583 message format, wire protocol and message flow.

2.1. International standard ISO 8583

Financial transaction card-originated messages
Interchange message specifications

You have to read it, period. And you have to read the correct one (1987/1993/2003) for your particular interchange. And you have to read your interchange specs as well. But while you manage to gather all that information, let's have a look at this brief introduction.

When talking about ISO-8583, we have to be aware of the difference between message format (its binary representation), wire protocol (how a message is transmitted over the wire), and message flow (e.g., send request for authorization, wait for response, retransmit, reversal, etc.).

2.2. Message structure

ISO-8583 messages are composed by fields, which are represented in different ways. Basically we have the following structure:

Table 2.1. ISO-8583 message structure

Field #	Description
0 - MTI	Message Type Indicator
1 - Bitmap	64 (or 128) bits indicating presence/absence of other fields
2 .. 128	Other fields as specified in bitmap

So let's have a look at a simple example:

Table 2.2. Sample 0800 message

#	Name	Value	Hex Value
0	MTI	0800	08 00
1	PRIMARY BITMAP	Indicates presence of fields 3, 11 and 41	`20 20 00 00 00 80 00 00`
3	PROCESSING CODE	000000	`00 00 00`
11	SYSTEM TRACE AUDIT NUMBER	000001	`00 00 01`
41	CARD ACCEPTOR TERMINAL IDENTIFICACION	29110001	`32 39 31 31 30 30 30 31`

Here is the binary representation of our 0800 message:

```
080020200000008000000000000000013239313130303031
```

In the previous example, 0800 is the **message type indicator ('MTI')**; The first position represents ISO-8583 version number:

- 0 for version 1987
- 1 for version 1993
- 2 for version 2003
- 3-7 reserved for ISO use
- 8 is reserved for national use
- 9 is reserved for private use

The second position represents **message class**:

- 0 is reserved for ISO use
- 1 authorization
- 2 financial
- 3 file update
- 4 reversals and chargebacks
- 5 reconciliation
- 6 administrative
- 7 fee collection
- 8 network management
- 9 reserved for ISO use

The third position is the **message function**:

- 0 request
- 1 request response
- 2 advice
- 3 advice response
- 4 notification
- 5-9 reserved for ISO use

And the last position is used to indicate the **transaction originator**:

- 0 acquirer
- 1 acquirer repeat
- 2 card issuer
- 3 card issuer repeat
- 4 other
- 5 other repeat
- 6-9 reserved for ISO use

So "0800" is a version 1987 network management request.

Next we have field 1, the primary bitmap:

Table 2.3. Primary Bitmap

byte	hex value	bit value	field #
0	20	`0010 0000`	3
1	20	`0010 0000`	11
2	00	`0000 0000`	
3	00	`0000 0000`	
4	00	`0000 0000`	
5	80	`1000 0000`	41
6	00	`0000 0000`	
7	00	`0000 0000`	

So now that we've parsed the MTI (0800) and bitmap (2020000000800000), we know that fields 3, 11 and 41 are present. So our next field is number 3.

2.2.1. ISO-8583 fields

ISO-8583 specifies different kinds of fields, which basically fall into the following categories:

- Fixed length
  - Numeric
  - Alphanumeric
  - Binary
- Variable length

- Maxlength 99
  - Numeric
  - Alphanumeric
  - Binary
- Maxlength 999
  - Numeric
  - Alphanumeric
  - Binary
- Maxlength 9999 (ISO-8583 version 2003 only)
  - Numeric
  - Alphanumeric
  - Binary
- Nested message

So far, so good, this is very simple stuff, isn't it? The problem is not complexity but diversity, ISO-8583 is not specific about how a given field is represented, so you can have a numeric field represented as a sequence of ASCII characters, EBCDIC characters, BCD, etc.

Variable length fields have a prefix specifying its length, but how this is represented is not defined. Different vendors use different representations (e.g., BCD, EBCDIC, binary value).

In our example, field #3 is using a BCD representation, so a value of "000000" is represented with just three bytes whose hex values are "00 00 00". Same goes for field #11 whose value is "000001" - it is represented as "00 00 01". In our example, field #41 is an eight-byte alphanumeric field represented as eight ASCII characters

```
 Message: 08002020 00000080 00000000 00000001
          32393131 30303031

     MTI: 0800
  Bitmap: 20200000 00800000
Field 03: 000000
Field 11: 000001
field 41: 3239313130303031 (ASCII for "29110001")
```

Let's have a look at another sample message:

Table 2.4. Another 0800 message

#	Name	Value	Hex Value
0	MTI	0800	`08 00`
1.a	PRIMARY BITMAP	Indicates presence of secondary bitmap plus fields	`A0 20 00 00 00 80 00 10`

#	Name	Value	Hex Value
		3, 11 and 41	
1.b	SECONDARY BITMAP	Indicates presence of field 70	`04 00 00 00 00 00 00 00`
3	PROCESSING CODE	000000	`00 00 00`
11	SYSTEM TRACE AUDIT NUMBER	000001	`00 00 01`
41	CARD ACCEPTOR TERMINAL IDEN-TIFICACION	29110001	`32 39 31 31 30 30 30 31`
60	RESERVED FOR PRIVATE USE	jPOS 1.4.1	`00 10 6A 50 4F 53 20 31 2E 34 2E 31`
70	NETWORK MANAGE-MENT INFORMATION CODE	301	`03 01`

Two new fields are present: #60 and #70. Here is our message representation:

```
         Message: 0800A020 00000080 00100400 00000000
                  00000000 00000001 32393131 30303031
                  00106A50 4F532031 2E342E31 0301

             MTI: 0800
  Primary bitmap: A0200000 00800010
Secondary bitmap: 04000000 00000000
        Field 03: 000000
        Field 11: 000001
        Field 41: 3239313130303031 (ASCII for "29110001")
        Field 60: 0010 6A504F5320312E342E31 (length=10, value="jPOS 1.4.1")
        Field 70: 0301
```

Let's break down this bitmap:

Table 2.5. Primary Bitmap

byte	hex value	bit value	field #
0	A0	`1010 0000`	secondary bitmap present plus #3
1	20	`0010 0000`	11
2	00	`0000 0000`	
3	00	`0000 0000`	
4	00	`0000 0000`	
5	80	`1000 0000`	41
6	00	`0000 0000`	
7	10	0001 0000	`60`

Table 2.6. Secondary Bitmap

byte	hex value	bit value	field #
0	04	`0000 0100`	70
1	00	`0000 0000`	
2	00	`0000 0000`	
3	00	`0000 0000`	
4	00	`0000 0000`	
5	80	`0000 0000`	
6	00	`0000 0000`	
7	00	`0000 0000`	

To make things complex to developers, different vendors choose different padding styles when handling odd-length BCD fields. So in order to represent "003" one vendor may use two bytes with the values `"00 03"` while others may use `"00 30"` or even `"00 3F"`.

Same goes for variable-length fields: field length as well as field values can be padded right or left (that's not defined by ISO-8583, it's just a matter of fact of different implementations).

Then we have nested fields - some implementations use "reserved for private use" fields to carry other ISO-8583 messages. These messages are usually packed as variable-length binary fields as seen by the outer message.

Later we will see how jPOS handles this problem in a very simple way so you don't have to worry about this low-level stuff.

2.3. Wire protocol

Once we have a binary representation of a given ISO-8583 message, we have to transmit it over the wire using some communication protocol (e.g., TCP/IP, UDP/IP, X.25, SDLC, SNA, ASYNC, QTP, SSL, etc.).

That communication protocol is not part of the ISO-8583 definition, so different vendors have choosen different protocols.

Many implementations (especially the older ones) require support for some kind of routing information (e.g., a CICS transaction name), so they use different sorts of `headers`.

A few of them (especially stream-based ones) require some kind of trailers as well.

So wire protocol is composed by:

- An optional header
- ISO-8583 message data
- An optional trailer

A TCP/IP-based implementation may use a couple of bytes to indicate message length, so our 0800 example described earlier would be sent as:

```
00 46 08 00 A0 20 00 00  00 80 00 10 04 00 00 00
00 00 00 00 00 00 00 00  00 01 32 39 31 31 30 30
30 31 00 10 6A 50 4F 53  20 31 2E 34 2E 31 03 01
```

0046 being the message length expressed in network byte order.

But this is just one way of specifying message length. Other implementations may choose to send four ASCII bytes, e.g.:

```
30 30 34 36 08 00 A0 20  00 00 00 80 00 10 04 00
00 00 00 00 00 00 00 00  00 00 00 01 32 39 31 31
30 30 30 31 00 10 6A 50  4F 53 20 31 2E 34 2E 31
03 01
```

`30 30 34 36` being the ASCII representation of "0046".

A few of them perform odd things with those headers, flagging rejected messages (e.g., you send a 0100 and instead of receiving a 0110 with a suitable response code you get back your own 0100 with some proprietary flag in the header indicating for example a temporarily failure such as destination unreachable).

Tip

It's very important to read your interchange specification(s) as soon as possible.

Later you'll see how jPOS deals with wire protocol by using a set of classes called ISOChannels, which hide wire protocol details and transparently interact with other jPOS components.

2.4. Message flow

Message flow will vary depending on your particular interchange specification. But let's have a look at a simple example:

Table 2.7. Sample authorization

Time	Acquirer	Issuer	Description
t0	0100 -->		authorization request
t1		<-- 0110	authorization response

While this is the typical case (you send a request, you get a response), sometimes there are temporary failures, and you don't get a response. You have to reverse the previously transmited transaction and then either retry your authorization request, abort that transaction or get an authorization approval by other means (e.g., by phone) and send an advice.

Table 2.8. Authorization timeout

Time	Acquirer	Issuer	Description
t0	0100 -->		authorization request

Time	Acquirer	Issuer	Description
t1			no response
t3	0400 -->		reverse previous authorization
t4		<-- 0410	reverse received
t5	0120 -->		authorization advice
t6		<-- 0130	advice received

Depending on your particular implementation, you may be able to send retransmissions as well (e.g., 0101 after an unanswered 0100). Some implementations use private messages (e.g., 9600) to request extended time to process a transaction. So you can see it is very important to become familiar with your interchange specifications as soon as possible.

jPOS provides tools to deal with message structure, wire protocol and message flow, but it's the responsibility of your higher-level application to interface the message flow with your business logic.

A real example may help you get the idea of what kind of information is exchanged during an authorization request and response. See below:

Table 2.9. Sample authorization request

Field #	Description	Value	Comments
0	MTI	0100	Authorization request
2	Primary Account Number	4321123443211234	
3	Processing Code	000000	
4	Amount transaction	000000012300	i.e., 123.00
7	Transmission data/ time	0304054133	MMYYHHMMSS
11	System trace audit number	001205	
14	Expiration date	0205	YYMM
18	Merchant Type	5399	
22	POS Entry Mode	022	Swiped Card
25	POS Condition Code	00	
35	Track 2	4321123443211234=02051231 2312332	
37	Retrieval Reference Number	206305000014	
41	Terminal ID	29110001	
42	Merchant ID	1001001	

Field #	Description	Value	Comments
49	Currency	840	American Dollars

Table 2.10. Sample authorization response

Field #	Description	Value	Comments
0	MTI	0110	Authorization response
2	Primary Account Number	4321123443211234	
3	Processing Code	000000	
4	Amount transaction	000000012300	i.e., 123.00
7	Transmission data/ time	0304054133	MMYYHHMMSS
11	System trace audit number	001205	
14	Expiration date	0205	YYMM
18	Merchant Type	5399	
22	POS Entry Mode	022	Swiped Card
25	POS Condition Code	00	
35	Track 2	4321123443211234=02051231 2312332	
37	Retrieval Reference Number	206305000014	
38	Authorization number	010305	
39	Response code	00	Approved
41	Terminal ID	29110001	
42	Merchant ID	1001001	
49	Currency	840	American Dollars

Chapter 3. jPOS approach to ISO-8583

This chapter describes how jPOS handles ISO-8583 messages.

3.1. ISOMsg & Co.

jPOS' internal representation of an ISO-8583 message is usually an ISOMsg object (or an ISOMsg's subclass).

The ISOMsg class uses the so-called **Composite pattern** (see *Design Patterns, elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson and Vlissides)

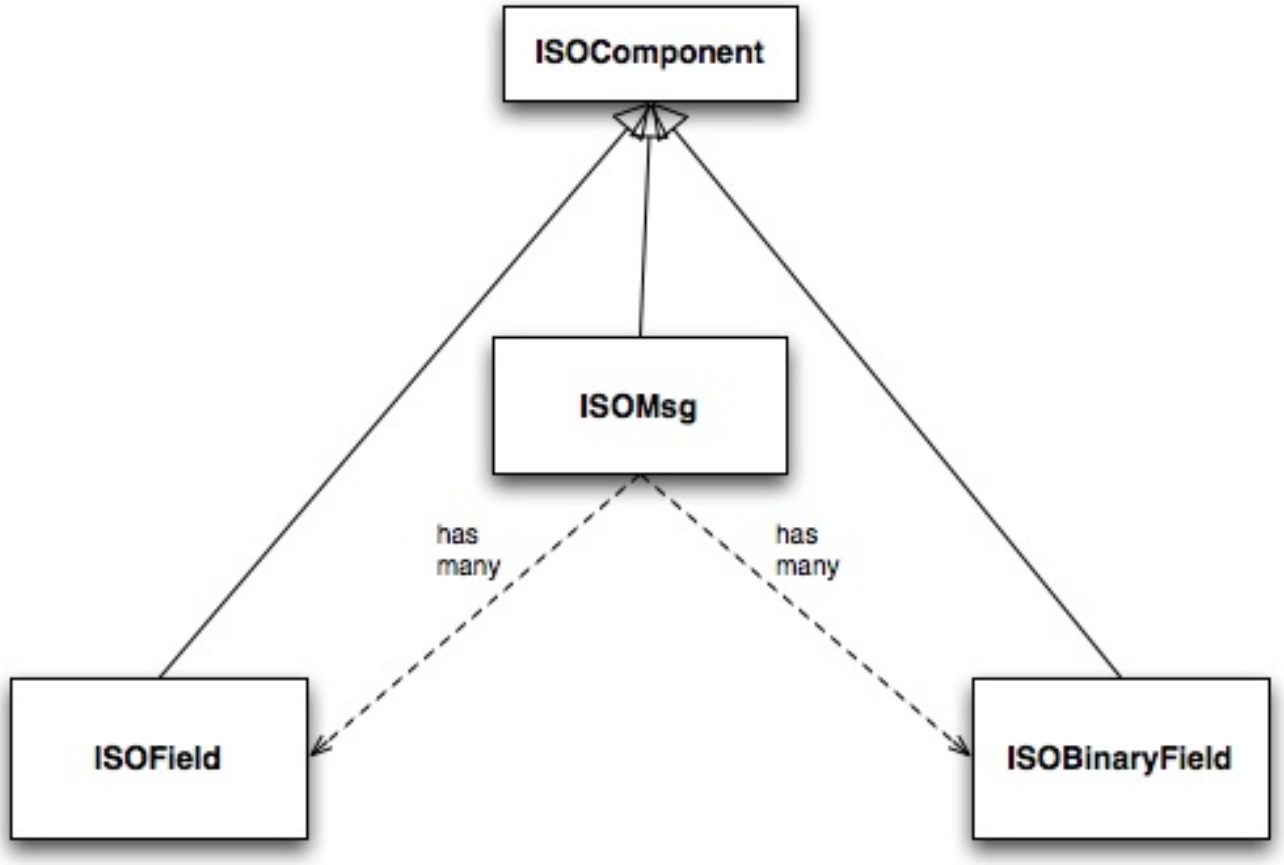
ISOMsg, ISOField, ISOBitMapField, ISOBinaryField and any custom field type that you may implement are subclasses of ISOComponent. Let's have a look at ISOComponent's methods:

```
public abstract class ISOComponent implements Cloneable {
    public void set (ISOComponent c) throws ISOException;
    public void unset (int fldno) throws ISOException;
    public ISOComponent getComposite();
    public Object getKey() throws ISOException;
    public Object getValue() throws ISOException;
    public byte[] getBytes() throws ISOException;
    public int getMaxField();
    public Hashtable getChildren();
    public abstract void setFieldNumber (int fieldNumber);
    public abstract void setValue(Object obj) throws ISOException;
    public abstract byte[] pack() throws ISOException;
    public abstract int unpack(byte[] b) throws ISOException;
    public abstract void dump (PrintStream p, String indent);
    public abstract void pack (OutputStream out) throws IOException, ISOException;
    public abstract void unpack (InputStream in) throws IOException, ISOException;
}
```

This approach has proven to be really useful and maps quite well to the ISO-8583 message structure.

There are many situations where some methods are not applicable (i.e., getChildren() has no meaning in a leaf field, same goes for methods such as getMaxField()), but as a general rule, using the same super-class for ISOMsg and ISOFields has proven to be a good thing. You can easily assign an ISOMsg as a field of an outer ISOMsg.

The following diagram shows how some ISOComponents interact with each other.



The following code can be used to create an internal representation of our 0800 message (described in [An ISO-8583 primer] chapter).

Example 3.1. ISOMsg/ISOField

```
import org.jpos.iso.*;

ISOMsg m = new ISOMsg();
m.set (new ISOField (0, "0800"));
m.set (new ISOField (3, "000000"));
m.set (new ISOField (11, "000001"));
m.set (new ISOField (41, "29110001"));
m.set (new ISOField (60, "jPOS 6"));
m.set (new ISOField (70, "301"));
```

We are just calling `ISOComponent.set (ISOComponent)` method.

In order to reduce typing and improve code readability, ISOMsg provides some handy methods such as

```
ISOMsg.setMTI (String)
```

and

```
ISOMsg.set (int fieldNumber, String fieldValue)
```

implemented like this:

```
public void set (int fldno, String value) throws ISOException {
    set (new ISOField (fldno, value));
}
```

```
    public void setMTI (String mti) throws ISOException {
        if (isInner())
            throw new ISOException ("can't setMTI on inner message");
        set (new ISOField (0, mti));
    }
```

So the previous example can be written like this:

Example 3.2. ISOMsg

```
ISOMsg m = new ISOMsg();
m.setMTI ("0800");
m.set (3, "000000");
m.set (11, "000001");
m.set (41, "29110001");
m.set (60, "jPOS 6");
m.set (70, "301");
```

Tip

ISOMsg is one of the most used classes in typical ISO-8583-based jPOS applications. While you can subclass it, you probably won't have to.

If there's a single class in all jPOS that you want to study in great detail, this is it. We recommend you to have a look at its API documentation [http://jpos.org/doc/javadoc/org/jpos/iso/ISOMsg.html] and play with its helper methods such as clone, merge, unset, etc.

3.2. Packing and unpacking

ISOComponents have two useful methods called:

```
public abstract byte[] pack() throws ISOException;
public abstract int unpack(byte[] b) throws ISOException;
```

pack returns a byte[] containing the binary representation of a given component (can be just a field or the whole ISOMsg); unpack does the opposite and also returns the number of consumed bytes.

jPOS uses yet another pattern called **Peer pattern** that allows a given ISOComponent to be packed and unpacked by a peer class, "plugged" at runtime.

So ISOMsg has a method called:

```
public void setPackager (ISOPackager p);
```

where one can assign a given packager to an ISOMsg, so you can write code like this:

Example 3.3. ISOPackager

```
ISOPackager customPackager = MyCustomPackager ();
ISOMsg m = new ISOMsg();
```

```
m.setMTI ("0800");
m.set (3, "000000");
m.set (11, "000001");
m.set (41, "29110001");
m.set (60, "jPOS 6");
m.set (70, "301");
m.setPackager (customPackager);
byte[] binaryImage = m.pack();
```

In order to unpack this binary image you may write code like this:

```
ISOPackager customPackager = MyCustomPackager ();
ISOMsg m = new ISOMsg();
m.setPackager (customPackager);
m.unpack (binaryImage);
```

It is very easy to create protocol converters using jPOS, e.g.:

```
ISOPackager packagerA = MyCustomPackagerA ();
ISOPackager packagerB = MyCustomPackagerB ();
ISOMsg m = new ISOMsg();
m.setPackager (packagerA);
m.unpack (binaryImage);
m.setPackager (packagerB);
byte[] convertedBinaryImage = m.pack();
```

ISOMsg.pack() delegates message packing/unpacking operations to its underlying "peer" ISOPackager. The code looks like this:

```
public byte[] pack() throws ISOException {
    synchronized (this) {
        recalcBitMap();
        return packager.pack(this);
    }
}
```

`packager.pack (ISOComponent)` also delegates its packing/unpacking duties to an underlying ISOFieldPackager. There are ISOFieldPackager implementations for many different ways of representing a field. It is very easy to create your own, if required.

The following code is used by an ISOFieldPackager implementation to pack and unpack fixed-length alphanumeric fields:

```
public byte[] pack (ISOComponent c) throws ISOException {
    String s = (String) c.getValue();
    if (s.length() > getLength())
        s = s.substring(0, getLength());
    return (ISOUtil.strpad (s, getLength())).getBytes();
}
public int unpack (ISOComponent c, byte[] b, int offset)
    throws ISOException
{
    c.setValue(new String(b, offset, getLength()));
    return getLength();
}
```

jPOS comes with many ISOFieldPackager implementations. You'll probably never have to write your own. Names choosen are somewhat cryptic, but so many people are using them for their own custom packagers so we'll probably have to live with those names for a while.

As a general rule, all ISOFieldPackagers live under package `org.jpos.iso` and start with the name **IF*** which stands for "ISO Field".

So we have things like this:

Table 3.1. ISOFieldPackagers

Name	Purpose
IF_CHAR	Fixed length alphanumeric (ASCII)
IFE_CHAR	Fixed length alphanumeric (EBCDIC)
IFA_NUMERIC	Fixed length numeric (ASCII)
IFE_NUMERIC	Fixed length numeric (EBCDIC)
IFB_NUMERIC	Fixed length numeric (BCD)
IFB_LLNUM	Variable length numeric (BCD, maxlength=99)
IFB_LLLNUM	Variable length numeric (BCD, maxlength=999)
IFB_LLLLNUM	Variable length numeric (BCD, maxlength=9999)
...	...
...	...

[See appendix B for a complete reference]

3.3. Creating custom packagers

jPOS provides the ability to create customized packagers for different kind of ISO-8583 implementations. Over the last few years, several developers have contributed their customized ISOPackagers and ISOFieldPackagers, so chances are good that you can find an implementation suitable for you, or something very close to what you need as part of jPOS distribution.

Tip

Before writing your own packager, have a look at the `modules/jpos/src/main/org/jpos/iso/packager` directory.

Writing a packager is very easy. There's a support class called ISOBasePackager that you can easily extend, e.g.:

```
public class ISO93APackager extends ISOBasePackager {
    protected ISOFieldPackager fld[] = {
    /*000*/ new IFA_NUMERIC (  4, "Message Type Indicator"),
    /*001*/ new IFA_BITMAP  ( 16, "Bitmap"),
    /*002*/ new IFA_LLNUM   ( 19, "Primary Account number"),
    /*003*/ new IFA_NUMERIC (  6, "Processing Code"),
    /*004*/ new IFA_NUMERIC ( 12, "Amount, Transaction"),
```

```
    /*005*/ new IFA_NUMERIC ( 12, "Amount, Reconciliation"),
    ...
    ...
    ...
    public ISO93APackager() {
        super();
        setFieldPackager(fld);
    }
}
```

So the programmer's task (BTW, an easy but boring one) is to verify that every single field in your packager configuration matches your interchange specifications.

An ISOPackager is not required to extend the supporting class ISOBasePackager, but we've found it quite convenient for most situations.

Tip

while you write your own packager implementation, we recommend you to write a unit test for it. Have a look at `modules/jpos/test/org/jpos/iso` directory to find some sample unit tests that can be used as a starting point.

After adding several packagers to our repository, jPOS developer Eoin Flood came up with a good idea: a GenericPackager that one could configure by means of an XML file. The GenericPackager configuration looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE isopackager SYSTEM "genericpackager.dtd">

<!-- ISO 8583:1993 (ASCII) field descriptions for GenericPackager -->

<isopackager>
  <isofield
      id="0"
      length="4"
      name="Message Type Indicator"
      class="org.jpos.iso.IFA_NUMERIC"/>
  <isofield
      id="1"
      length="16"
      name="Bitmap"
      class="org.jpos.iso.IFA_BITMAP"/>
  <isofield
      id="2"
      length="19"
      name="Primary Account number"
      class="org.jpos.iso.IFA_LLNUM"/>
  <isofield
      id="3"
      length="6"
      name="Processing Code"
      class="org.jpos.iso.IFA_NUMERIC"/>
  <isofield
      id="4"
      length="12"
      name="Amount, Transaction"
      class="org.jpos.iso.IFA_NUMERIC"/>
  <isofield
      id="5"
      length="12"
      name="Amount, Reconciliation"
      class="org.jpos.iso.IFA_NUMERIC"/>
  <isofield
```

```
      id="6"
      length="12"
      name="Amount, Cardholder billing"
      class="org.jpos.iso.IFA_NUMERIC"/>
      ...
      ...
      ...
</isopackager>
```

So now we have XML configurations for most packagers under the `org.jpos.iso.packager` package. They are available in the `src/config/packager` directory. If you are to develop a custom packager, we encourage you to use GenericPackager with a suitable custom configuration file instead. It will greately simplify your task.

Example 3.4.

We recommend that whenever possible, you use code like this:

```
    import org.jpos.iso.ISOPackager;
    import org.jpos.iso.packager.GenericPackager;
    import org.jpos.iso.packager.ISO87BPackager;
    ISOPackager p = new GenericPackager ("iso87binary.xml");
```

...instead of like this:

```
    import org.jpos.iso.ISOPackager;
    import org.jpos.iso.packager.ISO87BPackager;
    ISOPackager p = new ISO87BPackager();
```

Tip

If you're using Q2 to configure your packagers, GenericPackager uses the "packager-config" property in order to determine its configuration file.

Tip

If you need support for nested messages, you may want to have a look at src/config/packager/genericpackager.dtd as well as examples such as europay.xml (see field 48) or base1.xml (see field 127).

3.4. Managing the wire protocol with ISOChannel

jPOS uses an interface called ISOChannel to encapsulate wire protocol details.

ISOChannel is used to send and receive ISOMsg objects. It leverages the **peer pattern** where its *peer* is an ISOPackager instance. It has send and receive methods as well as means to set and get a peer packager:

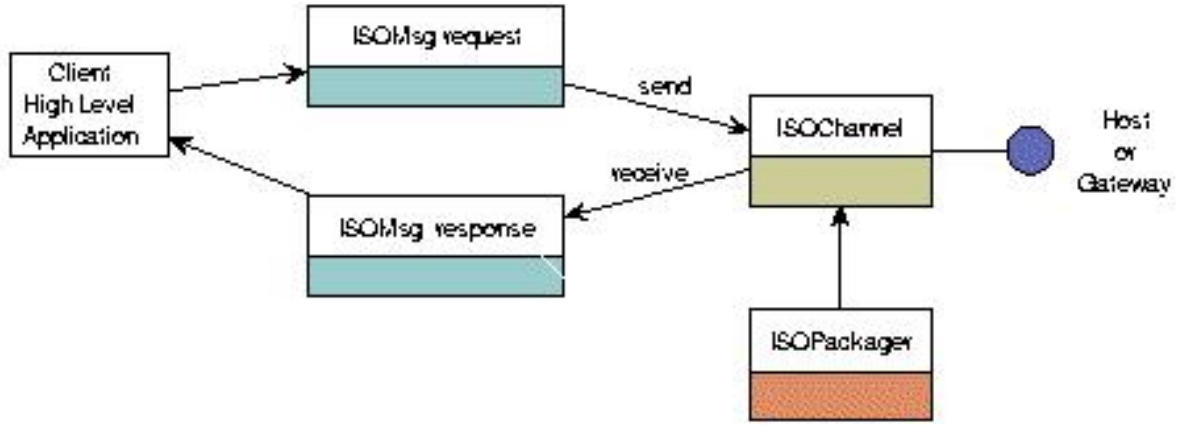
```
    ...
    public void send (ISOMsg m) throws IOException, ISOException;
    public ISOMsg receive() throws IOException, ISOException;
    public void setPackager(ISOPackager p);
    public ISOPackager getPackager();
    ...
```

Although not meaningful under all possible situations, ISOChannel has a few connection-related methods as well:

```
...
public void connect () throws IOException;
public void disconnect () throws IOException;
public void reconnect() throws IOException;
public void setUsable(boolean b);
public boolean isConnected();
...
```

In order for applications to bind jPOS components at runtime, there's a **Singleton** [1]class called `org.jpos.util.NameRegistrar` where you can register and get references to Objects. The ISOChannel interface provides handy methods to access ISOChannels at runtime by their name.

```
...
public void setName (String name);
public String getName();
...
```



ISOChannel extends ISOSource which reads like this:

```
public interface ISOSource {
  public void send (ISOMsg m)
   throws IOException, ISOException, VetoException;
  public boolean isConnected();
}
```

Different interchanges use different wire protocols. jPOS encapsulates that functionality in completely isolated ISOChannel implementations. It comes with many implementations and it's easy to write your own.

[1] See Design Patterns, elements of Reusable Object-Oriented Software

Table 3.2. Sample ISOChannel implementations

Name	Description		
ASCIIChannel	4 bytes message length plus ISO-8583 data		
LogChannel	Can be used to read jPOS's logs and inject messages into other channels		
LoopbackChannel	Every message sent gets received (possibly applying filters). Very useful for testing purposes.		
PADChannel	Used to connect to X.25 packet assembler/dissamblers		
XMLChannel	jPOS Internal XML representation for ISO-8583 messages		
...	...		
...	...		

Tip

(see `org.jpos.iso.channel.*` for a complete list)

Example 3.5. ISOChannel

This example sends a simple 0800 message into a local echo server:

Tip

If you are running Unix, please check your inetd setup.

```
import org.jpos.iso.*;
import org.jpos.util.*;
import org.jpos.iso.channel.*;
import org.jpos.iso.packager.*;

public class Test {
    public static void main (String[] args) throws Exception {
        Logger logger = new Logger();
        logger.addListener (new SimpleLogListener (System.out));
        ISOChannel channel = new ASCIIChannel (
            "localhost", 7, new ISO87APackager()
        );
        ((LogSource)channel).setLogger (logger, "test-channel");
        channel.connect ();

        ISOMsg m = new ISOMsg ();
        m.setMTI ("0800");
        m.set (3, "000000");
```

```
            m.set (41, "00000001");
            m.set (70, "301");
            channel.send (m);
            ISOMsg r = channel.receive ();
            channel.disconnect ();
        }
    }
```

If everything works fine, you'll see the following output:

```
<log realm="test-channel" at="Wed Mar 27 20:30:14 GMT-03:00 2002.883">
  <connect>
    localhost:7
  </connect>
</log>
<log realm="test-channel" at="Wed Mar 27 20:30:15 GMT-03:00 2002.153">
  <send>
    <isomsg direction="outgoing">
      <field id="0" value="0800"/>
      <field id="3" value="000000"/>
      <field id="41" value="00000001"/>
      <field id="70" value="301"/>
    </isomsg>
  </send>
</log>
<log realm="test-channel" at="Wed Mar 27 20:30:15 GMT-03:00 2002.224">
  <receive>
    <isomsg direction="incoming">
      <field id="0" value="0800"/>
      <field id="3" value="000000"/>
      <field id="41" value="00000001"/>
      <field id="70" value="301"/>
    </isomsg>
  </receive>
</log>
<log realm="test-channel" at="Wed Mar 27 20:30:15 GMT-03:00 2002.228">
  <disconnect>
    localhost:7
  </disconnect>
</log>
```

Tip

While we'll see many examples similar to the previous one throughout this document, where a simple main() method takes care of instantiating and configuring several jPOS components, later we'll introduce **Q2**, jPOS's component assembler/configurator. We **strongly recommend** to use Q2 to run jPOS. It will make your life easier.

Q2 lets you define your jPOS-based application in a very simple, easy to create and easy to maintain set of XML configuration files.

We recommend that you wait until we talk about Q2 before diving into coding your own jPOS-based application. Using code like the previous example is good to learn jPOS but not to run it in a production environment.

If you have a look at the ISOChannel implementations (most of them live in org.jpos.iso.channel package) you'll notice that many of them extend `org.jpos.iso.BaseChannel`.

BaseChannel is an abstract class that provides hooks and default implementations for several methods that are useful when writing custom channels. While you don't necesarily have to extend BaseChannel to write a custom channel, you'll probably find it very useful.

Depending on your wire protocol, you'll probably only need to extend BaseChannel and just override a few methods, i.e:

```
protected void sendMessageLength(int len) throws IOException;
protected int getMessageLength() throws IOException, ISOException;
```

(see modules/jpos/src/org/jpos/iso/channel/CSChannel.java)

You may want to have a look at the LoopbackChannel implementation for an example of an ISOChannel that doesn't extend BaseChannel.

3.4.1. Filtered Channels

Many ISOChannels implement `FilteredChannel` which looks like this:

```
public interface FilteredChannel extends ISOChannel {
  public void addIncomingFilter (ISOFilter filter);
  public void addOutgoingFilter (ISOFilter filter);
  public void addFilter (ISOFilter filter);
  public void removeFilter (ISOFilter filter);
  public void removeIncomingFilter (ISOFilter filter);
  public void removeOutgoingFilter (ISOFilter filter);
  public Collection getIncomingFilters();
  public Collection getOutgoingFilters();
  public void setIncomingFilters (Collection filters);
  public void setOutgoingFilters (Collection filters);
}
```

`ISOFilter` is very simple as well:

```
public interface ISOFilter {
 public ISOMsg filter (ISOChannel channel, ISOMsg m, LogEvent evt)
    throws VetoException;
}
```

Whenever you add a filter (be it incoming, outgoing, or both) to a FilteredChannel, all messages sent or received by that channel are passed through that filter.

Filters give you the opportunity to stop a given message from being sent or received by that channel, by throwing an ISOFilter.VetoException.

Let's have a look at a very simple filter, DelayFilter:

```
public class DelayFilter implements ISOFilter, ReConfigurable {
    int delay;
    public DelayFilter() {
        super();
        delay = 0;
    }
   /**
    * @param delay desired delay, expressed in milliseconds
    */
```

```
        public DelayFilter(int delay) {
            super();
            this.delay = delay;
        }
       /**
        * @param cfg
        * <ul>
        * <li>delay (expressed in milliseconds)
        * </ul>
        */
        public void setConfiguration (Configuration cfg) {
            delay = cfg.getInt ("delay");
        }

        public ISOMsg filter (ISOChannel channel, ISOMsg m, LogEvent evt)
        {
            evt.addMessage ("<delay-filter delay=\""+delay+"\"/>");
            if (delay > 0) {
                try {
                    Thread.sleep (delay);
                } catch (InterruptedException e) { }
            }
            return m;
        }
    }
```

DelayFilter simply applies a given delay to all traffic being sent or received by a given channel. It can be used to simulate remote host delays, a good tool for testing purposes.

Tip

The previous code introduces a few classes and interfaces, namely Configuration, ReConfigurable, LogEvent. We'll talk about these important parts of jPOS soon.

jPOS comes with many general purpose filters: MD5Filter can be used to authenticate messages; MacroFilter can be used to expand internal variables and sequencers; and XSLTFilter can be used to apply XSLT Transformations to ISO-8583 messages. [See the reference section of this document to get detailed information about available filters.]

There's a popular filter called BSHFilter that can execute BeanShell code placed in an external file that can be modified at runtime without restarting the system, providing an excellent way to make quick changes (which are welcome during tests and initial rounds of certifications - the BSH code can be easily migrated to Java later).

3.5. Accepting connections with ISOServer

ISOServer listens in a given port for incoming connections and takes care of accepting them and passing control to an underlying ISOChannel implementation.

Once a new connection is accepted and an ISOChannel is created, a ThreadPool-controlled Thread takes care of receiving messages from it. Those messages are passed to an ISORequestListener implementation.

Example 3.6. ISOServer

```
import org.jpos.iso.*;
import org.jpos.util.*;
import org.jpos.iso.channel.*;
import org.jpos.iso.packager.*;
```

```
public class Test {
    public static void main (String[] args) throws Exception {
        Logger logger = new Logger ();
        logger.addListener (new SimpleLogListener (System.out));
        ServerChannel channel = new XMLChannel (new XMLPackager());
        ((LogSource)channel).setLogger (logger, "channel");
        ISOServer server = new ISOServer (8000, channel, null);
        server.setLogger (logger, "server");
        new Thread (server).start ();
    }
}
```

Tip

The third argument of ISOServer's constructor is an optional ThreadPool. Should you pass a null parameter there, a new ThreadPool is created for you, which defaults to 100 threads. (new ThreadPool (1,100))

In order to test the previous server Test program (which is listening on port 8000), you can use a simple 'telnet' client where you will be able to type an XML-formatted ISO-8583 message, e.g.:

```
$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Now if you have a look at your running Test program you'll see something like this:

```
<log realm="server" at="Fri May 17 08:11:34 UYT 2002.824">
    <iso-server>
        listening on port 8000
    </iso-server>
</log>
```

Back on your telnet session, you can type in an XML formatted ISO-8583 message like this:

```
<isomsg>
 <field id="0" value="0800"/>
 <field id="3" value="333333"/>
</isomsg>

(please note XMLChannel expects <isomsg> as well as </isomsg>
 to be placed as the first thing in a line)
```

Your test program will then show:

```
<log realm="server.channel" at="Fri May 17 07:56:58 UYT 2002.407">
 <receive>
  <isomsg direction="incoming">
   <field id="0" value="0800"/>
   <field id="3" value="333333"/>
  </isomsg>
 </receive>
</log>
```

As stated above, you can add an ISORequestListener to your ISOServer that will take care of actually pro-

cessing the incoming messages. So let's modify our little Test program to answer our messages. Our Test class has to implement ISORequestListener, e.g.:

```
public class Test implements ISORequestListener {
    ...
    ...
    public boolean process (ISOSource source, ISOMsg m) {
        try {
            m.setResponseMTI ();
            m.set (39, "00");
            source.send (m);
        } catch (ISOException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }
    ...
    ...
}
```

You have to assign this request listener to your server. You can do this assignment with the following instruction:

```
    server.addISORequestListener (new Test ());
```

The full program looks like this:

```
import java.io.*;
import org.jpos.iso.*;
import org.jpos.util.*;
import org.jpos.iso.channel.*;
import org.jpos.iso.packager.*;

public class Test implements ISORequestListener {
    public Test () {
        super();
    }
    public boolean process (ISOSource source, ISOMsg m) {
        try {
            m.setResponseMTI ();
            m.set (39, "00");
            source.send (m);
        } catch (ISOException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }

    public static void main (String[] args) throws Exception {
        Logger logger = new Logger ();
        logger.addListener (new SimpleLogListener (System.out));
        ServerChannel channel = new XMLChannel (new XMLPackager());
        ((LogSource)channel).setLogger (logger, "channel");
        ISOServer server = new ISOServer (8000, channel, null);
        server.setLogger (logger, "server");
        server.addISORequestListener (new Test ());

        new Thread (server).start ();
    }
```

```
}
```

Now try to telnet to port 8000 and send another XML-formatted ISO-8583 message. You'll get a response, with a retcode "00" (field 39), e.g.:

```
(you type)
<isomsg>
 <field id="0" value="0800"/>
 <field id="3" value="333333"/>
</isomsg>

(and you should get)
<isomsg direction="outgoing">
  <field id="0" value="0810"/>
  <field id="3" value="333333"/>
  <field id="39" value="00"/>
</isomsg>
```

ISOServer uses a ThreadPool in order to be able to accept multiple connections at the same time. Every socket connection is handled by a single thread. If your request listener implementation takes too long to reply, new messages arriving over that session will have to wait for their response.

To solve this problem, your ISORequestListener implementation should run in its own thread pool so that its process(...) method will just queue requests to be processed by a peer thread.

ISOServer uses ISOChannel implementations to pull ISOMsgs from the wire. These ISOChannels can, of course, have associated filters as described earlier.

Note

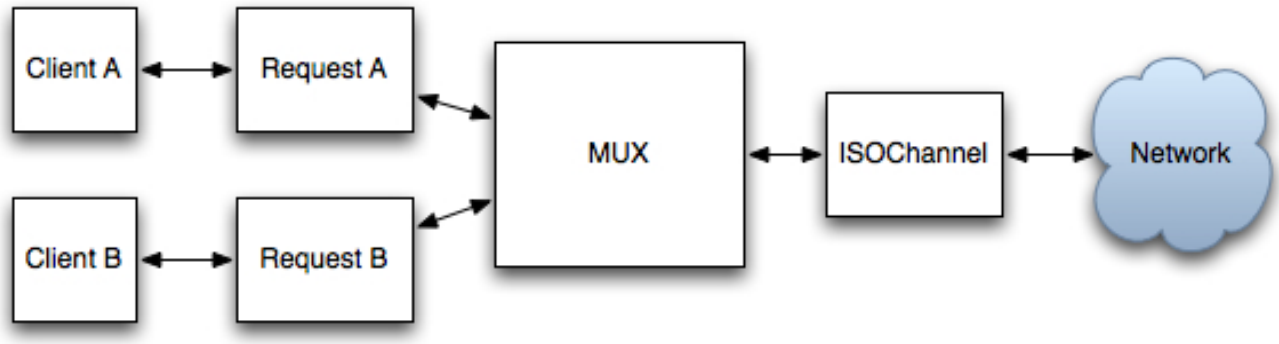
In modern jPOS applications ISOServer is usually managed by the QServer service (see Section 9.3, “QServer”). The ISORequestListener is usually a thin implementation that forwards the request to the TransactionManager (see Chapter 10, *TransactionManager*).

3.6. Multiplexing an ISOChannel with a MUX

Imagine an acquirer implementation that receives several requests at a time from several POS terminals and has to route them to an issuer institution by means of an ISOChannel.

While you can establish one socket connection per transaction, it is common use to setup just one socket connection (handled by an ISOChannel instance) and multiplex it.

So a MUX is basically a **channel multiplexer**. Once you have instantiated a MUX, you just send a request and wait for the response.



Originally, the MUX interface look like this:

```
public interface MUX {
    public ISOMsg request (ISOMsg m, long timeout) throws ISOException;
    public boolean isConnected();
}
```

- The `ISOMsg request(ISOMsg, long)` method queues a request to be sent by the underlying ISOChannel(s) and waits for the response up to the timeout specified in milliseconds. It either returns a response or null.
- `isConnected()` is self explanatory, it returns true if the underlying channel(s) are connected.

> **Note**
>
> MUX is an interface that can have many different implementations. Depending on the implementation and the configuration the value returned by `isConnected()` might not be reliable (it could return true even on an unconnected channel).

Recently [2] we've added the ability to asynchroneusly queue requests, the new MUX interface another `request` method that returns inmediately and calls an ISOResponseListener (with an optional handBack Object).

```
public interface MUX {
    ...
    ...
    public void request
        (ISOMsg m, long timeout, ISOResponseListener r, Object handBack)
    throws ISOException;
}
```

> **Note**
>
> This new asynchroneous way of calling the MUX is available in the `QMUX` implementation of the `MUX` interface but it has not been back-ported to the `ISOMUX` implementation which is going to be deprecated in future versions of jPOS.
>
> ISOMUX has a `queue` method that can be used to achieve a similar asynchroneous behavior.

In order to send responses to the appropiate sending thread, a `MUX` implementation uses selected fields from the original `ISOMsg` request expected to be present in the `ISOMsg` response. Although not part of the `MUX` interface, implementations such as QMUX (the new one) and ISOMUX (the old one) have a protected method called `String getKey(ISOMsg m)` that returns a matching key based on the `ISOMsg` content.

[2] r2510 July 2007

The default implementation uses fields such as 41 (Terminal ID) plus field 11 (Serial Trace Audit Number) to create an unique key. You can override `getKey()` in order to use other fields.

Example 3.7. MUX example

```
...
...
MUX mux = (MUX) NameRegister.get ("mux.mymultiplexer");
...
...
```

Once you have a reference to a MUX implementation you can send requests and wait for responses using code like this:

```
ISOMsg m = new ISOMsg();
m.setMTI ("0800");
m.set (11, "000001");
m.set (41, "00000001");
ISOMsg response = mux.request (m, 30000);
if (response != null) {
    // you've got a response
} else {
    // request has timed out - you may want to reverse
}
```

When a message arrives to MUX's underlying ISOChannel, the MUX implementation checks to see if that message's 'key' is registered as a pending request.

Should that key match a pending request, the response is handed to the waiting thread. If the key was registered as a request, or the response comes in too late then that response is (depending on the configuration) ignored, forwarded to an ISORequestListener or to a well defined Space queue. (see QMUX for details).

Under many situations, the same channel that a client application may use to send requests and wait for responses may also receive requests coming from the remote server.

Those unmatched requests coming from the remote server are delegated to an ISORequestListener (or a well defined "unhandled" Space queue).

Let's have a look at the ISORequestListener interface:

```
public interface ISORequestListener {
    public boolean process (ISOSource source, ISOMsg m);
}
```

Imagine we want to answer the 0800 echo requests arriving to our MUX. We can write the following implementation:

```
public class EchoHandler extends Log
     implements ISORequestListener
{
     public boolean process (ISOSource source, ISOMsg m) {
         try {
             if ("0800".equals (m.getMTI())) {
```

```
                    m.setResponseMTI ();
                    m.set (39, "00");
                    source.send (m);
                }
            } catch (Exception e) {
                warn ("echo-handler", e);
            }
            return true;
        }
    }
```

Chapter 4. jPOS supporting classes

util and core

This chapter describes jPOS's core and util packages which include components such as the Logger subsystem, the Name Registrar, Configuration stuff, Sequencer, Profiler, ThreadPool, etc.

Tip

While you probably want to carefully read Section 4.1, “jPOS' Logger”, Section 4.2, “NameRegistrar” and Section 4.3, “Configuration”, feel free to skip the rest (Profiler, DirPoll, ThreadPool) and use it as a reference only.

4.1. jPOS' Logger

Yet another Logger subsystem

You may wonder why we've choosen to develop our own Logger subsystem. The answer is very simple: when we wrote it, there were no other suitable logger subsystems available. Log4j was just a toy hosted in alphaWorks.

The main difference between our logger sub-system and other logger sub-systems out there is that we deal with **live objects**. A LogEvent holds live objects that can be handled by the LogListeners, for example to protect sensitive information (PCI requirement) or to act on special conditions (i.e. e-mailing an Operator on an Exception).

Note

While other logger subsystems are mostly "line oriented", jPOS' is mostly "transaction oriented". A jPOS LogEvent is likely to carry information for the whole transaction making it very suitable for audit and debugging purposes.

jPOS's logger subsystem is very easy to extend, so one can easily plug in other logger engines (such as Log4j, commons logging or the new JDK's 1.4 logging stuff). Our logger is implemented by the following main classes:

Table 4.1. Logger's main classes

Class	Description
Logger	Main logger class
LogListener	Listens to log events
LogSource	A log event producer has to implement LogSource
LogEvent	The Log Event

The Logger class has the following important methods:

```
public class Logger {
   public static void log (LogEvent ev);
   ...
   public void addListener (LogListener l);
   public void removeListener (LogListener l);
```

```
    public boolean hasListeners();
    ...
    ...
}
```

LogSource looks like this:

```
public interface LogSource {
   public void setLogger (Logger logger, String realm);
   public String getRealm ();
   public Logger getLogger ();
}
```

And LogEvent:

```
public class LogEvent extends EventObject {
   public LogEvent (LogSource source, String tag);
   ...
   ...
   public void addMessage (Object msg);
   ...
}
```

(please take a look at jPOS's javadoc or source code for a full description)

Here is a simple way to create a Logger:

```
Logger logger = new Logger();
logger.addListener (new SimpleLogListener (System.out));
```

Now you can easily attach that logger to any jPOS component implementing LogSource such as channels, packagers, multiplexers, etc. You can easily call:

```
component.setLogger (logger, "some-component-description");
```

You can use jPOS's logger subsystem to log events of your own. In those cases, you have to either implement LogSource or extend the SimpleLogSource class. Then you can write code like this:

```
LogEvent evt = new LogEvent (yourLogSource, "my-event");
evt.addMessage ("A String message");
evt.addMessage (anyLoggeableObject);
Logger.log (evt);
```

The Loggeable interface is a very simple way of letting an object render itself:

```
public interface Loggeable {
   public void dump (PrintStream p, String indent);
}
```

Most of jPOS's components already implement the Loggeable interface, but you can easily wrap any given object with a Loggeable class that holds the former object as its payload, e.g.:

```
```

```
package net.swini.util;

import java.io.PrintStream;
import org.jpos.util.Loggeable;

public abstract class LoggeableBase implements Loggeable {
    protected String toXML (String tag, String value, String indent) {
        StringBuffer sb = new StringBuffer (indent);
        sb.append ('<');
        sb.append (tag);
        sb.append ('>');
        sb.append (value);
        sb.append ("</");
        sb.append (tag);
        sb.append ('>');
        return sb.toString ();
    }
    public abstract void dump (PrintStream p, String indent);
}

package net.swini.util;

import java.io.PrintStream;
import net.jini.core.lookup.ServiceItem;
import net.jini.lookup.entry.ServiceInfo;

public class LoggeableServiceItem extends LoggeableBase {
    String tag;
    ServiceItem item;
    public LoggeableServiceItem (String tag, ServiceItem item) {
        super();
        this.tag  = tag;
        this.item = item;
    }
    public void dump (PrintStream p, String indent) {
        String inner = indent + "   ";
        p.println (indent + "<" + tag + ">");

        if (item.service != null) {
            p.println (toXML ("class", item.service.getClass().getName(), inner));
        } else {
            p.println (inner + "null item.service - (check http server)");
        }
        p.println (toXML ("id", item.serviceID.toString(), inner));

        for (int i=0 ; i<item.attributeSets.length ; i++) {
            if (item.attributeSets[i] instanceof ServiceInfo) {
                ServiceInfo info = (ServiceInfo) item.attributeSets[i];
                p.println (toXML ("name", info.name, inner));
                p.println (toXML ("manufacturer", info.manufacturer, inner));
                p.println (toXML ("vendor", info.vendor, inner));
                p.println (toXML ("version", info.version, inner));
                p.println (toXML ("model", info.model, inner));
                p.println (toXML ("serial", info.serialNumber, inner));
            }
            else {
                p.println (inner + "<attr>");
                p.println (inner + "  "+item.attributeSets[i].toString());
                p.println (inner + "</attr>");
            }
        }
        p.println (indent + "</" + tag + ">");
    }
}
```

There's a general purpose Loggeable class called SimpleMsg which has an overloaded constructor for several

commonly used Java types. You can easily add a SimpleMsg to your log stream with code like this:

```
...
...
evt.addMessage (new SimpleMsg ("demo", "boolean", true));
evt.addMessage (new SimpleMsg ("demo", "time", System.currentTimeMillis()));
evt.addMessage (new SimpleMsg ("demo", "dump", "TEST".getBytes()));
...
...
```

jPOS comes with several LogListener implementations and it's very easy to write your own. The ready available ones include:

Table 4.2. LogListener

Class	Description
SimpleLogListener	Dumps log events to a PrintStream (such as System.out)
RotateLogListener	Automatically rotate logs based on file size and time window
DailyLogListener	Automatically rotate logs daily. Has the ability to compress old log files
OperatorLogListener	Applies some filtering and e-mails log-events to an operator
Log4JListener	Forwards LogEvents to Log4J logger subsystem

Tip

In the jPOS-EE code base you can find some additional logger implementations such as IRCLogListener that forwards LogEVents to an irc channel.

LogListeners are called synchronously, so one listener has the chance to modify a given LogEvent; for example, ProtectedLogListener analyzes received LogEvents and **protects** important information (such as track-2 data).

4.2. NameRegistrar

org.jpos.util.NameRegistrar is a very simple **singleton** class that can be used to register and locate jPOS components. It's nothing but a simple, well-known Map where one can easily find components by an arbitrary name.

NameRegistrar has the following static methods:

```
public static void register (String key, Object value);
public static void unregister (String key);
public static Object get (String key)
    throws NameRegistrar.NotFoundException;
public static Object getIfExists (String key);
```

So you can write code like this:

```
...
...
ISOMUX mux = new ISOMUX (...);
NameRegistrar.register ("myMUX", mux);
...
...
```

and elsewhere in your application you can get a reference to your MUX with code like this:

```
try {
   ISOMUX mux = (ISOMUX) NameRegistrar.get ("myMUX");
} catch (NameRegistrar.NotFoundeException e) {
   ...
   ...
}
```

or

```
ISOMUX mux = (ISOMUX) NameRegistrar.getIfExists ("myMUX");
if (mux != null) {
    ...
    ...
}
```

Although we can use NameRegistrar in order to register jPOS components, sometimes it's better to use the component's setName(String name) method when available.

Most components have a setName (String name) method implemented like this:

```
public class ISOMUX {
    ...
    ...
    public void setName (String name) {
        this.name = name;
        NameRegistrar.register ("mux."+name, this);
    }
    ...
    ...
```

The prefix `"mux."` is used here in order to avoid a clash of names in the registrar between different classes of components using the same name (i.e. `"mux.institutionABC"` and `"channel.institutionABC"`).

Different components use different prefixes as shown in the following table:

Table 4.3. NameRegistrar's prefix

Component	Prefix	Getter
ConnectionPool	"connection.pool."	N/A
ControlPanel	"panel."	N/A
DirPoll	"qsp.dirpoll."	N/A
BaseChannel	"channel."	BaseChannel.getChannel

Component	Prefix	Getter
ISOMUX	"mux."	ISOMUX.getMUX
QMUX	"mux."	QMUX.getMUX
ISOServer	"server."	ISOServer.getServer
KeyStore	"keystore."	N/A
Logger	"logger."	Logger.getLogger
LogListener	"log-listener."	N/A
PersistentEngine	"persistent.engine."	N/A
SMAdapter	"s-m-adapter."	BaseSMAdapter.getSMAdapter

Tip

While we try to keep the previous prefix table up to date, we suggest that you double-check it against the source code if you have problems getting references to your components.

Using the getter (when available) lets us write code like this:

```
try {
   ISOMUX mux = ISOMUX.get ("myMUX");
} catch (NameRegistrar.NotFoundeException e) {
   ...
   ...
}
```

that will in turn call `NameRegistrar.get ("mux.myMUX")`. Later, we'll see that NameRegistrar is extensively used by jPOS' Q2 applications. Q2 takes care of configuring several jPOS components for you, but your code will have to locate them by a given name. That's where `NameRegistrar` comes in to play.

Tip

The `NameRegistrar` is a `Loggeable` object (see Section 4.1, “jPOS' Logger”) so its instance (`NameRegistrar.getInstance()`) can be added to a `LogEvent` in order to assist you during debugging sessions.

When running in a **Q2** environment we recommend to deploy a `sysmon` service in order to regularly view the NameRegistrar's content.

4.3. Configuration

Configuration, Configurable, ReConfigurable

org.jpos.core.Configuration is a general purpose property container extensively used by jPOS components.

The Configuration interface looks like this:

```
package org.jpos.core;

public interface Configuration {
  public void put (String name, Object value);
  public String get (String propertyName);
  public String get (String propertyName, String defaultValue);
  public String[] getAll  (String propertyName);
```

```
    public int[] getInts  (String propertyName);
    public long[] getLongs (String propertyName);
    public double[] getDoubles (String propertyName);
    public boolean[] getBooleans (String propertyName);
    public int getInt (String propertyName);
    public int getInt (String propertyName, int defaultValue);
    public long getLong (String propertyName);
    public long getLong (String propertyName, long defaultValue);
    public double getDouble (String propertyName);
    public double getDouble (String propertyName, double defaultValue);
    public boolean getBoolean (String propertyName);
    public boolean getBoolean (String propertyName, boolean defaultValue);
  }
```

Having our own Configuration interface lets us implement it in different ways. We have a very little class called SimpleConfiguration backed by a java.util.Properties, but nothing prevents us from creating a more sophisticated Configuration object capable of providing dynamic data (such as an SQLConfiguration, JavaSpacesConfiguration and the like).

We also have a very simple interface called Configurable:

```
package org.jpos.core;

public interface Configuration {
  public void setConfiguration (Configuration cfg)
     throws ConfigurationException;
}
```

Later, while looking at the Q2 application we'll see that Q2 pushes a configuration object in `[Re]Configurable` objects by calling its `setConfiguration` method.

```
<object name="myObject" class="com.mycompany.MyObject">
 <property name="myProperty" value="any Value" />
</object>
```

Should `com.mycompany.MyObject"` implement `Configurable`, Q2 would call its `setConfiguration()` method providing access to the underlying `myProperty` property.

It's interesting to note that Q2 provides the ability to have array of properties under the same name, i.e:

```
<object name="myObject" class="com.mycompany.MyObject">
 <property name="myProperty" value="Value A" />
 <property name="myProperty" value="Value B" />
 <property name="myProperty" value="Value C" />
</object>
```

where one can call handy methods like `String[] getAll(String)`.

`setConfiguration(Configuration cfg)` can check the Configuration object and might throw a ConfigurationException in case a required property is not present or is invalid.

4.4. SystemMonitor

org.jpos.util.SystemMonitor is a very simple class that periodically logs useful information such as the number of running threads, memory usage, etc.

Its constructor looks like this:

```
    public SystemMonitor (int sleepTime, Logger logger, String realm)
```

Note

See javadocs [http://jpos.org/doc/javadoc/org/jpos/util/SystemMonitor.html] for details.

Using SystemMonitor is very easy. You simply have to instantiate it with code like this:

```
    ...
    ...
    new SystemMonitor (60*60*1000, yourLogger, "system-monitor"); // dumps every hour
    ...
    ...
```

and it will dump info to your log every hour (60*60*1000 milliseconds).

The output looks like this:

```
<log realm="org.jpos.q2.qbean.SystemMonitor" at="Fri Nov 16 17:43:13 GMT-03:00 2007.471">
  <info>
    <memory>
       freeMemory=8634840
      totalMemory=19652608
      inUseMemory=11017768
    </memory>
    <threads>
            delay=0 ms
          threads=29
            Thread[Reference Handler,10,system]
            Thread[Finalizer,8,system]
            Thread[Signal Dispatcher,10,system]
            Thread[main,5,main]
            Thread[Timer-0,5,main]
            Thread[txnmgr-0,5,main]
            Thread[txnmgr-1,5,main]
            Thread[Thread-5,5,main]
            Thread[channel-sender-selftest-send,5,main]
            Thread[channel-receiver-selftest-receive,5,main]
            Thread[Thread-8,5,main]
            Thread[scanner,5,main]
            Thread[btpool0-0,5,main]
            Thread[btpool0-1,5,main]
            Thread[btpool0-2,5,main]
            Thread[btpool0-3,5,main]
            Thread[btpool0-4,5,main]
            Thread[btpool0-5,5,main]
            Thread[btpool0-6,5,main]
            Thread[btpool0-7,5,main]
            Thread[btpool0-8,5,main]
            Thread[btpool0-9 - Invalidator - /jposee,5,main]
            Thread[Store default Spool Thread,2,main]
            Thread[Store default Expiry Thread,1,main]
            Thread[Timer-1,5,main]
            Thread[com.mchange.v2.async.ThreadPoolAsynchronousRunner$PoolThread-#0,5,main]
            Thread[com.mchange.v2.async.ThreadPoolAsynchronousRunner$PoolThread-#1,5,main]
            Thread[com.mchange.v2.async.ThreadPoolAsynchronousRunner$PoolThread-#2,5,main]
            Thread[Thread-23,5,main]
            Thread[Rollover,5,main]
            Thread[SystemMonitor,5,main]
            Thread[PooledThread-0-running,5,ThreadPool-0]
    </threads>
```

```
    --- name-registrar ---
      tspace:org.jpos.transaction.TransactionManager@12a3793: org.jpos.space.TSpace
        <key>$TAIL</key>
        <keycount>1</keycount>
      mux.selftest-mux: org.jpos.q2.iso.QMUX
        tx=17, rx=16, tx_expired=0, tx_pending=0, rx_expired=0, rx_pending=1, rx_unhandled=0, rx_forwa
      server.jcard-server: org.jpos.iso.ISOServer
        connected=1, rx=17, tx=16
          127.0.0.1:33170: rx=17, tx=16
      logger.: org.jpos.util.Logger
      tspace:default: org.jpos.space.TSpace
        <key>selftest-send.2100000000002911000010000000000014.req</key>
        <key>$TAILLOCK.19543955</key>
        <keycount>2</keycount>
      jdbm:simulator:log/simulator: org.jpos.space.JDBMSpace
      txnmgr: org.jpos.transaction.TransactionManager
      logger.Q2: org.jpos.util.Logger
      capture-date: org.jpos.ee.CaptureDate
      channel.selftest-adaptor: org.jpos.iso.channel.CSChannel
      selftest-adaptor: org.jpos.q2.iso.ChannelAdaptor
        tx=16, rx=16, connects=1
  </info>
</log>
```

Tip

If you're using Q2 (revision greater than r2581), the default configuration deploys a `SystemMonitor` for you. See `deploy/99_sysmon.xml`.

4.5. Profiler

org.jpos.util.Profiler is a very simple and easy to use user-space Profiler. It leverages the Logger subsystem to provide accurate information about processing times.

These are Profiler's public methods:

```
    public void reset();
    public void checkPoint (String detail);
    public long getElapsed();
    public long getParcial();
```

See javadocs [http://www.jpos.org/doc/javadoc/org/jpos/util/Profiler.html] for details.

Profiler implements Loggeable, so you can easily add a Profiler Object to a LogEvent to produce convenient profiling information.

Example 4.1. Profiler

```
    Profiler prof = new Profiler();
    LogEvent evt = new LogEvent (this, "any-transaction", prof);

    // initialize message
    ISOMsg m = new ISOMsg ();
    m.setMTI ("1200");
    ...
    ...
    prof.checkPoint ("initialization");
```

```
// send message to remote host
...
...
ISORequest req = new ISORequest (m);
mux.queue (req);
ISOMsg response = req.getResponse (60000);
prof.checkPoint ("authorization");

// capture data in local database
...
...
prof.checkPoint ("capture");
...
...
Logger.log (evt);
```

The previous example would produce output like this:

```
<log realm="programmers-guide" at="Fri Jul 26 17:55:01 UYT 2002.178">
  <any-transaction>
    <profiler>
      initialization [2]
      authorization [167]
      capture [39]
      end [213]
    </profiler>
  </any-transaction>
</log>
```

The values inside the square brackets are expressed in milliseconds; with this info in hand, we can easily find out where the time is spent in a given transaction.

Tip

The "end" checkPoint is automatically computed at output time (that's when Logger calls its log listeners).

4.6. DirPoll

Many jPOS-based applications have to interact with third-party legacy software (e.g., retail applications). Most of the time one can be lucky enough to deal with legacy applications capable of sending transactions over decent protocols (such as native TCP/IP, RMI, SOAP, XML-RPC, etc.), but sometimes you are not that lucky and the best thing you can get is a disk-based interchange, i.e., they place a request in a given directory, you process that request and provide a response.

org.jpos.util.DirPoll uses the following directory structure:

```
....request
....response
....tmp
....run
....bad
```

and defines the following inner interfaces:

```
```

```
public interface Processor {
    public byte[] process(String name, byte[] request)
        throws DirPollException;
}
public interface FileProcessor {
    public void process (File name) throws DirPollException;
}
```

You can either create a Processor or a FileProcessor to handle incoming traffic.

Whenever a legacy application places a file in the `request` directory, your Processor (or FileProcessor) gets called, giving you a chance to process the given request and provide a response (if you're using a Processor, the response will be placed in the `response` directory).

Example 4.2. DirPoll Processor

```
public class DirPollProcessor implements DirPoll.Processor {
    DirPollProcessor () {
        super ();
        DirPoll dp = new DirPoll ();
        dp.setLogger (logger, "dir-poll");
        db.setPath ("/tmp/dirpoll");
        db.createDirs ();
        db.setProcessor (this);
        new Thread (dp).start ();
    }
    public byte[] process (String name, byte[] b) {
        return ("request: " + name + " content="+ new String (b)).getBytes();
    }
}
```

DirPoll has provisions to handle different kind of messages with different priority based on its file extension, so you can call:

```
...
...
dp.addPriority (".A");
dp.addPriority (".B");
dp.addPriority (".C");
...
...
```

in order to raise ".A" priority over ".B" and ".C" requests (you can use any extension name).

Before processing a given request, `DirPoll` moves it to the `run` directory, and then either to the `response` directory or to the `bad` directory (in case something goes wrong and a DirPollException has been thrown).

Note

If your application crashes, you have to take care of possible requests left sitting in the `run` directory.

It is very important that your application writes the requests in the `tmp` directory (or any other temporary directory in the same filesystem) and then moves them (after a proper operating system close operation) to the `request` directory in order to guarantee that once a request is present in the `request` directory, it is ready for DirPoll to process.

Tip

Don't trust your legacy application programmer. Please double check that the previous note has been taken into account.

4.7. ThreadPool

org.jpos.util.ThreadPool, as its name implies, takes care of managing a pool of threads.

Its constructor looks like this:

```
public ThreadPool (int initialPoolSize, int maxPoolSize)
```

(See javadocs [http://jpos.org/doc/javadoc/org/jpos/util/ThreadPool.html] for details).

It's very useful to process short-lived threads, such as processing an authorization transaction. Instead of creating a new thread per transaction, you can create a ThreadPool at initialization time and then call its `execute(Runnable r)` method.

The thread will be returned to the pool when your `run()` method ends, so it is not a good idea to have long-running items (e.g., a `for (;;) { ... }` loop) in your Runnable.

There's an inner interface called ThreadPool.Supervised that your Runnable can optionally implement:

```
public class ThreadPool {
    public interface Supervised {
        public boolean expired ();
    }
}
```

In this case, ThreadPool will call your `expired()` method, and - if true - will attempt to interrupt the expired thread. Note that while this does not guarantee that your thread will gracefully end, it gives you a chance to repair a possible problem.

Tip

You can write some 'self-healing' code in your `expired()` implementation, but please make sure your code won't block for too long. Use only if you know what you're doing.

ThreadPool implements ThreadPoolMBean, which exposes the following read-only properties:

```
public int getJobCount ();
public int getPoolSize ();
public int getMaxPoolSize ();
public int getIdleCount();
public int getPendingCount ();
```

Chapter 5. Implementing Custom Packagers

jPOS comes with several ISOPackager and ISOFieldPackager implementations which can be used either out-of-the-box or as a reference to encode (pack) and decode (unpack) messages that are built on the ISO-8583 standard.

Tip

For a list of out-of-the-box packagers you may want to have a look at the following directories:

- `modules/jpos/src/org/jpos/iso/packager` (Java based packagers)
- `src/config/packager` (GenericPackager configurations, see Section 5.1, “GenericPackager”)

Although not required, most ISOPackager implementations extend the supporting class ISOBasePackager. This approach makes writting a custom packager a very simple task. It's basically just a matter of calling its `public void setFieldPackager (ISOFieldPackager[] fld)` method with a suitable array of ISOFieldPackagers.

Let's look at a sample implementation:

Example 5.1. ISO-8583 version 1993 packager implementation

```
public class ISO93BPackager extends ISOBasePackager {
    private static final boolean pad = false;
    protected ISOFieldPackager fld[] = {
    /*000*/ new IFB_NUMERIC (  4, "Message Type Indicator", pad),
    /*001*/ new IFB_BITMAP  ( 16, "Bitmap"),
    /*002*/ new IFB_LLNUM   ( 19, "Primary Account number", pad),
    /*003*/ new IFB_NUMERIC (  6, "Processing Code", pad),
    /*004*/ new IFB_NUMERIC ( 12, "Amount, Transaction", pad),
    /*005*/ new IFB_NUMERIC ( 12, "Amount, Reconciliation", pad),
    /*006*/ new IFB_NUMERIC ( 12, "Amount, Cardholder billing", pad),
    /*007*/ new IFB_NUMERIC ( 10, "Date and time, transmission", pad),
    /*008*/ new IFB_NUMERIC (  8, "Amount, Cardholder billing fee", pad),
    /*009*/ new IFB_NUMERIC (  8, "Conversion rate, Reconciliation", pad),
    /*010*/ new IFB_NUMERIC (  8, "Conversion rate, Cardholder billing", pad),
    ...
    ...
    ...
    /*123*/ new IFB_LLLCHAR (999, "Reserved for private use"),
    /*124*/ new IFB_LLLCHAR (999, "Reserved for private use"),
    /*125*/ new IFB_LLLCHAR (999, "Reserved for private use"),
    /*126*/ new IFB_LLLCHAR (999, "Reserved for private use"),
    /*127*/ new IFB_LLLCHAR (999, "Reserved for private use"),
    /*128*/ new IFB_BINARY  (  8, "Message authentication code field")
    };
    public ISO93BPackager() {
        super();
        setFieldPackager(fld);
    }
}
```

Note

[For a complete listing please see `modules/jpos/src/org/jpos/iso/packager/ISO93BPackager.java`]

We hope you see the key idea: writing a custom packager involves diving into your interchange specification and setting up a suitable kind of field packager for every possible field. [1]

5.1. GenericPackager

After writing multiple ISOFieldPackager implementations, jPOS developer Eoin Flood came up with a nice idea: writing a GenericPackager that would read an XML configuration file and instantiate an ISOFieldPackager on-the-fly.

Using this approach, the same packager we've seen in the previous example can be easily configured using `GenericPackager` and a simple XML file like this:

Example 5.2. ISO-8583 version 1993 packager configuration

```
<!DOCTYPE isopackager SYSTEM "genericpackager.dtd">

<!-- ISO 8583:1993 (BINARY) field descriptions for GenericPackager -->

<isopackager>
  <isofield
      id="0"
      length="4"
      name="Message Type Indicator"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="1"
      length="16"
      name="Bitmap"
      class="org.jpos.iso.IFB_BITMAP"/>
  <isofield
      id="2"
      length="19"
      name="Primary Account number"
      pad="false"
      class="org.jpos.iso.IFB_LLNUM"/>
  <isofield
      id="3"
      length="6"
      name="Processing Code"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="4"
      length="12"
      name="Amount, Transaction"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="5"
      length="12"
      name="Amount, Reconciliation"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="6"
```

[1] See appendix B (ISOFieldPackagers)

```
      length="12"
      name="Amount, Cardholder billing"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="7"
      length="10"
      name="Date and time, transmission"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="8"
      length="8"
      name="Amount, Cardholder billing fee"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="9"
      length="8"
      name="Conversion rate, Reconciliation"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="10"
      length="8"
      name="Conversion rate, Cardholder billing"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
   ...
   ...
   ...
  <isofield
      id="123"
      length="999"
      name="Reserved for private use"
      class="org.jpos.iso.IFB_LLLCHAR"/>
  <isofield
      id="124"
      length="999"
      name="Reserved for private use"
      class="org.jpos.iso.IFB_LLLCHAR"/>
  <isofield
      id="125"
      length="999"
      name="Reserved for private use"
      class="org.jpos.iso.IFB_LLLCHAR"/>
  <isofield
      id="126"
      length="999"
      name="Reserved for private use"
      class="org.jpos.iso.IFB_LLLCHAR"/>
  <isofield
      id="127"
      length="999"
      name="Reserved for private use"
      class="org.jpos.iso.IFB_LLLCHAR"/>
  <isofield
      id="128"
      length="8"
      name="Message authentication code field"
      class="org.jpos.iso.IFB_BINARY"/>
</isopackager>
```

Note

[For a complete listing please see `modules/jpos/cfg/packager/iso93binary.xml`]

GenericPackager uses a DTD defined in `modules/jpos/cfg/packager/genericpackager.dtd` that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT isopackager (isofield+,isofieldpackager*)*>
<!ATTLIST isopackager maxValidField CDATA        #IMPLIED>
<!ATTLIST isopackager bitmapField   CDATA        #IMPLIED>
<!ATTLIST isopackager firstField    CDATA        #IMPLIED>
<!ATTLIST isopackager emitBitmap    (true|false) #IMPLIED>

<!-- isofield -->
<!ELEMENT isofield (#PCDATA)>
<!ATTLIST isofield id     CDATA        #REQUIRED>
<!ATTLIST isofield length CDATA        #REQUIRED>
<!ATTLIST isofield name   CDATA        #REQUIRED>
<!ATTLIST isofield class  NMTOKEN      #REQUIRED>
<!ATTLIST isofield token  CDATA        #IMPLIED>
<!ATTLIST isofield pad    (true|false) #IMPLIED>

<!-- isofieldpackager -->
<!ELEMENT isofieldpackager (isofield+,isofieldpackager*)*>
<!ATTLIST isofieldpackager id       CDATA        #REQUIRED>
<!ATTLIST isofieldpackager name     CDATA        #REQUIRED>
<!ATTLIST isofieldpackager length   CDATA        #REQUIRED>
<!ATTLIST isofieldpackager class    NMTOKEN      #REQUIRED>
<!ATTLIST isofieldpackager token    CDATA        #IMPLIED>
<!ATTLIST isofieldpackager pad      (true|false) #IMPLIED>
<!ATTLIST isofieldpackager packager NMTOKEN      #REQUIRED>
<!ATTLIST isofieldpackager emitBitmap (true|false) #IMPLIED>
<!ATTLIST isofieldpackager maxValidField CDATA        #IMPLIED>
<!ATTLIST isofieldpackager bitmapField CDATA        #IMPLIED>
<!ATTLIST isofieldpackager firstField  CDATA        #IMPLIED>
```

GenericPackager's DTD eases the configuration of nested messages (an ISO-8583 field that is a full ISO-8583 message itself), e.g.:

```
  ...
  ...
 <isofieldpackager
     id="127"
     length="255"
     name="FILE RECORS(S) ACTION/DATA"
     class="org.jpos.iso.IFB_LLHBINARY"
     packager="org.jpos.iso.packager.GenericSubFieldPackager">
     <isofield
         id="0"
         length="1"
         name="FILE UPDATE COD"
         class="org.jpos.iso.IFE_CHAR"/>
     <isofield
         id="1"
         length="19"
         name="ACCOUNT NUMBER"
         pad="true"
         class="org.jpos.iso.IFB_LLHNUM"/>
     <isofield
         id="2"
         length="4"
         name="PURGE DATE"
         pad="true"
         class="org.jpos.iso.IFB_NUMERIC"/>
     ...
     ...
```

```
    ...
</isofieldpackager>
```

Chapter 6. Channel Implementations

jPOS comes with several channel implementations, most of which are available in the `modules/jpos/src/org/jpos/iso/channel` directory.

6.1. TCP/IP Socket-based channels

Most TCP/IP-based channel implementations extend `org.jpos.iso.BaseChannel` and just override the `sendMessageLength` and `getMessageLength` methods.

Let's have a look at `org.jpos.iso.channel.CSChannel`: it uses a two-byte message length header sent in network byte order (nbo) plus two bytes reserved for future use:

```
public class CSChannel extends BaseChannel {
    ...
    ...
    protected void sendMessageLength(int len) throws IOException {
        serverOut.write (len >> 8);
        serverOut.write (len);
        serverOut.write (0);
        serverOut.write (0);
    }
    ...
    ...
    protected int getMessageLength() throws IOException, ISOException {
        int l = 0;
        byte[] b = new byte[4];
        while (l == 0) {
            serverIn.readFully(b,0,4);
            l = ((((int)b[0])&0xFF) << 8) | (((int)b[1])&0xFF);
            if (l == 0) {                                                    (1)
                serverOut.write(b);
                serverOut.flush();
            }
        }
        return l;
    }
}
```

(1) If message length is "0", CSChannel bounces it immediately. This procedure is used to keep firewalls from timing out

Here is a partial list of current channel implementations (for a complete list, have a look at `modules/jpos/src/org/jpos/iso/channel`):

Table 6.1.

Class name	Wire protocol
CSChannel	LL LL 00 00 [header] ISO-DATA LL LL represents the [header+] ISO-DATA length in network byte order 00 00 reserved for future use

Class name	Wire protocol
	Header is optional ISO-DATA: ISO-8583 image
NACChannel	LL LL [TPDU] ISO-DATA LL LL represents the TPDU+ISO-DATA length in network byte order Optional TPDU (transport protocol data unit) ISO-DATA: ISO-8583 image
NCCChannel	LL LL [TPDU] ISO-DATA LL LL represents the TPDU+ISO-DATA length in BCD (binary coded decimal) Optional TPDU (transport protocol data unit) ISO-DATA: ISO-8583 image
ASCIIChannel	LLLL [header] ISO-DATA LLLL four bytes ASCII [header+] ISO-DATA length Optional header ISO-DATA: ISO-8583 image
RawChannel	LL LL LL LL [header] ISO-DATA LL LL LL LL is [header+] ISO-DATA length in network byte order ISO-DATA: ISO-8583 image
VAPChannel	LL LL 00 00 header ISO-DATA LL LL represents the header+ISO-DATA length in network byte order 00 00 reserved for future use VAP-specific header ISO-DATA: ISO-8583 image
PADChannel	[header] ISO-DATA Stream-based channel reads messages on-the-fly without using any kind of message boundary indicator.
X25Channel	X25 is similar to PADChannel but uses a slightly different strategy. Instead of pulling an ISO-8583 from a stream, unpacking it on the fly, X25Channel attempts to read full TCP/IP packets by specifying a small timeout value. Whenever possible, PADChannel seems like a better solution; however, certain X.25 packet assembler/disassemblers sometimes send garbage over the wire (i.e. ETXs) which might confuse PADChannel.

Class name	Wire protocol
XMLChannel	Send/Receive messages in jPOS's internal XML message representation
LogChannel	Similar to XMLChannel, but you can feed it a jPOS Log, which is suitable to replay sessions

6.2. LoopbackChannel

Loopback channel bounces all received messages using a blocking queue. It can be used for simulation purposes. When using in combination with a suitable ISOFilter, you can modify the outgoing message so it can easily simulate a response from an dummy remote host.

```
package loopback;

import java.io.IOException;
import org.jpos.iso.ISOMsg;
import org.jpos.iso.ISOFilter;
import org.jpos.iso.ISOChannel;
import org.jpos.iso.ISOException;
import org.jpos.iso.channel.LoopbackChannel;
import org.jpos.util.LogEvent;

public class Test implements ISOFilter {
    public static void main (String[] args) {
        try {
            new Test().run();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void run () throws ISOException, IOException {
        LoopbackChannel channel = new LoopbackChannel ();
        channel.addIncomingFilter (this);                                    (1)
        ISOMsg request  = createRequest();
        request.dump (System.out, "request>  ");
        channel.send (request);
        ISOMsg response = channel.receive();
        response.dump (System.out, "response> ");
    }

    private ISOMsg createRequest () throws ISOException {
        ISOMsg m = new ISOMsg ("0800");
        m.set (11, "000001");
        m.set (41, "29110001");
        m.set (70, "301");
        return m;
    }
    public ISOMsg filter (ISOChannel channel, ISOMsg m, LogEvent evt) {      (2)
        try {
            m.setResponseMTI ();
            m.set (39, "00");
        } catch (ISOException e) {
            e.printStackTrace();
        }
        return m;
    }
}
```

(1) All incoming messages will go through our custom filter

(2) Filter implementation

The previous program produces the following output:

```
request>  <isomsg>
request>    <field id="0" value="0800"/>
request>    <field id="11" value="000001"/>
request>    <field id="41" value="29110001"/>
request>    <field id="70" value="301"/>
request>  </isomsg>
response> <isomsg direction="incoming">
response>   <field id="0" value="0810"/>
response>   <field id="11" value="000001"/>
response>   <field id="39" value="00"/>
response>   <field id="41" value="29110001"/>
response>   <field id="70" value="301"/>
response> </isomsg>
```

Tip

For a better way to simulate a remote host, you can have a look at the **serversimulator** module in the jPOS-EE distribution.

6.3. ChannelPool

ChannelPool is an ISOChannel implementation that delegates channel operations to its children channels.

It can handle several children channels, making it suitable to implement transparent failover.

By using its `addChannel` and `removeChannel` methods, you can react to network problems on-the-fly without affecting higher-level layers of your application.

Tip

As an alternative to the `ChannelPool`, Q2 applications can use multiple `ChannelAdaptors` configured with the same set of Space queues (in/out).

In addition, there's a `MUXPool` that provides failover as well as round-robin load balancing.

6.4. SSL

SocketFactories

`ISOServer`, as well as most channels that inherit from `BaseChannel` can delegate socket creation to an optional socket factory.

We have two kinds of socket factories:

- `ISOClientSocketFactory`
- `ISOServerSocketFactory`

```
public interface ISOClientSocketFactory {
    public Socket createSocket(String host, int port)
        throws IOException, ISOException;
```

```
}

public interface ISOServerSocketFactory {
    public ServerSocket createServerSocket(int port)
        throws IOException, ISOException;
}
```

...as well as a provider that implements both of them: `org.jpos.iso.SunJSSESocketFactory`

Note

SunJSSESocketFactory is available under the 'src/ext' module.

Q2 (actually the ChannelAdaptor and QServer qbeans), accepts an optional 'socketFactory' property in the channel configuration, e.g.:

Example 6.1. SocketFactory configuration

```
<channel-adaptor name='sslclient'
    class="org.jpos.q2.iso.ChannelAdaptor" logger="Q2">
 <channel class="org.jpos.iso.channel.NACChannel" logger="Q2"
       packager="org.jpos.iso.packager.ISO87BPackager">

  <property name="host" value="127.0.0.1" />
  <property name="port" value="10000" />
  <property name="timeout" value="360000" />
  <property name="socketFactory" value="org.jpos.iso.SunJSSESocketFactory" />
 </channel>
 <in>sslsend</in>
 <out>sslreceive</out>
 <reconnect-delay>10000</reconnect-delay>
</channel-adaptor>
```

Tip

While SunJSSESocketFactory can be used to demonstrate SSL support in jPOS, production-grade installations should consider it just a reference/sample implementation. It uses ${user.home}/.keystore with a default password, so **at the very least** you want to override its `getPassword()` method.

Chapter 7. jPOS Space

7.1. Overview

jPOS's Space is a general-purpose coordination component inspired after **The Linda Coordination Language.** [1]

While jPOS's Space **is not** a Linda implementation, we highly recommend learning about **Linda** in order to better understand our Space component.

You can think about jPOS's Space component as being like a Map where its entries are lists of objects and its operations are fully synchronized.

There are three basic operations:

- void out (Object key, Object value)

  Put an object into the space. If an object under the given key already exists, the object is queued at the end of a list.

- Object rd (Object key)

  Reads an object from the space. Block until one is present under the given key.

- Object in (Object key)

  Take the object off the queue. Block until the object under the given key is present.

In addition to those three basic operations, org.jpos.space.Space adds a few handy methods:

- void out (Object key, Object value, long timeout)

  Place an object into the space using an expiration timeout. The entry is automatically removed upon expiration.

- Object rd (Object key, long timeout)

  Wait a maximum of `timeout` milliseconds for a given entry; otherwise, return null.

- Object in (Object key, long timeout)

  Wait a maximum of `timeout` milliseconds for a given entry; otherwise, return null.

- Object rdp (Object key)

  Read an entry if it exists.

- Object inp (Object key)

  Take an entry if it exists.

[1] See http://www.cs.yale.edu/Linda/linda-lang.html

- void push (Object key, Object value)

  Same as `out` but the entry is placed at the head of the queue (like a Stack).

- void push (Object key, Object value, long timeout)

  Same as the previous `push` operation with a timeout.

jPOS Space interface:

```
public interface Space {
    public void out (Object key, Object value);
    public void out (Object key, Object value, long timeout);
    public Object in  (Object key);
    public Object rd  (Object key);
    public Object in  (Object key, long timeout);
    public Object rd  (Object key, long timeout);
    public Object inp (Object key);
    public Object rdp (Object key);
    public void push (Object key, Object value);
    public void push (Object key, Object value, long timeout);
}
```

jPOS provides several Space implementations:

- TSpace

  in-memory space implementation.

- JDBMSpace

  Persistent space implementation - uses JDBM [2] for persistence.

- SpaceProxy

  An RMI-based space proxy

- SpaceInterceptor

  SpaceInterceptor can be used to intercept calls to the Space without having to extend a particular space implementation

- SpaceLets

  A SpaceLet is a QBean that combines a space factory and a space interceptor with scripting capabilities. [3]

Note
VERY IMPORTANT: All Space operations may throw an 'SpaceError' unchecked exception.

7.2. SpaceFactory

Although most Space implementations have either public constructors or factory methods that can be used to

[2] JDBM project: See http://sf.net/projects/jdbm
[3] See Chapter 8, *Q2 - second generation QSP* for details

create instances of their respective classes, we highly recommend using the `SpaceFactory` as the entry point for space creation or to obtain references to spaces that were previously created.

Example 7.1. Using the SpaceFactory

```
import org.jpos.space.Space;
import org.jpos.space.SpaceFactory;

Space sp = SpaceFactory.getSpace();
```

The previous example returns a reference to the default space, which happens to be a TSpace implementation registered with the name `default`. It's the same as calling:

```
Space sp = SpaceFactory.getSpace("tspace");
```

...which is also the same as calling:

```
Space sp = SpaceFactory.getSpace("tspace:default");
```

SpaceFactory decodes a space `URI` based on the space implementation type, followed by an optional name and optional parameter(s): `spacetype[:spacename[:spaceparam}}`

Table 7.1. Space URI

Type	Implementation
tspace	Creates or returns a reference to a previously-created instance of `TSpace`
jdbm	Creates or returns a reference to a previously-created instance of `JDBMSpace`. This URI accepts an optional parameter (after the Space name) which is a path to the persistent store, e.g., `jdbm:myspace:/tmp/myspace`.
spacelet	Returns a reference to a previously-created instance of `SpaceLet`

7.3. Using the space

jPOS's Space can be used to coordinate processes and to manage access to resources. We present here a couple of examples:

Example 7.2. Controlling system throughput

Imagine a situation where you want no more than 10 transactions to be processed simultaneously at any given

time. There are several ways to do this using standard programming techniques, but let's look at how we'd implement this feature using jPOS's Space. We'll also discuss the advantages of the Space approach. At startup time:

```
import org.jpos.space.Space;
import org.jpos.space.SpaceFactory;
public static final String PROCESS_TICKET = "PROCESS_TICKET";

Space sp = SpaceFactory.getSpace();
for (int i=0; i<10; i++) {
    sp.out (PROCESS_TICKET, new Object ());
}
```

Then, at process time:

```
Object ticket = null;
try {
   ticket = sp.in (PROCESS_TICKET); // would block until a ticket is available
   //
   // process transaction
   //
} finally {
   sp.out (PROCESS_TICKET, ticket);
}
```

The above example could be easily written using standard programming techniques, but using the Space approach has some advantages; for example, you can use a `SpaceProxy` to implement a centralized throughput controller, or you could use the `ReplicatedSpace` [4] to implement a cluster-wide controller.

Example 7.3. Coordinating clients and servers

Coordinating clients and servers using the Space is very simple: the client just places a request object in the space under a given queue name and the server pulls these objects out of it.

If you have multiple clients and multiple servers, you have to agree on a naming convention so the server can ensure it is responding to the correct client thread.

In order to `connect` a given instance of a client with a given instance of a server, we have found it very useful to use 'little spaces' to wrap the real request objects so that responses are sent back by the server using the same little space.

`TinySpace` is a lightweight space implementation suitable for just this kind of usage. We can have client code like this:

```
public ISOMsg process (ISOMsg m, long timeout) {
   Space sp = SpaceFactory.getSpace ("tspace:myspace");
   TinySpace ts = new TinySpace ();
   ts.out ("Request", m);
   sp.out ("QUEUE", ts);
   return (ISOMsg) ts.in ("Response", timeout);
}
```

[4] currently available in the jPOS-EE distribution.

...with server code like this:

```
Space sp = SpaceFactory.getSpace ("tspace:myspace");
for (;;) {
   Space ts = (Space) sp.in ("QUEUE");
   ISOMsg m = (ISOMsg) ts.in ("Request");
   ...
   ... process request, prepare response
   ...
   ts.out ("Response", m);
}
```

Note

jPOS does extensive use of the Spaces in order to provide inter-component communications. Objects such as ISOMsgs are moved along the system from one component to another one using Spaces.

7.4. SpaceTap

The SpaceTap is a SpaceListener that can be used to monitor a given LocalSpace for new entries under a given key.

Once a SpaceTap is created, it register itself as a listener in the source LocalSpace and copies all new entries to a destination space.

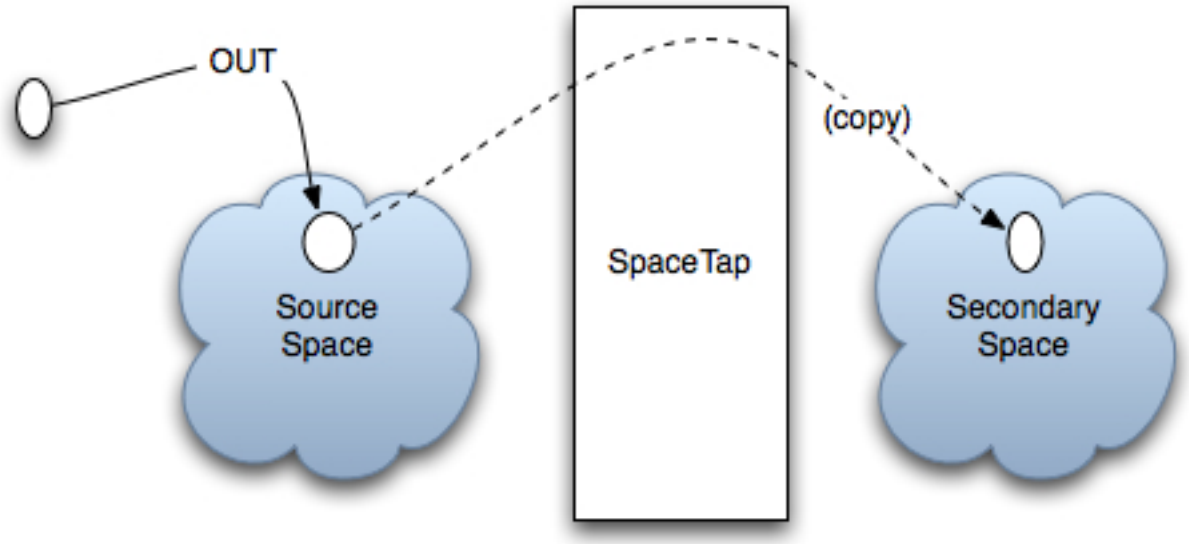


Figure 7.1. Space Tap

If we have a source LocalSpace `ssp` and a destination LocalSpace `dsp` and we want to monitor an entry called "ERRORS", we can use code like this:

```
SpaceTap spt = new SpaceTap (ssp, dsp, "ERRORS", "ERRORS.COPY", 5000L);
```

Tip

If your "source" space and "destination" space are the same, you can use the shorter constructor:

```
SpaceTap (LocalSpace ssp, Object key, Object tapKey, long timeout);
```

A SpaceTap can be used for system monitoring purposes as it provides a non-intrusive way to "tap" any given space queue.

Chapter 8. Q2 - second generation QSP

8.1. About Q2

After deploying QSP (see Chapter 13, *Legacy components*) in several mission-critical installations, we found it inconvenient to include all the components in a single [huge] configuration file for many reasons:

- Although several QSP components support some limited ReConfiguration, many others don't. As a result, major changes usually involve restarting the application (a very costly operation in a 24/7 system)
- If for some reason, the changes involved go beyond just tweaking a configuration file and require additional changes in a supporting jar file, the application has to be restarted (QSP doesn't support dynamic classloading).
- Having a single big configuration file has proven to be error-prone. Although initially intended to be accessible to system operators, changing QSP files on critical systems became an **art** reserved for experienced COOs ...

Therefore, we've decided to use a simpler approach: a file per component and a very simple lifecycle to ease the implementation of such components, called **QBeans** (Q2 Beans).

> **Note**
> We use the terms **QBeans** and **Q2 service** interchangeable.

QBeans are MBeans (see JMX specs) that implement the Q2's lifecycle (init/start/stop/destroy) set of operations. Q2 takes care of registering them with an MBeanServer.

8.2. Building Q2

Q2 is just another jPOS module (see `modules/q2`) so unless you explicitly ignore it (by adding `q2/**` to the `modules/ignore.list` file) it will get compiled as part of the standard build. See Section 1.5, “Building jPOS” for details.

8.3. Building Q2 Modules

The former `q2Modules` directory (used in versions prior to jPOS 6) with its own build system now sits in the `modules/q2mod_jpos` so unless you explicitly ignore it (by adding `q2mod_jpos/**` to the `modules/ignore.list` file) it will get compiled as part of the standard build. See Section 1.5, “Building jPOS” for details.

8.4. Q2 Primer

1. Running Q2

As of jPOS 6 and newer versions, running Q2 is basically running jPOS. Please review Section 2, “Running jPOS” for details.

If you add the '--help' parameter when starting jPOS (e.g: `java -jar jpos.jar --help`), you should see an output like this:

```
usage: Q2
 -c,--command <arg>     Command to execute                                  (1)
 -C,--config <arg>      Configuration bundle                                (2)
 -d,--deploydir <arg>   Deployment directory                                (3)
 -e,--encrypt <arg>     Encrypt configuration bundle                        (4)
 -h,--help              Usage information                                   (5)
 -i,--cli               Command Line Interface                              (6)
 -r,--recursive         Deploy subdirectories recursively                   (7)
 -v,--version           Q2's version                                        (8)
```

(1) a CLI command to execute (CLI is Q2's Command Line Interface).
(2) Uses a QSP-like single file configuration for a group of QBeans
(3) Lets you define an alternate deploy directory. The default is `deploy`
(4) It is possible to encrypt an MBean deployment descriptor in order to protect it from the ocasional lurker
(5) This help.
(6) Enables Command Line Interface (CLI). Type "help" for a list of installed commands.
(7) Recurse over directories inside the `deploy` directory (can be used to group a set of related QBeans).
(8) Displays Q2 version

We'll revisit these individual options, but for now, let's start Q2 by using the following command: `java -jar jpos.jar`

You should see an output like this:

```
<log realm="Q2.system" at="Wed May 26 09:15:18 UYT 2004.22">
  <info>
    Q2 started, deployDir=/home/jpos/q2/deploy
  </info>
</log>
```

At this point, your Q2 is running and checking the `deploy` directory for new QBean descriptors (XML files) as well as the `deploy/lib` directory for new `jars`.

Now, let's write a test QBean. The following QBean does very little but it is very useful to learn how Q2 works. QTest is a very simple QBean that just outputs a "tick" every second.

It is presented here in small fragments of code (with some parts omitted for clarity's sake). You can see a complete listing in Appendix E, *QTest and QTestMBean source code*.

```
package org.jpos.qtest;
public class QTest extends QBeanSupport implements Runnable, QTestMBean {
    long tickInterval = 1000;
```

`QTest` extends `QBeanSupport` which is a general purpose QBean implementation that provides handy functionality such as Logging, implementing the `Configurable` interface, etc. but you are actually not required to extend QBeanSupport as long as you implement the `org.jpos.q2.QBean` interface. QBeanSupport is located in the `q2` module in the `modules/q2/src/org/jpos/q2` directory.

In order to implement a QBean you basically need to provide four methods that implements the Q2 services li-

fecycle:

- **init**
- **start**
- **stop**
- **destroy**

`QBeanSupport` implements those methods for you providing a default implementation that in turn call four protected methods:

- **initService**
- **startService**
- **stopService**
- **destroyService**

as well as some handy methods that provide access to the QBean configuration (including its XML content), logging and even some helpers to write back the XML configuration.

In addition, the `initService`, `startService`, `stopService` and `destroyService` set of protected methods take care of catching and logging any possible exception that your implementation may throw and keeping track of the QBean's "state" (STARTING / STARTED / STOPPING / STOPPED / DESTROYED / FAILED) for you.

By extending QBeanSupport you don't have to implement all the QBean lifecycle callbacks, just the ones you need.

In our QTest case, we want our QBean to be a Runnable, so we just need to do something at "start" time, therefore we only have to override `startService`:

```
public void startService() {
    new Thread(this).start();
}
```

`startService` creates a new Thread passing an instance of your QTest class (which happens to be a Runnable) and starts it.

```
public void run () {
    for (int tickCount=0; running (); tickCount++) {
        log.info ("tick " + tickCount);
        ISOUtil.sleep (tickInterval);
    }
}
```

The run method is very simple, two things worth noting:

- We can use "log.info" here because QBeanSupport creates a Log (`org.jpos.util.Log`) object for you.

- the `running()` method is also provided by QBeanSupport and returns true if the service is either in the STARTING or STARTED state.

In order to demonstrate the lifecycle of a QBean, we override some methods such as its constructor as well as the `init()`, `start()`, `stop()` and `destroy()` methods just to log an info message, but you obviously don't need to do that in your application.

```
public QTest () {
    super();
    log.info ("constructor");
}
public void init () {
    log.info ("init");
    super.init ();
}
public void start() {
    log.info ("start");
    super.start ();
}
public void stop () {
    log.info ("stop");
    super.stop ();
}
public void destroy () {
    log.info ("destroy");
    log = null;
}
```

QBeans support two kind of configuration: the so-called IoC [1] where the container "pushes" the configuration onto the QBean and a regular "pull" method where you can query the Configuration object. *(technically the second method is still a "push" scheme as Q2 will "push" a Configuration Object onto the component at "init" time and the component can either hold a reference to the Configuration object or hold references to a subset of its properties)*

If you configure your QBean using **properties**, the configuration would look like this:

```
<qtest class="org.jpos.test.QTest" logger="Q2">
 <property name="mypropA" value="abc" />
 <property name="mypropB" value="123" />
</qtest>
```

So you can use code like this:

```
String a = cfg.get ("mypropA");
String b = cfg.get ("mypropB");
```

or you can override `void setConfiguration(Configuration)` and store your `a` and `b` Strings as instance variables.

On the other hand, if you want to use **attributes**, you have to provide setters and getters and you need to expose those setters and getters as part of your MBean interface.

In our QTest example we want to expose the "tickInterval" as an attribute so we have to provide the following methods:

[1] Inversion of Control

```
 public void setTickInterval (long tickInterval) {
     this.tickInterval = tickInterval;
 }
 public long getTickInterval () {
     return tickInterval;
 }
```

In addition we have to define an interface called QTestMBean that extends QBeanSupportMBean and expose those two methods as well:

```
public interface QTestMBean extends org.jpos.q2.QBeanSupportMBean {
  public void setTickInterval(long tickInterval) ;
  public long getTickInterval() ;
}
```

Note

JMX requires these *MBean interfaces in order to know which attributes and operations should be managed by JMX and available through the MBeanServer. Covering JMX is beyond the scope of this guide, it's a small yet powerful API that we strongly recommend you to read.

When using attributes, the QBean descriptor looks like this:

```
<qtest class="org.jpos.test.QTest" logger="Q2">
 <attr name="tickInterval" type="java.lang.Long">5000</attr>
</qtest>
```

Note

If you look at Appendix E, *QTest and QTestMBean source code* you'll see that we also provide a `void setPersist(Element)` and `Element getPersist()` operations.

These are extensions to the QBean provided by QBeanSupport that let a QBean return its own XML configuration so QBeanSupport can update the XML file in order to persist changed properties.

You are free to ignore these feature unless you have a strong use case for it. See `org.jpos.q2.QPersist` for further details.

Compiling QTest

You can compile QTest and QTestMBean using whatever way you like, however, we'll tell you here how we'd do it in a jPOSsy way, by means of a little `module`.

Let's call our module `qtest`, you want to create the following directory structure:

```
  modules/qtest/src/org/jpos/test/QTest.java
  modules/qtest/src/org/jpos/test/QTestMBean.java
  modules/qtest/deploy/10_qtest.xml
```

Once you call "ant", your *.java classes will be compiled into `build/jpos.jar` and your deployment descriptor will be copied to `build/deploy/10_qtest.xml` so the next time you start Q2, your QBean is going to be

launched.

Once you start jPOS/Q2 again you'd see:

```
<log realm="Q2.system" at="Wed May 26 09:50:59 UYT 2004.134">
  <info>
    deploy:qtest.xml
  </info>
</log>
<log realm="org.jpos.q2.example.QTest" at="Wed May 26 09:50:59 UYT 2004.290">
  <info>
    constructor
  </info>
</log>
<log realm="org.jpos.q2.example.QTest" at="Wed May 26 09:50:59 UYT 2004.296">
  <info>
    setName qtest
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:50:59 UYT 2004.302">
  <info>
    setPersist
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:50:59 UYT 2004.313">
  <info>
    init
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:50:59 UYT 2004.316">
  <info>
    start
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:50:59 UYT 2004.321">
  <info>
    tick 0
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:51:00 UYT 2004.343">
  <info>
    tick 1
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:51:01 UYT 2004.353">
  <info>
    tick 2
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:51:02 UYT 2004.363">
  <info>
    tick 3
  </info>
</log>
...
...
```

QTest is a verbose QBean, so it logs an output message when its constructor is called as well as when other lifecycle methods, such as init, start, stop and destroy are called. In our example, we also implement QPersist so we see the calls to `setPersist` that you are free to ignore.

Try to move QTest (`build/deploy/10_qtest.xml`) away from your deploy directory and it should stop running.

```
<log realm="Q2.system" at="Wed May 26 09:51:11 UYT 2004.883">
```

```
  <trace>
    undeploying:qtest.xml
  </trace>
</log>
<log realm="qtest" at="Wed May 26 09:51:11 UYT 2004.884">
  <info>
    stop
  </info>
</log>
<log realm="qtest" at="Wed May 26 09:51:11 UYT 2004.884">
  <info>
    destroy
  </info>
</log>
<log realm="Q2.system" at="Wed May 26 09:51:11 UYT 2004.885">
  <info>
    undeployed:qtest.xml
  </info>
</log>
```

Tip

Please note the `realm` of these log events: QTest-originated messages use the realm 'qtest' while Q2 uses 'Q2.system'.

You can use this facility to write a custom LogListener that can react to specific realms.

Our QTest bean supports a `tickInterval` attribute. So, while Q2 is still running, try to modify qtest.xml so it looks like this:

```
<qtest class="org.jpos.q2.example.QTest">
 <attr name="tickInterval" type="long">5000</attr>
</qtest>
```

Now, the tick shows every five seconds instead of every second.

QBeans are registered into an JMX MBeanServer, so we can change their parameters either by modifying the deployment descriptors, or by using a JMX to communicate with them. You can use `jconsole` in order to connect to the running JVM and modify the tickInterval on the fly.

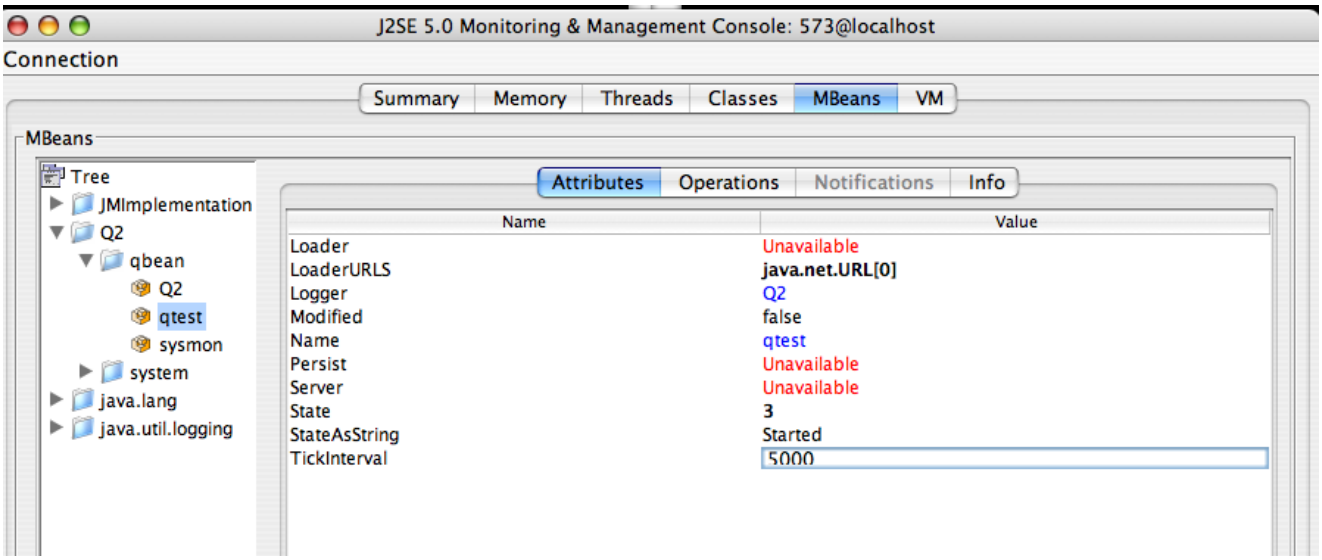


Figure 8.1. QTest in jConsole

Tip

In order to run `jconsole` you want to start Q2 using `-Dcom.sun.management.jmxremote`. See the `bin/q2` Unix script for details.

From there, you can change the tickInterval. Let's put a value of 20000 in there and check the results.

As you can see, QTest now 'ticks' every 20 seconds, but what's more important is that `deploy/10_qtest.xml` now looks like this:

```
<qtest class="org.jpos.q2.example.QTest" logger="Q2">
  <attr name="tickInterval" type="long">20000</attr>
</qtest>
```

It has been automatically persisted back to disk by Q2 (see QPersist javadocs for details)

2. QBean descriptors

The XML used to configure a QBean is extremely flexible and uses sensible defaults whenever possible.

The minimum QBean configuration has an element name with a class attribute:

```
  <myqbean class="org.project.MyClass" />
```

Your `org.project.MyClass` class is supposed to implement the interface `org.jpos.q2.QBean` which can be easily achieved by extending `org.jpos.q2.QBeanSupport` as previously described.

In the previous sample QBean configuration, Q2 would attempt to instantiate your class and register it with the MBeanServer (see JMX specs) using the name "Q2:type=qbean,service=myqbean".

In addition, if your class happens to have a method with a signature `void setName (String)`, then Q2 would register the instance with the local NameRegistrar (see Section 4.2, “NameRegistrar”).

The "name" used both to register the QBean with the MBeanServer and eventually with the NameRegistrar can be taken either from the element name or from an explicit `name` attribute, e.g:

```
  <any-element-name name="myqbean" class="org.project.MyClass" />
```

If we don't specify a classname, Q2 tries to get a suitable class based on the element name using an internal resource bundle called `QFactory.properties` (located in the `modules/q2/src/org/jpos/q2` directory).

As of the release r2584 (Dec/2007) `QFactory.properties` looks like this:

```
  logger=org.jpos.q2.qbean.LoggerAdaptor
  shutdown=org.jpos.q2.qbean.Shutdown
  script=org.jpos.q2.qbean.BSH
  jython=org.jpos.q2.qbean.Jython
  spacelet=org.jpos.q2.qbean.SpaceLet
  sysmon=org.jpos.q2.qbean.SystemMonitor
  txnmgr=org.jpos.transaction.TransactionManager
  transaction-manager=org.jpos.transaction.TransactionManager
  qmux=org.jpos.q2.iso.QMUX
```

```
channel-adaptor=org.jpos.q2.iso.ChannelAdaptor
```

This means that when you configure a QBean using one of the aforementioned element names, you don't need (you still can if you want) to specify its class name, e.g.:

```
<logger name="Q2">
  ...
  ...
</logger>
```

would be basically the same as

```
<logger name="Q2" class="org.jpos.q2.qbean.LoggerAdaptor">
  ...
  ...
</logger>
```

Note

At this point you might see why we need a special `name` attribute, if we have many loggers and we don't want to specify its class name, we want to use the shortcut `<logger ... >`, but we still need a way to differentiate them by using different names.

The Shutdown QBean

Based on the previous QFactory.properties based mapping we could deploy a file `shutdown.xml` like this:

```
<shutdown />
```

Note

The name `shutdown.xml` could be any other name you want.

The shutdown QBean is implemented like this:

```
package org.jpos.q2.qbean;

import org.jpos.q2.QBeanSupport;

public class Shutdown extends QBeanSupport {
    public void startService() {
        getServer().shutdown ();
    }
}
```

By deploying the `shutdown` QBean you have a clean way to stop a given Q2 instance.

Tip

If you are a using jPOS-EE, you'd see that `bin/stop` deploys a shutdown QBean while the `bin/q2` script makes sure it is not present in the deploy directory at startup time.

In addition to the `class` attribute, Q2 supports two additional attributes:

- **`logger`**

  If the the QBean is an instance of the `LogSource` interface (which is the case for all QBean implementations that extend `QBeanSupport`), a logger object is pushed to the QBean by calling its `void setLogger (Logger logger, String realm)` method.

- **`realm`**

  If a `realm` attribute is present, then it will get passed in the call to `setLogger`, otherwise the QBean's name would be used as the realm.

The XML configuration file can have any format up to the point that a QBean can implement the `org.jpos.core.XmlConfigurable` interface (in this case, Q2 would push a JDom Element to the QBean so it can deal with the XML fragment in any way it needs).

But most of the time the QBean implementation just requires some configuration properties, so it suffice to implement `org.jpos.core.Configurable` which requires a single method `void setConfiguration (Configuration) throws ConfigurationException` (which is provided by `QBeanSupport`).

8.5. Q2 Dynamic ClassLoading

Q2 scans the `deploy` directory looking for new QBean descriptors, and also scans the `deploy/lib` directory looking for new jars, which become automatically available to QBeans.

If you have to add a new feature to a running system, you can isolate its functionality in a new jar, place that jar in `deploy/lib` and then place the XML descriptor in the `deploy` directory so that Q2 can take care of starting your QBean.

In addition to local dynamic classloading, Q2 can pick the required jars from a remote repository or location withing the filesystem. In order to take advantage of this feature, your QBean descriptor should look like this:

```
<qbean class="com.example.YourQBean" name="your-qbean" logger="Q2">
 <classpath>
  <url>http://com.example/q2/Your.jar</url>
  <url>http://com.example/q2/YourOther.jar</url>
 </classpath>
 ...
 ...
</qbean>
```

Tip

Using a remote repository can be very useful when running a cluster of Q2 nodes.

Note

Dynamic Classloading brings a subtle problem into play when dealing with `Singletons` such as the `NameRegistrar` as you end up having one instance per classloader. If you intend to use this feature, you have to be aware of this problem and make sure that you have a single jPOS.jar available in your main `lib` directory.

8.6. Q2 Scripts

Q2 supports a very easy way to run BSH or Jython-based scripts:

```
<script name="MyBSHScript">
 print ("Hello BSH Script");
</script>
```

...or:

```
<jython name="MyJythonScript">
 print ("Hello Jython");
</jython>
```

If you look at `q2/src/main/org/jpos/q2/QFactory.properties`, you'll notice that the special names `script` and `jython` are mapped to the QBean implementations:

```
 script=org.jpos.q2.qbean.BSH
 jython=org.jpos.q2.qbean.Jython
```

The BSH script support places a few useful references to Q2 objects that can be used by your script implementation, namely:

Table 8.1. BSH script context variables

Variable name	Description
qbean	a reference to `this` qbean, that can be used for example to check for `qbean.running()`
log	the log object associated with this qbean, you can use call `log.info ("BSH is running");`

Example 8.1. BSH based Q2 script

```
<script name='echo-test'>
 import org.jpos.iso.*;
 import org.jpos.util.*;

 MUX mux = (MUX) NameRegistrar.get ("mux.mymux");

 for (int i=1; qbean.running(); i++) {
     ISOMsg m = new ISOMsg();
     m.setMTI ("0800");
     m.set (11, ISOUtil.zeropad (Integer.toString (i % 1000000), 6));
     m.set (70, "301");
     ISOMsg r = mux.request (m, 5000);
     if (r == null) {
         i == 0;
         ISOUtil.sleep (10000);
     } else {
         ISOUtil.sleep (60000);
```

```
      }
  }
 </script>
```

Tip

BSH supports a powerful (yet dangerous, security-wise) debugging tool that can be deployed using a single-line script (see below), and then telneting to your host (in the example below, by using port 6666), in order to get an interactive `bsh` shell that runs on your live Q2 JVM. This tool can be used to debug a running application, query the NameRegistrar contents, check memory usage and take corrective actions.

```
<script name='bsh-server'>
 server(6666);
</script>
```

8.7. Q2 SpaceLet

A SpaceLet (`org.jpos.q2.qbean.SpaceLet`) is a Space interceptor with scripting capabilities. It lets the user define a special space that has the opportunity to react to certain space operations using BSH scripts.

Example 8.2. SpaceLet example

```
<spacelet name="mytest">
 <space>jdbm:test</space>

 <out>
  String cntKey = key + ".count";
  Integer counter = sp.inp (cntKey);
  int cnt = counter != null ? counter.intValue() : 0;
  sp.out (cntKey, new Integer (++cnt));
 </out>

 <in>
  if (sp.rdp (key) != null) {
     String cntKey = key + ".count";
     Integer counter = sp.inp (cntKey);
     int cnt = counter != null ? counter.intValue() : 0;
     sp.out (cntKey, new Integer (--cnt));
  }
 </in>
</spacelet>
```

In the previous example, whenever you `out` an entry into the Space `spacelet:mytest`, another entry with the same key and a `".count"` suffix is automatically maintained by the SpaceLet.

In addition to `<in>` and `<out>` , the SpaceLet QBean supports a `<rd>` element as well as an `<init>` script that can be used to perform initial housekeeping of the Space, as well as multiple `<run>` script(s) that can be used to perform additional Space-related tasks (such as space monitoring, house-keeping, etc).

`<out>` , `<in>` and `<rd>` accept a `"source"` attribute, so you can have an external bsh source file instead of the internal script.

If you specify a name in the spacelet element, then that name is used in the space URI (used by SpaceFactory) instead of "default", e.g.,: `<spacelet name="mySpace">` would be registered as `"spacelet:mySpace"`.

In the `out` operation, if the script returns a true boolean value, it is considered that it already has taken care of the operation so that SpaceLet won't out that entry into the space. On the other hand, if the script returns false (or it doesn't return anything), the SpaceLet implementation would honor the operation.

In a `rd` or `in` operation, if the script returns true, then the "value" script variable is returned instead of querying the underlying space.

Tip

While this feature sounds very powerful, you should use it with care, as SpaceLet operations are not as fast as direct space operations or compiled SpaceInterceptors.

SpaceLets can be used for simple tasks such as triggering the garbage collector in a JDBM space, e.g.:

```
<spacelet name="test">
 <space>jdbm:test:/tmp/test</space>
 <run>
  import org.jpos.space.JDBMSpace;
  if (sp instanceof JDBMSpace) {
    while (spacelet.running()) {
        Thread.sleep (60000);
        ((JDBMSpace)sp).gc ();
    }
  }
 </run>
</spacelet>
```

In the previous example, there's no performance impact as the application would use `jdbm:test` directly and not the intercepted space `spacelet:test`.

Chapter 9. Q2Mod jPOS

Q2Mod jPOS (located at `modules/q2mod_jpos`) gives Q2 most of its jPOS functionality. It is required to run jPOS inside the Q2 container.

It contains adaptors which provide the Q2 lifecycle (init/start/stop/destroy) to existing jPOS and legacy QSP components. It also adds a few components unique to Q2.

The adaptors live in the following packages:

- `org.jpos.q2.iso`
- `org.jpos.q2.security`
- `org.jpos.q2.ui`

Table 9.1. Q2 Modules

Package	Name	Description
iso	ChannelAdaptor	Can be used to configure existing ISOChannel implementations
iso	QMUX	A new MUX implementation that replaces ISOMUX
iso	QServer	Can be used to configure existing ISOServer
iso	DirPollAdaptor	Can be used to configure DirPoll
iso	TaskAdaptor	Adaptor for legacy QSP Tasks
iso	DailyTaskAdaptor	Adaptor for legacy QSP Daily Tasks
iso	CardAgentAdaptor	Used to configure CardAgents
security	KeyStoreAdaptor	Used to configure a KeyStore
security	SMAdaptor	Used to configure a Security Module provider
ui	UI	Can be used to configure UI components

9.1. ChannelAdaptor

The ChannelAdaptor is an unique component not available in previous versions of jPOS/QSP. It is used to configure a standard client ISOChannel and it takes care of keeping it connected to a remote host.

In addition to that, ChannelAdaptor's communication with other components is performed through a Space-based interface, which brings into play the ability to have more than one channel connected to a given host, as well as the ability to distribute channels accross multiple Q2 machines in a cluster.

Let's have a look at a sample configuration:

```
<channel-adaptor name='sample-channel-adaptor'
    class="org.jpos.q2.iso.ChannelAdaptor" logger="Q2">
 <channel class="org.jpos.iso.channel.ASCIIChannel" logger="Q2"
       packager="org.jpos.iso.packager.GenericPackager"
       header="ISO016000055">

  <property name="host" value="192.168.0.1" />
  <property name="port" value="1234" />
  <property name="packager-config" value="cfg/iso87ascii.xml" />
 </channel>
 <in>sample-send</in>
 <out>sample-receive</out>
 <reconnect-delay>5000</reconnect-delay>
 <space>tspace:default</space>
</channel-adaptor>
```

Tip

`in` and `out` are seen from the ChannelAdaptor's standpoint. Input to the ChannelAdaptor means something that has to be actually sent throught the underlying ISOChannel.

When you have to send a message to a channel using the ChannelAdaptor, you can use code like this:

```
  import org.jpos.space.Space;
  import org.jpos.space.SpaceFactory;
  import org.jpos.iso.ISOMsg;
  ...
  ...

  Space sp = SpaceFactory.getSpace();
  ISOMsg m = new ISOMsg ("0800");
  m.set (3, "000001");
  m.set (70, "301");

  sp.out ("sample-send", m);   // "sample-send" is ChannelAdaptor's input

  ISOMsg m = (ISOMsg) sp.in ("sample-receive", 60000);
  ...
  ...
```

The ChannelAdaptor places a special entry in the Space in order to inform other components about the underlying channel status. That entry has a special arbitrary name, which is composed by the adaptor's name plus the literal `.ready`. So in our previous example, the adaptor was called "sample-channel-adaptor". If we want to know if the channel is ready, we can easily check:

```
if (sp.rdp ("sample-channel-adaptor.ready" != null)) {
   ...
   ...
}
```

ChannelAdaptor implements the `org.jpos.iso.Channel` interface, so you can locate the channel by its name via the `NameRegistrar` and use traditional methods like:

```
   public void send (ISOMsg m);
   public ISOMsg receive ();
   public ISOMsg receive (long timeout);
```

```
```

In addition to those methods, ChannelAdaptor provides:

```
public void send (ISOMsg m, long timeout);
public boolean isConnected();
```

Note

ChannelAdaptor was designed to operate with QMUX, the new-generation space-based multiplexer intended to replace ISOMUX (see Section 9.2, “QMUX”).

The 'channel' element of the channel adaptor supports the following attributes and properties:

Table 9.2. ChannelAdaptor attributes

Attribute	Description
class	ISOChannel implementation classname
packager	ISOPackager implementation classname
logger	an optional logger
realm	optional logger's realm
header	optional channel header
socketFactory	optional socketFactory implementation

Table 9.3. Channel properties

Property	Description
host	remote host name or IP address
port	remote port
max-packet-length	optional parameter, defaults to 100000
local-iface	optional parameter, local interface name used in multi-homed setup
local-port	optional local port used in multi-homed setup
alternate-host	if main host/port can't be reached, use list of alternate hosts/ports
alternate-port	see previous property, alternate-host. The number of alternate-port properties must match the number of alternate-host properties
override-header	optional parameter, defaults to false. If true, the channel's header overrides the ISOMsg's header
timeout	socket level timeout in millis. The channel is considered invalid if no messages are received after the given timeout. Defaults to no timeout but it's strongly

Property	Description
	recommended to set one
connect-timeout	timeout used at connect time in millis

9.2. QMUX

QMUX is a modern, very simple yet powerful Q2 component designed to replace ISOMUX. It closely adheres to the Q2 framework design where components can be added and removed on the fly.

The main difference between ISOMUX and QMUX is that the latter uses the Space in order to communicate with the underlying channels; this strategy brings into play a whole new set of deployment options, including the ability to multiplex several channels for redundancy/load-balancing. These channels doesn't even have to run on the same machine. They can use distributed/remote space implementations.

A QMUX configuration looks like this:

```
<mux class="org.jpos.q2.iso.QMUX" logger="Q2" name="mymux">
 <in>sample-receive</in>
 <out>sample-send</out>
</mux>
```

If you recall our ChannelAdaptor configuration in the previous section, you'll notice that the "in" and "out" queue names are interchanged. `in` and `out` are seen from the component's perspective.

QMUX is registered in the NameRegistrar under the name provided in the qbean configuration file ("mymux" in our example). So that other components can get a reference, cast it to MUX and use its:

```
public ISOMsg request (ISOMsg m, long timeout) throws ISOException;
```

method.

If you remember QSP and ISOMUX, you might wonder how to handle incoming messages that don't match an outgoing request (i.e. late responses, network management messages originated by a remote server, etc.).

The old ISOMUX had a request listener for that, but in Q2, with clusters, redundancy and scalability in mind, we use the space instead. You can configure an optional `unhandled` queue, and QMUX will place all arriving messages that don't match any pending request in that queue. The configuration looks like this:

```
<mux class="org.jpos.q2.iso.QMUX" logger="Q2" name="mymux">
 <in>sample-receive</in>
 <out>sample-send</out>
 <unhandled>mymux.unhandled</unhandled>
</mux>
```

A separate process can be used to pull messages out of the `mymux.unhandled` queue for further processing.

In addition to the `unhandled` queue, QMUX supports multiple request listeners, e.g:

```
  <mux class="org.jpos.q2.iso.QMUX" logger="Q2" name="mymux">
  <in>sample-receive</in>
  <out>sample-send</out>
  <unhandled>mymux.unhandled</unhandled>
  <request-listener class="my.request.listener" logger="Q2" realm="myrealm">
    <property name="myproperty" value="abc" />
    <property name="myotherproperty" value="xyz" />
    <property file="cfg/myprop.cfg" /> <!-- properties taken from a file -->
  </request-listener>
</mux>
```

In order to match responses with their original requests, QMUX uses a constructed **key** which defaults to the matching response MTI plus data element 41 (terminal ID) and data element 11 (Serial Trace Audit Number).

The user can override the fields used to compose a key by using the 'key' element, e.g:

```
  <mux ...>
   ...
   <key>11, 37, 41, 42</key>
  </mux>
```

9.3. QServer

QServer is a simple service that wraps an ISOServer. The configuration looks like this:

```
  <server name="xml-server" class="org.jpos.q2.iso.QServer" logger="Q2">
   <attr name="port" type="java.lang.Integer">8000</attr>
   <!-- optional server-socket-factory element
   <server-socket-factory class="org.jpos.iso.SunJSSESocketFactory" logger="Q2">
    <property name="keystore" value="/path/to/your/keystore.jks" />
     <property name="clientauth" value="true" />
     <property name="serverauth" value="true" />
     <property name="storepassword" value="secret" />
     <property name="keypassword" value="secret" />
     <property name="addEnabledCipherSuite" value="SSL_RSA_WITH_3DES_EDE_CBC_SHA" />
     <property name="addEnabledCipherSuite" value="TLS_DHE_DSS_WITH_AES_256_CBC_SHA" />
   </server-socket-factory>
   -->
   <channel name="xml.channel"
           class="org.jpos.iso.channel.XMLChannel"
           packager="org.jpos.iso.packager.XMLPackager" logger="Q2">
   </channel>
   <request-listener class="my.request.Listener" logger="Q2">
     <property name="my-property" value="ABC" />
     <property name="my-other-property" value="XYZ" />
   </request-listener>
  </server>
```

QServer uses a ThreadPool to handle incoming connections. The default limits for the ThreadPool are 1 (minSessions) and 100 (maxSessions), but you can easily change that by adding the following attributes to your configuration :

```
   <attr name="minSessions" type="java.lang.Integer">10</attr>
   <attr name="maxSessions" type="java.lang.Integer">250</attr>
```

QServer supports access control based on the peer's IP address, e.g.:

```
<property name="allow" value="192.168.1.1" />
<property name="allow" value="192.168.1.2" />
...
...
```

Note

jPOS/ISOServer uses a ThreadPool in order to accept simultaneous connections. Every thread in that pool then handles incoming messages, and forwards them to the registered ISORequestListeners in a synchronous way (single-thread). It is the responsibility of the ISORequestListener to use another ThreadPool in order to handle simultaneous, long-lived transactions.

Tip

In order to avoid having to use a ThreadPool to handle long-lived sessions, the user may choose to use the space in order to hand off the processing of incoming requests to another component, such as the TransactionManager.

9.4. DirPoll

`org.jpos.q2.iso.DirPollAdaptor` is used to instantiate a DirPoll component, which can be used to poll a directory for new requests or files to process (i.e. interchange files, settlement files, or simple disk-based authorization requests) (See Section 4.6, “DirPoll”).

The DirPoll configuration looks like this:

```
<dir-poll  class="org.jpos.q2.iso.DirPollAdaptor" logger="Q2">
  <attr name="path">spool</attr>
  <attr name="priorities">.txt</attr>
  <attr name="poolSize" type="java.lang.Integer">1</attr>
  <attr name="pollInterval" type="java.lang.Long">5000</attr>
  <attr name="processor">my.dirpoll.Processor</attr>
</dir-poll>
```

Table 9.4. DirPoll attributes

Attribute	Description
path	Directory path where DirPoll should poll for requests
poolSize	Indicates the maximum number of concurrent requests that can be delegated simultaneously to DirPoll's processor.
pollInterval	Poll time expressed in milliseconds.
priorities	An optional attribute consisting of a space-separated list of file extensions indicating poll priority.
processor	A class implementing either the org.jpos.util.DirPoll.Processor or org.jpos.util.DirPoll.FileProcessor interface.

9.5. TaskAdaptor

QSP has a handy way of integrating POJOs by using a `<task>` configuration fragment.

Q2 has an adaptor for these simple tasks that you can use with a configuration like this:

```
<task name="MyTask" class="org.jpos.q2.iso.TaskAdaptor" logger="Q2">
 <class>com.mycompany.jpos.MyTask</class>
 <property name="OptionalProperty" value="MyProperty" />
</task>
```

If your task happens to implement `org.jpos.core.Configurable`, then TaskAdaptor will take care of pushing a `Configuration` object.

In case your task implements `Runnable`, TaskAdaptor will start a new thread for it.

At stop time, if your task happens to implement `org.jpos.util.Destroyable`, its `destroy()` method would get called.

TaskAdaptor also attempts to set a Logger - provided that the managed class implements `org.jpos.util.LogSource`.

> **Note**
> This component is intended to be used with "legacy" task implementations. In the case of new code, we strongly recommend you implement a QBean, ideally by extending QBeanSupport.

9.6. DailyTaskAdaptor

DailyTaskAdaptor can be used to run jobs at a specific time. It is configured in a similarly to the previously described `TaskAdaptor` with an additional `start` property.

```
<daily-task
 name="MyDailyTask" class="org.jpos.q2.iso.DailyTaskAdaptor" logger="Q2">
 <class>com.mycompany.jpos.MyDailyTask</class>
 <property name="start" value="23:00:00" />
</daily-task>
```

9.7. SMAdaptor

`org.jpos.q2.security.SMAdaptor` takes care of instantiating a security module adaptor and registering it with the NameRegistrar:

Example 9.1. SMAdaptor Configuration

```
<ssm name="MySM" class="org.jpos.q2.security.SMAdaptor" logger="Q2">
 <attr name="impl">org.jpos.security.jceadapter.JCESecurityModule</attr>
 <property name="provider" value="com.sun.crypto.provider.SunJCE" />
 <property name="lmk" value="cfg/lmk-test" />
```

```
  </ssm>
```

The `impl` attribute is used to define the SMAdaptor provider implementation.

9.8. KeyStoreAdaptor

This QBean takes care of instantiating a KeyStore and registering it with the NameRegistrar:

Example 9.2. KeyStoreAdaptor configuration

```
<ks class="org.jpos.q2.security.KeyStoreAdaptor" logger="Q2">
 <attr name="impl">org.jpos.security.SimpleKeyFile</attr>
 <property name="key-file" value="cfg/keys-test" />
</ks>
```

9.9. QExec

QExec lets you run external applications at start and stop time. It can be used to notify other applications (such as system status monitoring, database servers, etc.) about Q2 starting or stopping.

Example 9.3. QExec configuration

```
<exec class="org.jpos.q2.qbean.QExec">
  <attr name="start">path/to/start_program</attr>;
  <attr name="shutdown">path/to/stop_program</attr>
</exec>
```

9.10. Jetty Integration

Q2 can be wrapped inside a 'war' or 'sar' and deployed inside a webcontainer or application server, such as Tomcat or JBoss. Or it can be used the other way around, starting an embedded web container, such as Jetty.

Running Jetty inside Q2 is as simple as deploying the following QBean:

```
<jetty class="org.jpos.q2.jetty.Jetty">
 <attr name="config">path/to/jetty.xml</attr>
</jetty>
```

> **Note**
> Describing the Jetty configuration (`path/to/jetty.xml`) is beyond the scope of this programmer's guide; detailed information can be obtained in Jetty's website [http://www.mortbay.com/].

Chapter 10. TransactionManager

10.1. Overview

jPOS is typically used to implement mission-critical applications that have to deal carefully with error conditions.

When you access a web page and a transient network error occurs, you just hit the *reload* button on your browser. By contrast, a complex financial transaction involves a lot of activities such as contacting remote hosts, perform PIN-based validations and pinblock transactions, database logging, etc. So, if something goes wrong or your machine just dies due to a power failure, its more complicated than simply hitting the *reload* button: you have to reverse the impact of whatever actions had been committed until the failure point.

The `org.jpos.transaction` package - along with the Q2-based **TransactionManager** implementation - provides the necessary framework and components required to deal with the previous scenerio. This combination also fosters code reuse and 'componentization.'

The key class is the **TransactionParticipant** http://jpos.org/doc/javadoc/org/jpos/transaction/TransactionParticipant.html, which exposes the following the following interface:

```
public interface TransactionParticipant extends TransactionConstants {
    public int  prepare (long id, Serializable context);
    public void commit  (long id, Serializable context);
    public void abort   (long id, Serializable context);
}

(for the records, TransactionConstants provides the following constants)

public interface TransactionConstants {
    public static final int ABORTED  = 0;
    public static final int PREPARED = 1;
    public static final int NO_JOIN  = 0x40;
    public static final int READONLY = 0x80;
}
```

The TransactionManager implementation takes care of 'driving' the transaction by calling all of the participants' `prepare` methods. If all of them return 'PREPARED', then the transaction moves to the *committing* phase, at which point the TransactionManager will call all of the participants' `commit` methods.

If one of the participants' `prepare` methods were to return `ABORTED`, then the transaction would move into an *aborting* phase, and all the participants' `abort` methods would get called.

10.2. Transaction Constants

Table 10.1. Return values

Name	Value	Description
ABORTED	0	The participant is not prepared. The transaction must be aborted.

Name	Value	Description
PREPARED	1	This participant is prepared to commit the transaction, provided all other participants down the list return PREPARED, too.
Modifiers		
NO_JOIN	0x40	This modifier is a hint to the TransactionManager to let it know that it is not required to call this participant's `commit` or `abort` methods once the *committing* or *aborting* phase of the transaction is reached.
READONLY	0x80	This modifier is a hint to the TransactionManager to let it know that this participant has not modified any persistent information in the context, so saving a snapshot of the context is not required.

Note

Despite the fact that a partipant may indicate that it doesn't want to JOIN a given transaction (by using the `NO_JOIN` run-time modifier), under certain recovery situations the TransactionManager may still call its `commit` or `abort` method, so the participant developer should be prepared to receive a `commit` or `abort` call for an unknown transaction. The code should check the `long id` and / or `Serializable context` in order to figure out what to do.

10.3. Transaction Context

The only constraint imposed on a Context implementation is that it has to implement the `java.io.Serializable` interface. That's because the TransactionManager has to write `snapshots` of it at different points.

You can use any `Serializable` object, either a custom object such as an application-specific *Bean*, or a general-purpose object such as a `java.util.Map` implementation (e.g., a `Hashmap`).

We found it very useful to use a general-purpose context holding two maps, a regular (persistent) map and a transient map, e.g.:

```
public class Context implements Serializable, Loggeable {
    private transient Map tmap; // transient map
    private Map map;            // persistent (serializable) map

    public Context () {
        super ();
        tmap = Collections.synchronizedMap (new LinkedHashMap ());
        map  = Collections.synchronizedMap (new LinkedHashMap ());
    }
}
```

...with methods such as:

```
public void put (Object key, Object value) {
    map.put (key, value);
}
public Object get (Object key) {
    return map.get (key);
}
public Object remove (Object key) {
    return map.remove (key);
}
```

...which operate in the standard (persistent/serializable) map... and a set of `t*` methods:

```
public void tput (Object key, Object value) {
    tmap.put (key, value);
}
public Object tget (Object key) {
    return tmap.get (key);
}
public Object tremove (Object key) {
    return tmap.remove (key);
}
```

...which operate in the transient map.

The transient map gives you the ability to place non-Serializable objects in the context, for example: a *live* object such as a socket connection; a jdbc connection; a thread, managed object, etc.

10.4. Context Recovery interface

In the previous section, we described a Transaction Context holding two maps: a transient map and a persistent map.

In a situation where the TransactionManager dies (e.g., during a power failure), a transaction may be in its *preparing*, *committing* or *aborting* phase.

Either the `commit` or `abort` methods will be called on all participants, but before that happens, the TransactionManager gives the developer the posibility to let the participant know that we are not dealing with a normal situation but a recovery situation.

The developer can also implement the `ContextRecovery` interface:

```
public interface ContextRecovery {
    public Serializable recover
        (long id, Serializable context, boolean commit);                    (1)
}
```

(1) commit = true (if the transaction is going to commit); false (if it's going to abort).
The TransactionManager provides the opportunity to build up the transient part of the Context (e.g., re-stablishing a JDBC connection, re-fetching a database record based on some persistent ID number, etc.).

10.5. AbortParticipant

Imagine you have a list of participants that conform a transaction, for example:

- ValidateMessage (sanity checks)
- FetchData (i.e. get Terminal/Merchant info from database)
- QueryRemoteHost
- LogTransaction
- SendResponse

If everything goes okay and all participants return `PREPARED`, then you'll have no problem reaching the last set of participants. By contrast, if for some reason a given participant fails (e.g., suppose `FetchData` fails), then the succeeding participants down the list (in our example, `FetchData` through `SendRepsonse`) won't get called because the transaction manager will initiate the *aborting* procedure (which will call `abort(id,context)` only on the previously-called participants, i.e., only on `ValidateMessage` in our example).

In the previous example, while it's okay to ignore a call to the `QueryRemoteHost` participant, you may still want to send a response to the client, or even log the transaction.

So the developer can implement the optional `AbortParticipant` interface:

```
    public interface AbortParticipant extends TransactionParticipant {
        public int  prepareForAbort (long id, Serializable context);
    }
```

...in order to let the TransactionManager know that you still want to join the transaction, even if it's known to have failed.

10.6. TransactionManager

Reference Implementation

`org.jpos.q2.transaction.TransactionManager` provides a Q2-compatible reference implementation of the previous contracts. Let's have a look at a sample configuration:

```
<txnmgr name="MyTxn" logger="Q2"
       class="org.jpos.q2.transaction.TransactionManager">
 <property name="space"            value="tspace:default" />
 <property name="queue"            value="MyTxnQueue" />
 <property name="persistent-space" value="jdbm:MyTxnSpace" />
 <property name="sessions"         value="32" />

 <participant class="com.my.company.ValidateMessage"   logger="Q2" />
 <participant class="com.my.company.FetchData"         logger="Q2" />
 <participant class="com.my.company.QueryRemoteHost"   logger="Q2" />
 <participant class="com.my.company.LogTransaction"    logger="Q2" />
 <participant class="com.my.company.SendResponse"      logger="Q2">
  <property name="my.optional.property" value="abc" />
  <property name="my.other.optional.property" value="xyz" />
 </participant>
</txnmgr>
```

Table 10.2. TransactionManager configuration

Property	Description
space	Space used to inject transactions into this transaction manager.
queue	Queue name where the transaction manager pulls for new messages to be processed (`sp.in(queue)`).
sessions	Number of transactions that can be processed simultaneously.
persistent-space	This TransactionManager reference implementation uses a persistent space to store a snapshot of ongoing transactions for recovery procedures. This persistent space has to be unique (see note).

Note

The transaction manager requires exclusive access to its persistent space. If you plan to deploy more than one transaction manager, you have to use different persistent spaces.

10.7. Integration with ISORequestListener

It is extremely easy to integrate this TransactionManager framework with our ISORequestListeners. Here is how a `process` can be implemented:

```
// Constants defined elsewhere ...
public static final String REQUEST   = "REQUEST";
public static final String ISOSOURCE = "ISOSOURCE";

public boolean process (ISOSource source, ISOMsg m) {
    Context ctx = new Context ();
    ctx.put  (REQUEST, m);
    ctx.tput (ISOSOURCE, source);                                    (1)
    sp.out (cfg.get "queue", ctx, cfg.getLong ("timeout"));          (2)
    return true;
}
```

(1) Please note ISOSource is a `live object`. Therefore, it makes no sense to store it in our persistent snapshot for recovery. Instead, we use the transient map of our custom context (as described above).

(2) Hands off the request to the TransactionManager. ISORequestListener is free to return.

The transaction manager then pulls that context out of the queue, processes it, and uses the `ISOSource` reference in order to send a response back to the client.

10.8. GroupSelector

Having a configuration like this:

```
<txnmgr ...>
 <participant A />
 <participant B />
 <participant C />
 <participant D />
```

```
  ...
</txnmgr>
```

...may be good for some applications, but you risk ending up having to configure multiple transaction managers for different classes of transactions (e.g., network management, authorization, draft capture, etc.).

In order to simplify the TransactionManager configuration, we've added a very simple interface called `GroupSelector`:

```
  public interface GroupSelector extends TransactionParticipant {
      public String select (long id, Serializable context);
  }
```

A participant implementing the GroupSelector interface can modify the flow of this transaction by returning a space-separated list of group names (or can specify 'null' to signify no action).

Our Q2-based TransactionManager reference implementation supports this interface and lets you design your own configuration file with a structure like this:

```
<txnmgr ...>
 <participant A />
 <participant B />
 ...
 ...
 <group name="GroupA">
  <participant A />
  <participant B />
  <participant C />
 </group>
 <group name="GroupB">
  <participant J />
  <participant K />
  <participant L />
 </group>
 <group name="GroupC">
  <participant X />
  <participant Y />
  <participant Z />
 </group>
 ...
 ...
</txnmgr>
```

Example 10.1. GroupSelector

```
  public class Switch implements GroupSelector {
      public int  prepare (long id, Serializable context) {
          return PREPARED | READONLY | NO_JOIN;
      }
      public void commit   (long id, Serializable context) { }
      public void abort    (long id, Serializable context) { }
      public String select (long id, Serializable context) {
          try {
              ISOMsg m = getRequest ((Context) context);
              String groups = cfg.get (m.getMTI(), null);
              return groups;
          } catch (Exception e) {
              warn (e);
```

```
                return null;
            }
        }
    }
```

By using the `Switch` presented in the previous example, you can write a TransactionManager configuration file like this:

```
   ...
   ...
   <participant class="org.jpos.my.Switch" logger="Q2">
    <property name="0100" value="Authorization Response Log" />
    <property name="0200" value="Financial Response Log" />
    <property name="0220" value="Notification Response Log" />
    <property name="0221" value="Notification Response Log" />
    <property name="0420" value="Reversal Response Log" />
    <property name="0421" value="Reversal Response Log" />
    <property name="0500" value="BatchManagement Response Log" />
    <property name="0421" value="Reversal Response Log" />
    <property name="0800" value="NetworkManagement Response Log" />
   </participant>
   ...
   ...
   <group name="Financial">
    <participant class="com.my.company.CheckRequiredFields">
     <property name="fields" value="0,3,4,17,49,32,41,43,37,PAN,AMOUNT" />
    </participant>
    <participant class="com.my.company.CheckCurrency" />
    ...
    ...
   </group>

   <group name="Reversal">
    ...
    ...
   </group>
   ...
   ...
```

Using the previous approach, the application can be designed using small reusable participants.

We have found it very useful to have very small participants to peform tasks like: Debug the context; introduce Delays (during testing); Open and Close O/R mapping sessions, etc.

10.9. Context implementation

Example 10.2. General Purpose context implementation

```
package org.jpos.transaction;

import java.util.Map;
import java.util.LinkedHashMap;
import java.util.Iterator;
import java.util.Collections;
import java.io.IOException;
import java.io.Serializable;
import java.io.PrintStream;
import org.jdom.Element;
import org.jdom.output.Format;
```

```
import org.jdom.output.XMLOutputter;
import org.jpos.util.Loggeable;

public class Context implements Serializable, Loggeable {
    private transient Map tmap; // transient map
    private Map map;            // persistent (serializable) map

    public Context () {
        super ();
        tmap = Collections.synchronizedMap (new LinkedHashMap ());
        map  = Collections.synchronizedMap (new LinkedHashMap ());
    }
    /**
     * Transient Put
     */
    public void tput (Object key, Object value) {
        tmap.put (key, value);
    }
    /**
     * Transient Get
     */
    public Object tget (Object key) {
        return tmap.get (key);
    }
    /**
     * Transient remove
     */
    public Object tremove (Object key) {
        return tmap.remove (key);
    }
    /**
     * Transient get with default
     */
    public Object tget (Object key, Object defaultValue) {
        Object value = tmap.get (key);
        if (value == null)
            value = defaultValue;
        return value;
    }
    /**
     * Persistent put
     */
    public void put (Object key, Object value) {
        map.put (key, value);
    }
    /**
     * Persistent get
     */
    public Object get (Object key) {
        return map.get (key);
    }
    /**
     * Persistent remove
     */
    public Object remove (Object key) {
        return map.remove (key);
    }
    /**
     * Persistent get with default
     */
    public Object get (Object key, Object defaultValue) {
        Object value = map.get (key);
        if (value == null)
            value = defaultValue;
        return value;
    }
    public long getLong (Object key, long defaultValue) {
        Long L = (Long) map.get (key);
        return L == null ? defaultValue : L.longValue();
    }
    public long getLong (Object key) {
```

```
        return getLong (key, 0L);
    }
    public long getTLong (Object key, long defaultValue) {
        Long L = (Long) tmap.get (key);
        return L == null ? defaultValue : L.longValue();
    }
    public long getTLong (Object key) {
        return getTLong (key, 0L);
    }
    /**
     * @return transient map
     */
    public Map getTransient() {
        return tmap;
    }
    /**
     * @return persistent map
     */
    public Map getPersistent() {
        return map;
    }
    /**
     * Persistent get String
     */
    public String getString (Object key) {
        return (String) map.get (key);
    }
    public void dump (PrintStream p, String indent) {
        String inner = indent + "  ";
        p.println (indent + "<context>");
        p.println (indent + " <transient>");
        dumpMap (tmap, p, inner);
        p.println (indent + " </transient>");

        p.println (indent + " <persistent>");
        dumpMap (map, p, inner);
        p.println (indent + " </persistent>");

        p.println (indent + "</context>");
    }
    private void dumpMap (Map map, PrintStream p, String indent) {
        if (map == null)
            return;

        Iterator iter = map.entrySet().iterator();
        while (iter.hasNext()) {
            Map.Entry entry = (Map.Entry) iter.next ();

            p.print (indent +
                "<entry key=\"" + entry.getKey().toString() + "\">");
            Object value = entry.getValue();
            if (value instanceof Loggeable) {
                p.println ("");
                ((Loggeable) value).dump (p, indent + " ");
                p.print (indent);
            } else if (value instanceof Element) {
                p.println ("");
                p.println (indent+ "<![CDATA[");
                XMLOutputter out = new XMLOutputter (Format.getPrettyFormat ());
                out.getFormat().setLineSeparator ("\n");
                try {
                    out.output ((Element) value, p);
                } catch (IOException ex) {
                    ex.printStackTrace (p);
                }
                p.println ("");
                p.println (indent + "]]>");
            } else if (value != null) {
                p.print (value.toString ());
            } else {
                p.print ("nil");
```

```
            }
            p.println ("</entry>");
        }
    }
    private static final long serialVersionUID = 1L;
}
```

Chapter 11. UI Framework

11.1. Overview

jPOS UI is a very simple and lightweight UI layer that enables you to write reusable UI components and easily assemble them by using a QBean configuration file.

It can be embedded in standalone applications (see `src/examples/ui/Test.java`) or used inside Q2.

The UI framework is based in two core interfaces: `UIFactory` and `ActionListeners (java.awt.event.ActionListener)`.

The components are created by UI factories that honor the following contract:

```
   public interface UIFactory {
       public JComponent create (UI ui, Element config);
   }
```

The overall structure of the UI configuration looks like this:

```
  <ui>
   <caption>...</caption>
   <components>
    ...
    ...
   </components>
   <action ... >
   <action ... >
   ...
   <custom-panels>
     <!-- see src/examples/ui/redirect.xml -->
     ...
     ...
   </custom-panels>
   <custom-panels2>
     ...
     ...
   </custom-panels2>
  </ui>
```

We provide a set of ready-to-deploy UIFactories that can be used to create and assemble many Swing components, such as Labels, Buttons, Trees as well as higher-level components such as LogListeners, ISOMeters, etc. Let's have a look at a simple example:

```
<qbean class="org.jpos.q2.ui.UI" name="HelloUI" logger="Q2">
 <ui width="600" height="400"
    full-screen="false" undecorated="false" close="true">
  <caption>jPOS UI Test</caption>

  <menubar id="MAIN">
   <menu id="File" accesskey="f">
    <menuitem id="Quit"  accesskey="Q" action="exit" />
   </menu>
  </menubar>
  <components>
   <border-layout>
```

```
    <north>
     <label font="helvetica-normal-22">Hello jPOS UI</label>
    </north>
    <center>
     <html editable="false" follow-links="true" scrollable="true">
      http://jpos.org
     </html>
    </center>
    <south>
     <label font="arial-italic-8">Copyright (c) 2004 - jPOS.org</label>
    </south>
   </border-layout>
  </components>
 </ui>
 <object class="org.jpos.ui.action.Exit" id="exit" />
</qbean>
```

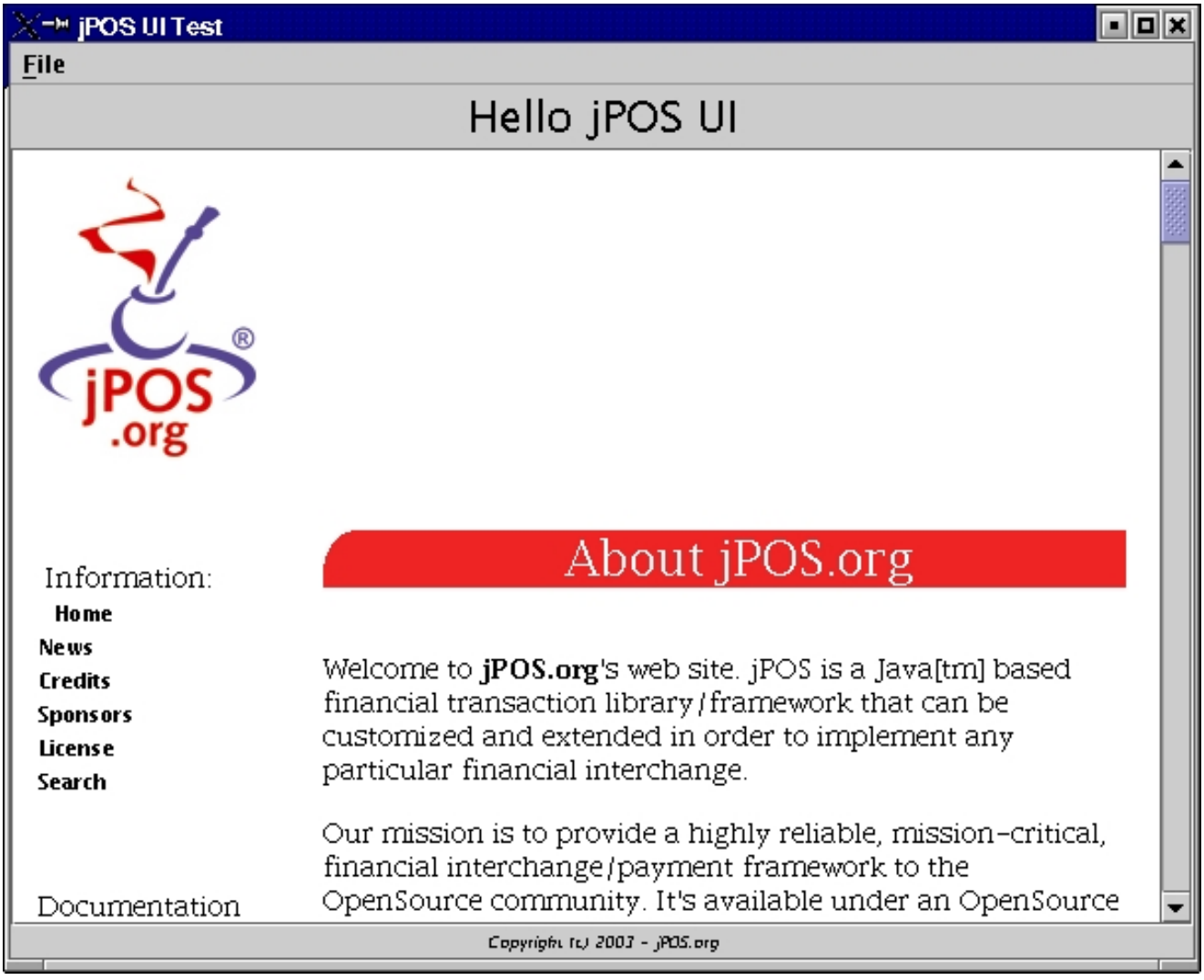


Figure 11.1. Hello UI

11.2. UI configuration

The <ui> configuration element supports the following optional attributes:

Table 11.1. UI configuration attributes

Attribute	Description
width	Width in pixels
height	Height in pixels
look-and-feel	Optional Look and Feel
undecorated	`true` or `false`, defaults to `false`.
close	`true` or `false`, defaults to `true`.

If you run the UI inside Q2, you have to 'wrap' the `<ui>..</ui>` configuration inside a QBean configuration, e.g.:

```
  <qbean class="org.jpos.q2.ui.UI" name="UI" logger="Q2"
      provider="org.jpos.bsh.BSHUI">
    <ui ...>
      ...
      ...
    </ui>
  </qbean>
```

Please note the optional `provider` attribute. This is a 'hook' intended to give the developer the ability to extend the UI framework. It defaults to `org.jpos.ui.UI`. jPOS comes with another provider called `org.jpos.bsh.BSHUI` which enables BSH script-based customization for components.

If you use the BSHUI provider, you can write configuration blocks like this:

```
    ...
    ...
    <label font="helvetica-normal-22">Hello jPOS UI
     <script>
      ((JLabel) component).setForeground (Color.RED);
     </script>
    </label>
    ...
    ...
```

11.3. menubar

The `menubar` can have `menus` which in turn can have `menuitems` as described in the following example:

```
<menubar id="MAIN">
 <menu id="File" accesskey="f">
  <menuitem id="Open"  accesskey="O"/>
  <menuitem id="Edit"  accesskey="E"/>
  <menuseparator />
  <menuitem id="Close" accesskey="C"/>
  <menuitem id="Quit"  accesskey="Q" action="dispose" />
 </menu>
 <menu id="Buttons" accesskey="b">
  <button-group>
   <radio-button id="Button A" selected="false" accesskey="A" />
```

```
    <radio-button id="Button B" selected="true"  accesskey="B" />
    <radio-button id="Button C" selected="false" accesskey="C" />
   </button-group>
   <menuseparator />
   <check-box id="CheckBox D" state="true" command="CheckBoxD" action="debug" />
   <check-box id="CheckBox E" />
   <check-box id="CheckBox F" />
   <check-box id="CheckBox G" />
   <menu id="More Buttons">
    <check-box id="CheckBox A1" state="true" />
    <check-box id="CheckBox B1" />
    <check-box id="CheckBox C1" />
    <check-box id="CheckBox D1" />
    <menuseparator />
    <check-box id="CheckBox D1" state="true" />
    <check-box id="CheckBox E1" />
    <check-box id="CheckBox F1" />
    <check-box id="CheckBox G1" />
    <menu id="Yet More Buttons">
     <check-box id="CheckBox A2" state="true" />
     <check-box id="CheckBox B3" />
     <check-box id="CheckBox C4" />
     <check-box id="CheckBox D5" />
     <menuseparator />
     <check-box id="CheckBox D6" state="true" />
     <check-box id="CheckBox E7" />
     <check-box id="CheckBox F8" />
     <check-box id="CheckBox G9" />
    </menu>
   </menu>
  </menu>
  <menu id="Help" accesskey="h">
   <menuitem id="About"
      accesskey="A" icon="http://www.jpos.org/images/jpos_t.jpg" />
  </menu>
 </menubar>
```

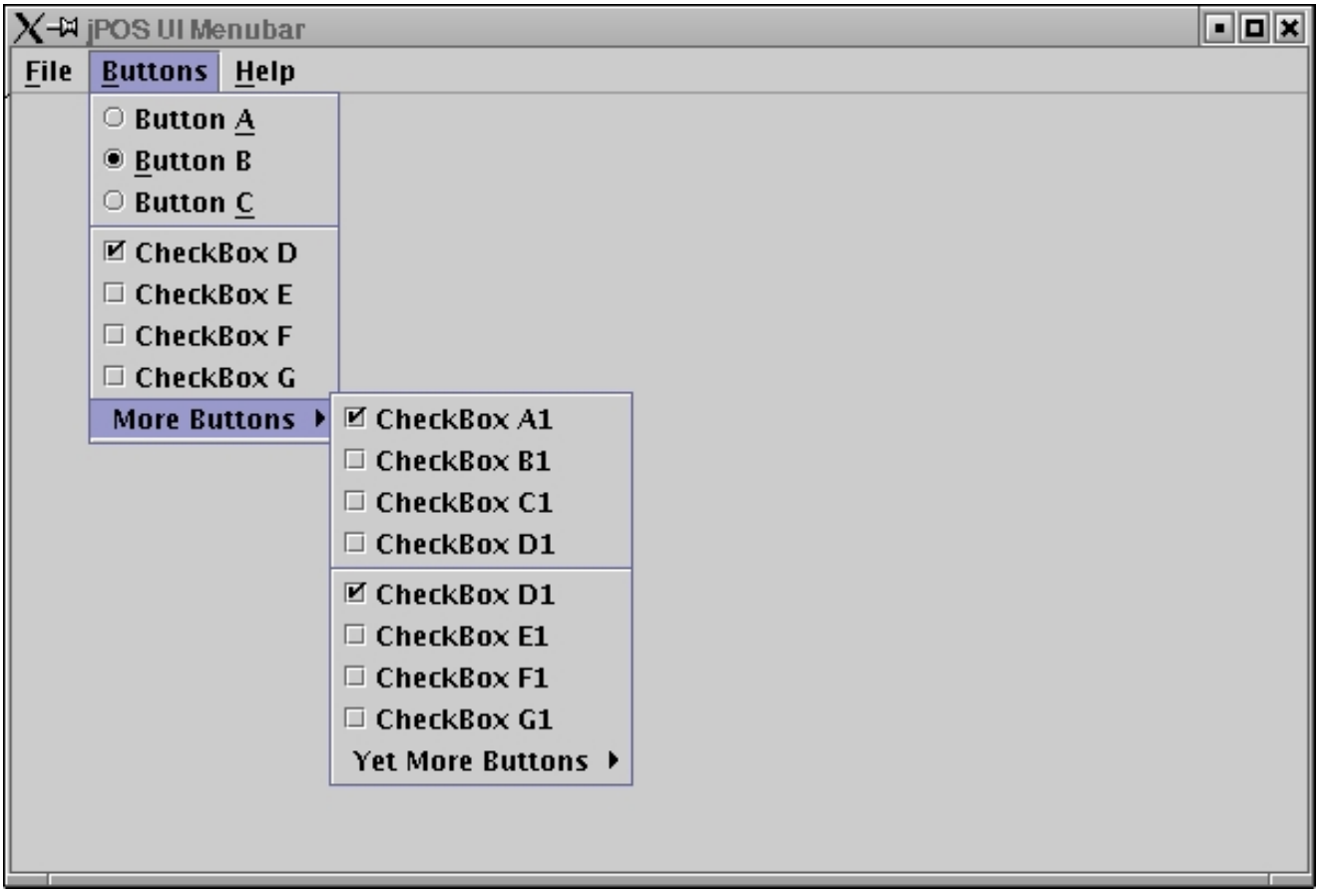


Figure 11.2. Menubar

The optional `id` attribute can be used by the rest of your application to reference instances of these components (i.e., by using the UI object as an instance registrar) in order to modify them on-the-fly (e.g., enable/disable, changing state, etc.).

Most components can be associated with an ActionListener by using the `action` attribute. The optional attribute `command` can be used as a 'hand back' object in order to reuse ActionListener for different components.

11.4. UI Factories

jPOS UI provides several general-purpose factories, but the main idea is that this is a framework where you can also create your own factories.

Let's have a look at a very simple factory, JLabelFactory (`src/main/org/jpos/ui/factory/JLabelFactory.java`):

```
public class JLabelFactory implements UIFactory {
    public JComponent create (UI ui, Element e) {
        JLabel label = new JLabel (e.getText());
        String font = e.getAttributeValue ("font");
        if (font != null)
            label.setFont (Font.decode (font));
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setBorder(new EmptyBorder(3, 3, 3, 3));
        return label;
    }
}
```

Factories just return JComponents, which can be as simple as a label, a button, or any complex, custom component that you may wish to create.

When you define components, you can use any XML element name, but you have to specify which class implements that component, e.g.:

```
<mylabel class="com.mycompany.jpos.ui.MyLabelFactory">My Label</label>
```

In order to make XML configuration files more readable, we provide a mapping file called `UI.properties` (located in `src/main/org/jpos/ui` directory) which currently has the following mappings (at the time of this writing):

```
panel=org.jpos.ui.factory.PanelFactory
label=org.jpos.ui.factory.JLabelFactory
button=org.jpos.ui.factory.JButtonFactory
border-layout=org.jpos.ui.factory.BorderLayoutFactory
grid=org.jpos.ui.factory.GridLayoutFactory
hsplit=org.jpos.ui.factory.HSplitFactory
vsplit=org.jpos.ui.factory.VSplitFactory
tree=org.jpos.ui.factory.JTreeFactory
tabbed-pane=org.jpos.ui.factory.JTabbedPaneFactory
html=org.jpos.ui.factory.HtmlFactory
text=org.jpos.ui.factory.TextFactory
log-listener=org.jpos.ui.factory.LogListenerFactory
iso-meter=org.jpos.ui.factory.ISOMeterFactory
```

```
    #
    # The following element names depend on EXTension modules (see ext/README)
    #
    swixml=org.jpos.ui.factory.SwiXMLFactory
```

By using these mappings, you don't have to specify a `class` attribute when configuring commonly-used components. As a result, you don't have to write:

```
<button class="org.jpos.ui.factory.JButtonFactory">MyButton</button>
```

Instead, you simply write:

```
<button>MyButton</button>
```

The definitive documentation for these factories, is actually the source code, specifically the `create (UI ui, Element e)` method. Even complex factories such as the log-listener have very easy-to-read create methods, e.g.:

```
public JComponent create (UI ui, Element e) {
    JTextArea textArea = new JTextArea (25, 80);
    String font = e.getAttributeValue ("font");
    if (font != null)
        textArea.setFont (Font.decode (font));
    String logId = e.getAttributeValue ("logger", "Q2");
    try {
        int maxEvents = Integer.parseInt (
            e.getAttributeValue ("max-events", "100")
        );
        int maxLines = Integer.parseInt (
            e.getAttributeValue ("max-lines", "1000")
        );
        Logger logger = (Logger) NameRegistrar.get ("logger." + logId);
        logger.addListener (
            new Listener (logger, ui, textArea, maxEvents, maxLines)
        );
    } catch (NameRegistrar.NotFoundException ex) {
        textArea.setText (ex.toString ());
    }
    return textArea;
}
```

You can easily identify which attributes are supported by this component:

Table 11.2. LogListenerFactory configuration

Attribute	Description
font	You can specify an optional font attribute, e.g., `font="helvetica-bold-16"`
logger	The logger name (defaults to `Q2`)
max-events	Maximum number of events that you want to display
max-lines	Maximum number of lines to display

Let's have a look at another example:

```
<qbean class="org.jpos.q2.ui.UI" name="HelloUI" logger="Q2">
 <ui width="600" height="400"
    full-screen="false" undecorated="false" close="true">
  <caption>jPOS UI Layouts</caption>

  <components>
   <border-layout>
    <north>
     <label font="helvetica-normal-22">North</label>
    </north>
    <center>
     <hsplit>
      <left>
       <tree>
        <node>Links
         <node>Link A</node>
         <node>Link B</node>
        </node>
        <node>Nodes
         <node>Node A</node>
         <node>Node B</node>
         <node>Node C</node>
        </node>
       </tree>
      </left>
      <right>
       <vsplit>
        <top>
         <text type="text/html"><![CDATA[
            <h2>The quick brown fox jumped over the lazy dog</h2>
            ]]>
         </text>
        </top> <bottom>
         <button action="test" command="test.bsh">Hit to Test</button>
        </bottom>
       </vsplit>
      </right>
     </hsplit>
    </center>
    <south>
     <label font="helvetica-normal-22">South</label>
    </south>
   </border-layout>
  </components>
 </ui>
 <object class="org.jpos.bsh.BSHAction" id="test" />
</qbean>
```

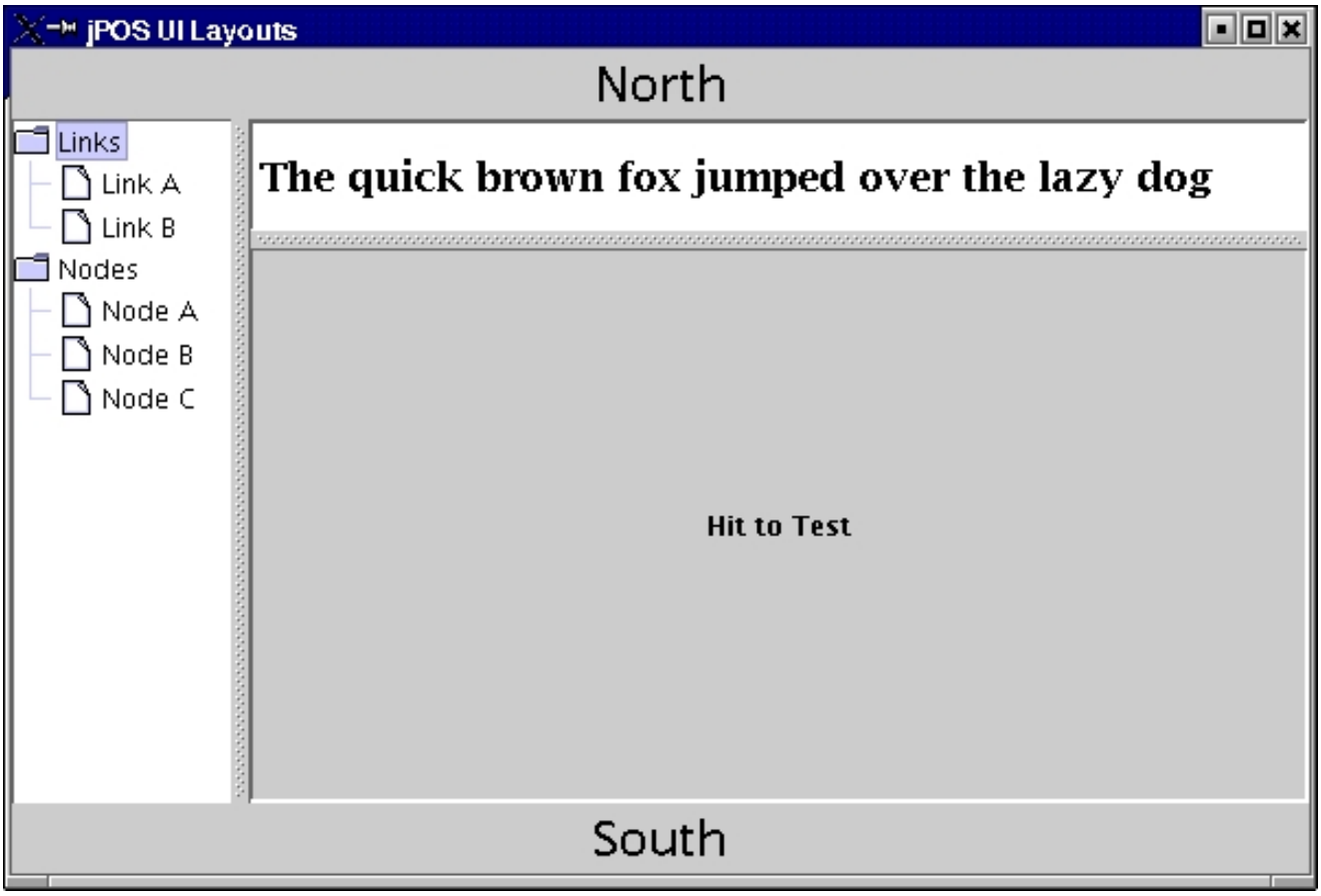


Figure 11.3. Layout example

The configuration file is self-explanatory: if you look at the button, it has an action, defined outside the <ui>...</ui> block. In this particular case, we use a special kind of action that lets you run an external script (here, `test.bsh`).

You can create a simple `test.bsh` file, e.g.:

```
print ("Hello jPOS UI Actions");
```

...in order to give it a try.

11.4.1. JLabelFactory

Description

Can be used to create and configure `JLabel` objects.

Usage

```
 <label>Simple Label</label>
 ...
 ...
 <label id="myLabel" font="arial-normal-22">My Label</label>
 ...
 ...
```

Table 11.3. Attributes

Attribute	Description
id	An optional id; can be used to get a reference to this component at a later time
font	Optional font
scrollable	if `true`, UI wraps the component with a `JScrollPane`
width	Width in pixels
height	Height in pixels

11.4.2. JButtonFactory

Description

Create and configure `JButton` objects.

Usage

```
<button>MyButton</label>
...
...
```

Table 11.4. Attributes

Attribute	Description
id	Optional id; can be used to get a reference to this component at a later time
font	Optional font
scrollable	if `true`, UI wraps the component with a `JScrollPane`
width	Width in pixels
height	Height in pixels
action	An optional action listener id.
command	An optional 'hand back' command string for the action listener.

11.4.3. TextFactory

Description

Can be used to create and configure `JEditorPane` objects.

Usage

```
<text>
 This text defaults to text/plain mimetype
</text>

<text id="MyPane" type="text/html">
 <![CDATA[
   <h1>this is an HTML EditorPane</h1>
 ]]>
</text>
```

Table 11.5. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
type	JEditorPane's mime type; defaults to `text/plain`, supports `text/html`.
editable	Either `true` or `false`; defaults to `false`.
width	Width in pixels
height	Height in pixels

11.4.4. HtmlFactory

Description

Can be used to create an embedded 'mini-browser'.

Usage

```
<html editable="false" follow-links="true" scrollable="true">
 http://jpos.org
</html>
```

Table 11.6. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
follow-links	Either `true` or `false`; defaults to `false`. Can be used to customize browser's behavior (e.g., if you want it to follow hyperlinks or not).
editable	Either `true` or `false`; defaults to `false`.

Attribute	Description
width	Width in pixels
height	Height in pixels

11.4.5. PanelFactory

Description

A Panel (with its `id`) can be used as a placeholder in order to replace its content at a later time.

You may want to create an empty `mainpanel`, and then **reconfigure** it later (e.g., see the `redirect` action).

Usage

```
<!-- empty mainpanel -->
<panel id="mainpanel" />

<!-- panel with initial content that can be replaced later -->
<panel id="otherpanel">
  <border-layout>
   <north>  ...  </north>
   <center> ... </center>
   <south>  ...  </south>
  </border-layout>
</panel>
```

Table 11.7. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
width	Width in pixels
height	Height in pixels

11.4.6. BorderLayoutFactory

Description

Creates a Panel with a BorderLayout manager and adds to it the optional UI components described inside child elements named `<north>`, `<center>`, `<south>`, `<east>` and `<west>`.

Usage

```
<border-layout>
 <north> ...  </north>
 <center> ...  </center>
 <south> ...  </south>
 <east> ...  </east>
```

```
     <west> ...  </west>
    </border-layout>
```

Table 11.8. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
width	Width in pixels
height	Height in pixels

11.4.7. GridLayoutFactory

Description

Creates a Panel with a GridLayout manager and adds to it the optional UI components described inside child elements named `<cell>`s.

Usage

```
    <grid rows="2" columns="2">
     <cell>...</cell> <cell>...</cell>
     <cell>...</cell> <cell>...</cell>
    </grid>
```

Table 11.9. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
rows	Number of rows
columns	Number of columns
width	Width in pixels
height	Height in pixels

11.4.8. HSplitFactory

Description

Horizontal split creates a `JSplitPane` and adds the child elements named `<left>` and `<right>`.

Usage

```
<hsplit divider="100">
 <left> ... </left>
 <right> ... </right>
</hsplit>
```

Table 11.10. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
divider	Set divider location.
width	Width in pixels
height	Height in pixels

11.4.9. VSplitFactory

Description

Vertical split creates a `JSplitPane` and adds the child elements named `<top>` and `<bottom>`.

Usage

```
<vsplit divider="100">
 <top> ..... </top>
 <bottom> ... </bottom>
</vsplit>
```

Table 11.11. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
divider	Set divider location.
width	Width in pixels
height	Height in pixels

11.4.10. JTabbedPaneFactory

Description

Can be used to create multiple, tabbed panes.

Usage

```
<tabbed-pane font="helvetica-bold-32">
  <pane title="Tab 1" action="myaction" command="mycommand">
    ...
    ...
  </pane>
  <pane title="Tab 2" action="myaction" command="mycommand">
    ...
    ...
  </pane>
</tabbed-pane>
```

Table 11.12. tabbed-pane attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
font	An optional font
width	Width in pixels
height	Height in pixels

Table 11.13. pane attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
title	The pane's title
action	An optional action listener id
command	An optional 'hand back' command string defined at the `node` level.
width	Width in pixels
height	Height in pixels

> **Tip**
>
> Components can be nested, so you can use a `grid` inside one of the `border-layout` components named (`south`, `north`, `center` ...), `vsplit`, `hsplit`, etc.

11.4.11. JTreeFactory

Description

Creates a JTree and adds nodes to id. (see `src/examples/ui/tree.xml`).

Usage

```
<tree width="120" height="281" action="myaction">jPOS
 <node>
   Channels
   <node command="acquirer A">Acquirer A</node>
   <node command="acquirer B">Acquirer B</node>
   <node>Acquirer C</node>
  </node>
  <node>
   MUXes
   <node>Acquirer A</node>
   <node>Acquirer B</node>
   <node>Acquirer C</node>
  </node>
  <node>
   Servers
   <node>Acquirer A</node>
   <node>Acquirer B</node>
   <node>Acquirer C</node>
  </node>
 </tree>
```

Table 11.14. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
width	Width in pixels
height	Height in pixels
action	An optional action listener id
command	Optional 'hand-back' command string defined at the `node` level

11.4.12. LogListenerFactory

Description

Creates a TextArea and adds a listener to a given logger, then displays upcoming events in that text area.

Usage

```
<log-listener scrollable="true" logger="Q2"
    font="fixed-normal-12"
     max-events="100" max-lines="50" />
```

Table 11.15. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
width	Width in pixels
height	Height in pixels
scrollable	if `true`, UI wraps the component with a `JScrollPane`
logger	The logger name (defaults to `Q2`)
max-events	Maximum number of events that you want to display
max-lines	Maximum number of lines to display

Example 11.1. Configuring a Swing based LogListener

```
<qbean class="org.jpos.q2.ui.UI" name="HelloUI" logger="Q2">
 <ui width="600" height="400"
    full-screen="false" undecorated="false" close="true">
  <caption>jPOS UI Test</caption>
  <components>
   <vsplit divider="300">
    <top>
     <html scrollable="true">http://jpos.org</html>
    </top>
    <bottom>
    <log-listener scrollable="true"
                 logger="Q2" font="fixed-normal-12"
                             max-events="100" max-lines="50" />
    </bottom>
   </vsplit>
  </components>
 </ui>
</qbean>
```

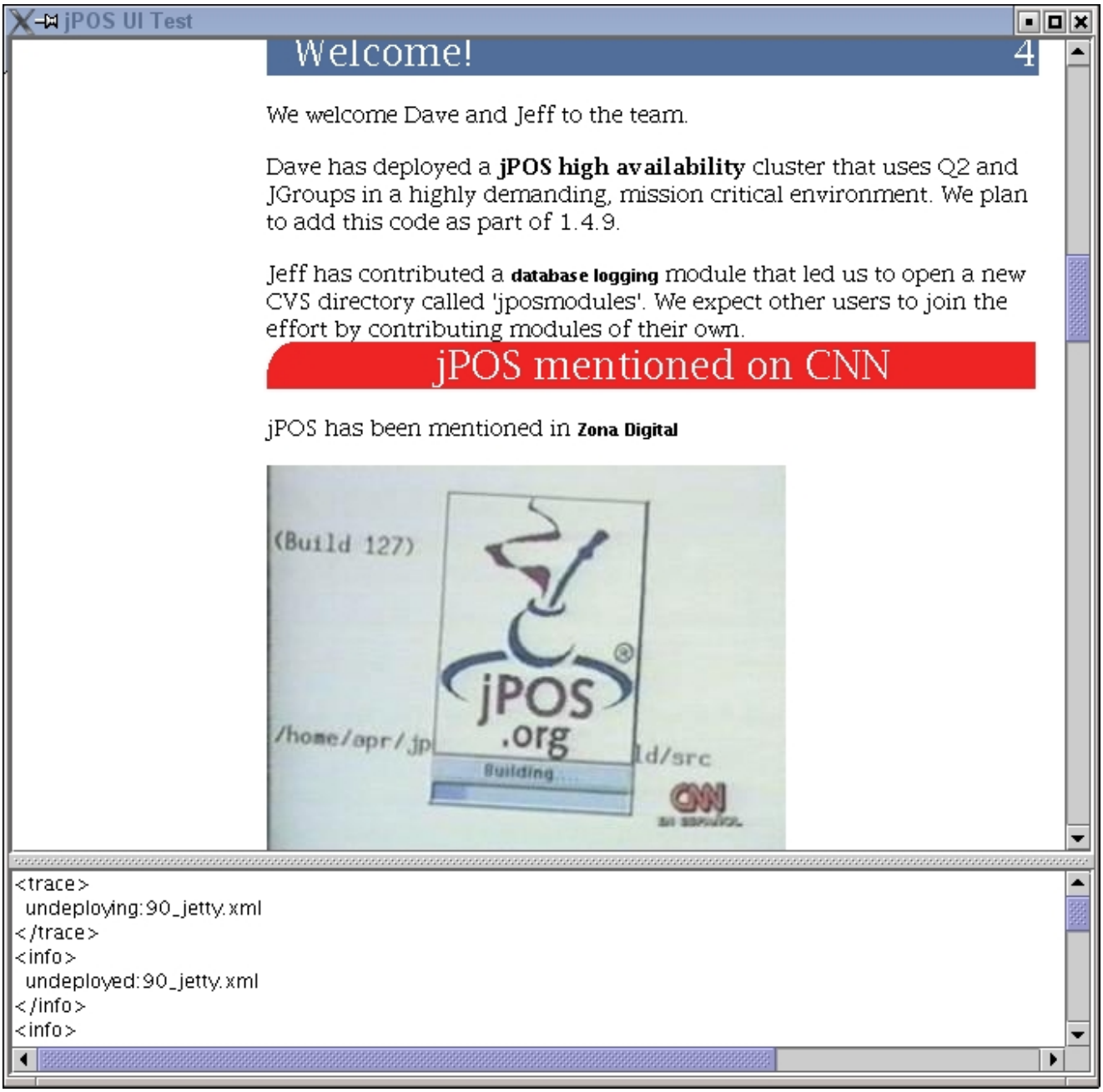


Figure 11.4. LogListener

11.4.13. ISOMeterFactory

Description

Creates an ISOMeter.

Usage

```
<iso-meter refresh="250" idref="channel.name">MyChannel</iso-meter>
```

Table 11.16. Attributes

Attribute	Description
id	Optional id; can be used to reference this component later.
scroll	Defaults to `true`, but can be set to `false` if the user wants to externally call the meter's scroll method.
refresh	Refresh rate in milliseconds (defaults to 50ms).

Example 11.2. ISOMeter UI configuration

```
<qbean class="org.jpos.q2.ui.UI" name="UI" logger="Q2">
 <ui width="600" height="400"
    full-screen="false" undecorated="true" close="true">
  <caption>jPOS Control Center</caption>

  <components>
   <border-layout>
    <center>
     <grid rows="2" columns="2">
      <cell>
       <iso-meter refresh="250"
          idref="channel.channel_A">Host A</iso-meter>
      </cell>
      <cell>
       <iso-meter refresh="250"
          idref="channel.channel_B">Host B</iso-meter>
      </cell>
      <cell>
       <iso-meter refresh="250"
          idref="channel.channel_C">Host C</iso-meter>
      </cell>
      <cell>
       <iso-meter refresh="250"
          idref="channel.channel_D">Host D</iso-meter>
      </cell>
     </grid>
    </center>
    <south>
     <label font="helvetica-bold-8">jPOS Control Center</label>
    </south>
   </border-layout>
  </components>
 </ui>
</qbean>
```

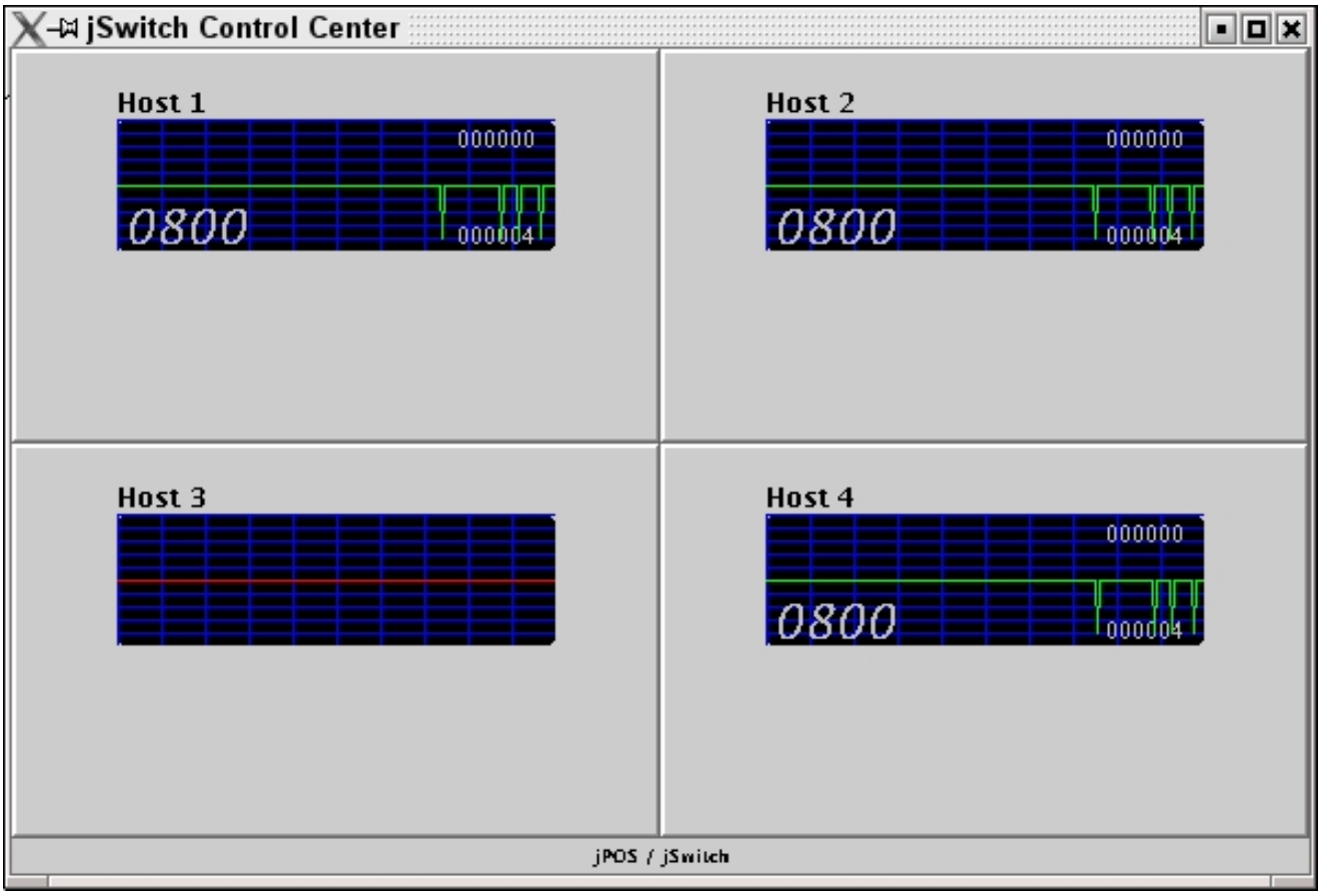


Figure 11.5. ISOMeters

Chapter 12. Filter Implementations

12.1. MD5Filter

The MD5Filter can be used to authenticate (i.e., sign and verify the signature of) ISO-8583 messages. It basically applies MD5 to an initial key and a set of relevant ISO-8583 fields (such as 2, 4, 7, 14, 41, 42).

MD5Filter is a cheap way (in terms of computing resources) to authenticate traffic in proprietary interchanges (i.e., where you have control over both ends).

Let's take a look at a simple example:

Example 12.1. MD5Filter configuration

```
<channel ... >
 ...
 ...
 <filter class="org.jpos.iso.filter.MD5Filter" direction="both">
  <property name="key" value="AnyKey" />
  <property name="fields" value="0 7 11 37 41" />
 </filter>
 ...
 ...
</channel>
```

Now suppose that with that MD5Filter configuration example in effect, we send this simple 0800 message over a channel:

```
<isomsg>
 <field id="0" value="0800" />
 <field id="2" value="4000000000000001" />
 <field id="3" value="000000" />
 <field id="11" value="000001" />
 <field id="35" value="4000000000000100000001=0203" />
 <field id="45" value="4000000000000100000001^POS, J POS^0201" />
 <field id="41" value="00000001" />
 <field id="48" value="01001" />
 <field id="55" value="123" />
 <field id="70" value="301" />
</isomsg>
```

An MD5 digest will be generated using the following components:

```
MD5("AnyKey","0800","000001","00000001") = "7EC81080B0946EF919BE532B4316F7AB"
```

[NOTE: Fields 7 and 37 are not present in the original message, so they are ignored.]

MD5Filter operates as an outgoing filter (by computing an MD5 digest and placing the split result into ISO fields 64 and 128) as well as an incoming filter (by verifying the contents of ISO fields 64 and 128). In our previous example, MD5Filter would produce the following ISO-8583 outgoing message:

```
<isomsg direction="outgoing">
  <field id="0" value="0800"/>
  <field id="2" value="4000000000000001"/>
  <field id="3" value="000000"/>
  <field id="11" value="000001"/>
  <field id="35" value="40000000000000100000001=0203"/>
  <field id="41" value="00000001"/>
  <field id="45" value="40000000000000100000001^POS, J POS^0201"/>
  <field id="48" value="01001"/>
  <field id="55" value="123"/>
  <field id="64" value="7EC81080B0946EF9" type="binary"/>
  <field id="70" value="301"/>
  <field id="128" value="19BE532B4316F7AB" type="binary"/>
</isomsg>
```

The MD5 digest (i.e., the contents of ISO fields 64 and 128) will be verified at the receiving end as part of the MD5 authentication process.

In order to be as transparent as possible, MD5Filter will unset fields 64 and 128 at the receiving end, so that the higher-level application or request listeners don't have to deal with them.

12.2. MacroFilter

MacroFilter is a simple yet useful filter that can be used to set ISOMsg's fields according to runtime properties, as well as to unset arbitrary group of fields.

Example 12.2. MacroFilter configuration

```
<channel ... >
 ...
 ...
 <filter class="org.jpos.iso.filter.MacroFilter" direction="outgoing">
  <property name="sequencer" value="sequencer" />
  <property name="srcid"  value="123456" />
  <property name="termid" value="29110001" />
  <property name="valid"  value="0 3 7 11 32 41" />
  <property name="unset"  value="42" />
 </filter>
 ...
 ...
</channel>
```

As an example, take the following ISOMsg:

```
 <isomsg>
  <field id="0"  value="0800" />
  <field id="7"  value="$date" />
```

```
    <field id="11" value="#trace" />
    <field id="32" value="$srcid" />
    <field id="41" value="$termid" />
    <field id="42" value="ToBeRemoved" />
    <field id="43" value="ToBeRemovedToo" />
   </isomsg>
```

After it passes through our MacroFilter (as configured in the previous example), the following output would be produced:

```
<isomsg>
 <field id="0"  value="0800" />
 <field id="7"  value="1103194659" />
 <field id="11" value="000001" />
 <field id="32" value="123456" />
 <field id="41" value="29110001" />
</isomsg>
```

MacroFilter interprets as follows:

- Properties:

  Fields starting with `$` are replaced by the property named after the `$` sign.

  There's one special case here to note: the literal `$date` expands to current date/time.

  Tip

  You can also specify a timezone, i.e: `$date GMT-03`

- Sequencers:

  Fields starting with `#` are interpreted as sequences. You can have as many sequences as you want within your application (e.g., `#trace`, `#mytrace`, `#counter`, etc.).

  You can specify a sequencer (such as ReliableSequencer) in order to preserve values across different runs. If you don't specify a sequencer (i.e., using the "sequencer" property), then MacroFilter will use a VolatileSequencer by default.

- Valid fields

  When you specify a set of valid fields, MacroFilter will take care of removing any additional field(s).

- Unset fields

  The `unset` property can be used to have MacroFilter remove fields from the list (any field not specified here would pass through).

12.3. XSLTFilter

XSLTFilter takes an ISOMsg, converts it to XML, applies XSL transformations to produce a new [modified] XML document and then returns the modifed ISOMsg.

XSLTFilter can read the XSLT configuration file at runtime, so it's very useful for simulation purposes and for changing things on the fly.

Example 12.3. XSLTFilter configuration

```
<channel ... >
 ...
 ...
 <filter class="org.jpos.iso.filter.XSLTFilter" direction="incoming">
  <property name="xsltfile" value="incoming-filter.xsl" />
  <property name="reread" value="no" />
 </filter>
 ...
 ...
</channel>
```

- xsltfile

  points to the XSL-T file

- reread

  When set to true, the XSL-T file is read everytime a message is processed (useful during tests/debuging/simulations).

Describing how to use XSLT to transform XML documents is beyond the scope of this guide. Here we have provided a simple annotated example:

Example 12.4.

```
<?xml version="1.0"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

 <xsl:template match="/isomsg">
  <isomsg>
   <!-- Force field 39 to "00" - use with care :) -->
   <field id="39" value="00" />
   <xsl:apply-templates />
  </isomsg>
 </xsl:template>

 <!-- handle inner isomsgs -->
 <xsl:template match="isomsg">
  <isomsg id="{@id}">
   <xsl:apply-templates />
  </isomsg>
 </xsl:template>

 <xsl:template match="field">
  <field>
    <xsl:attribute name="id">
     <xsl:value-of select="@id" />
```

```
    </xsl:attribute>
    <xsl:attribute name="value">
     <xsl:value-of select="@value" />
    </xsl:attribute>
   <xsl:if test="@type">
    <xsl:attribute name="type">
     <xsl:value-of select="@type" />
    </xsl:attribute>
   </xsl:if>
   <xsl:apply-templates/>
  </field>
 </xsl:template>

 <!-- Set response MTI for xx00 messages -->
 <xsl:template match="field[@id='0' and substring(@value,3,1)='0']">
  <field value="{substring(@value,1,2)}10" id="0" />
 </xsl:template>

 <!-- Set response MTI for xx20 messages -->
 <xsl:template match="field[@id='0' and substring(@value,3,1)='2']">
  <field value="{substring(@value,1,2)}30" id="0" />
 </xsl:template>

 <!-- set a dummy RRN on field 37 based on content from Field 11 -->
 <xsl:template match="field[@id='11']">
  <field value="RRN-{@value}" id="37" />
  <field value="{@value}" id="38" />
 </xsl:template>

 <!-- set field 39 based on processing code -->
 <xsl:template match="field[@id='3' and @value='920000']">
  <field id="39" value="95" />
 </xsl:template>
</xsl:transform>
```

Chapter 13. Legacy components

jPOS is a mature project about to celebrate its 10th aniversary. Along the way we have developed many components. Some of those have proven to be good to have and have been maintained and improved. Some others were good proof of concept that have been replaced by better implementations (i.e. QSP became Q2).

The following sections document legacy and deprecated components.

> **Tip**
> You don't want to use these components in new applications and you don't need to learn them. They are presented here just in case you are maintaining an old jPOS application.

13.1. QSP - jPOS' component assembler

13.1.1. About QSP

QSP, jPOS's component assembler

jPOS has quite a number of components that have to be configured at startup time, and then subsequently interconnected and monitored.

A typical jPOS application's main() would start by reading some sort of configuration file. It would instantiate a few components, start a few threads, initialize database connection pools, etc.

QSP is a standard way of doing the aforementioned tasks. It reads a standard XML configuration file and launches all the components for you.

1. Why the weird name?

I'd written QSP as a way to configure a jPOS-based gateway a few years ago. QSP was in charge of routing/forwarding ISO-8583 messages coming from various locations into several financial institutions.

QSP means **relay message for free**, (see Q-signals [http://odin.prohosting.com/n3uvr/q-sigs.shtml/#qsp]) which makes a lot of sense, as this particular application was in charge of relaying messages, and being jPOS a Free As In Beer [http://c2.com/cgi/wiki?FreeAsInBeer] OpenSource project the "for free" part was quite correct.

Anyway, I agree that QSP is a weird and nerdy name ... but that's what we have.

2. Why should you use it?

If you're relying on jPOS to build your applications, we strongly recommend that you use QSP.

> **Note**
> As of version 1.4.6, jPOS's QSP has evolved into a new framework called Q2 (i.e., kind of a QSP 'version 2'). QSP is still available and fully supported. For new applications, we suggest that you use Q2. In any case, the QSP concepts and documentation described here are very similar to those used by Q2, so we highly recommend that you continue reading this section.

Among other benefits, if you're a QSP user, upgrading to latest jPOS version is just a matter of dropping a new jpos.jar file in your CLASSPATH and tweaking a couple of lines in QSP's configuration file.

These little configuration blocks are very easy to read, can be easily shared via e-mail and can be used as building blocks for bigger applications.

Adding features such as JMX support is a matter of adding a few lines to your configuration file, lines that you can cut & paste from other applications, or take from examples and posts from the jpos-dev mailing list.

Another benefit is that when you have a problem, you can get better support if you simply send us your XML config file and possibly a couple of your supporting files rather than your whole application.

13.1.2. Installing QSP

1. Building

QSP comes pre-built within the dist/jpos-x.x.x.jar file. If you feel the need to rebuild it, you can try the following command (see ext/README file for details):

```
bin/build clean ext jar
```

2. Running QSP

2.1. Required files

It's very simple - create a new directory and place the following files there:

- jpos-x-x-x.jar (available in jpos/dist)
- jmxri.jar (available in jpos/lib)
- xerces_x_x_x.jar (available in jpos/lib)
- qsp-config.dtd (available in jpos/src/config/qsp)

2.2. Testing the waters

Try the following command:

```
java -jar jpos-x-x-x.jar
```

You should get the following output:

```
Usage: org.jpos.apps.qsp.QSP <configfile>
```

Now, let's create a simple configuration file so we can verify that QSP is properly installed. Write the following content in a file named `test.xml`:

```
<?xml version="1.0" ?>

<!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
<qsp-config reload="5000" logger="qsp" realm="test">
 <logger name="qsp">
    <log-listener class="org.jpos.util.SimpleLogListener" />
 </logger>
</qsp-config>
```

Although this might give you an idea of how QSP is configured, don't worry about the content of this little `test.xml` file just yet. We'll cover it in great detail in an upcoming section. Now run:

```
    java -jar jpos-x-x-x.jar test.xml
```

You should see output messages like the following:

```
<log realm="test" at="Mon Aug 19 18:11:58 UYT 2002.33">
  <configure>
    logger
    qsp-config
    log-listener
    object
    connection-pool
    persistent-engine
    sequencer
    s-m-adapter
    secure-key-store
    control-panel
    channel
    filter
    mux
    server
    request-listener
    card-agent
    dir-poll
    task
    daily-task
    -extended tags-
  </configure>
</log>
<log realm="monitor" at="Mon Aug 19 18:11:58 UYT 2002.291">
  <SystemMonitor>
    <memory>
       freeMemory=461344
      totalMemory=2031616
      inUseMemory=1571208
    </memory>
    <threads>
            delay=0 ms
          threads=5
            Thread[Reference Handler,10,system]
            Thread[Finalizer,8,system]
            Thread[Signal Dispatcher,10,system]
            Thread[CompileThread0,10,system]
```

```
            Thread[main,5,main]
            Thread[logger,5,main]
            Thread[Thread-2,5,main]
            Thread[SystemMonitor,1,main]
    </threads>
    <name-registrar>
      <name>logger.qsp</name>
      <class>org.jpos.util.Logger</class>
      <name>qsp.default</name>
      <class>org.jpos.apps.qsp.QSP</class>
    </name-registrar>
  </SystemMonitor>
</log>
```

Open another shell (window) and move away your `test.xml` file. QSP will shutdown, showing a couple of messages like this:

```
<log realm="test" at="Mon Aug 19 18:12:03 UYT 2002.303">
  <shutdown-start/>
</log>
<log realm="test" at="Mon Aug 19 18:12:03 UYT 2002.306">
  <shutdown/>
</log>
```

13.1.3. Configuration

The QSP configuration starts with:

```
    <?xml version="1.0" ?>
    <!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
    <qsp-config reload="5000" logger="qsp" realm="test">
      ...
      ...
      ...
    </qsp-config>
```

The "reload" attribute is used to have QSP monitor its configuration file for any changes. If something changes, QSP will reload the file and attempt to reconfigure its components.

`logger` and `realm` are optional attributes which enable you to assign a given logger to your QSP instance.

> **Tip**
>
> We strongly suggest that you assign a logger to `<qsp-config>` and to your QSP components; this suggestion especially holds true while you are debugging an application.

13.1.4. <object>

QSP <object> elements are used to instantiate and eventually configure and run user-defined Objects.

There are at least two ways to run QSP:

- Launch QSP from your application
- Launch your application from QSP

In order to start QSP from your application, you can add the following line to your initialization code:

```
import org.jpos.apps.qsp.QSP;
...
...
QSP.launch ("your-config-file.xml");
...
...
```

Tip

Though possible, it is not a good idea to instantiate more than one QSP in a given JVM.

In general, we strongly recommend the second approach, i.e., launching your application from QSP. This can be done easily by using <object> elements in your QSP xml-based configuration file.

First, create a "classes" directory in your QSP testbed tree and compile the following simple class:

```
public class MyObject {
    public MyObject() {
        super();
        System.out.println ("MyObject has been instantiated");
    }
}
```

Now add the following block to your QSP configuration file (from our previous example, `test.xml`):

```
<?xml version="1.0" ?>
<!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
<qsp-config>
 <object class="MyObject" />
</qsp-config>
```

Try to run:

```
java -jar jpos-x.x.x.jar test.xml
```

...and you'll see a message like this:

```
MyObject has been instantiated
```

You can define as many <object> elements as you want. As you might have guessed by now, at this point you have the opportunity to initialize the rest of your application.

Now let's have a look at the `qsp-config.dtd` related to the <object> element.

```
<!-- object -->
<!ELEMENT object (property)*>
<!ATTLIST object class NMTOKEN #REQUIRED>
```

```
    <!ATTLIST object logger IDREF #IMPLIED>
    <!ATTLIST object realm CDATA #IMPLIED>
    <!ATTLIST object name ID #IMPLIED>
    <!ATTLIST object connection-pool IDREF #IMPLIED>
```

1. Configurable object

You can add some <property> elements to your <object> as shown in the following example:

```
     <?xml version="1.0" ?>
     <!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
     <qsp-config>
      <object class="MyObject">
       <property name="MyProperty" value="MyValue" />
       <property name="AnotherProperty" value="AnotherValue" />
      </object>
     </qsp-config>
```

In order to be able to read these properties, our MyObject has to implement the `org.jpos.core.Configurable` interface. So let's modify it to read like this:

```
import org.jpos.core.Configurable;
import org.jpos.core.Configuration;

public class MyObject implements Configurable {
    public MyObject() {
        super();
        System.out.println ("MyObject has been instantiated");
    }
    public void setConfiguration (Configuration cfg) {
        System.out.println ("     MyProperty: "+cfg.get ("MyProperty"));
        System.out.println ("AnotherProperty: "+cfg.get ("AnotherProperty"));
    }
}
```

Compile it with the following command:

```
    javac -classpath jpos-x.x.x.jar -d classes MyObject.java
```

...and run it:

```
    java -jar jpos-x.x.x.jar test.xml
```

The output should look like this:

```
     MyObject has been instantiated
          MyProperty: MyValue
     AnotherProperty: AnotherValue
```

2. ReConfigurable Object

With a little change to our previous example, we can make our object ReConfigurable. It's just a `tag interface`. So instead of implementing Configurable, change it to implement ReConfigurable. The code would look like this:

```
import org.jpos.core.ReConfigurable;
import org.jpos.core.Configuration;

public class MyObject implements ReConfigurable {
    public MyObject() {
        super();
        System.out.println ("MyObject has been instantiated");
    }
    public void setConfiguration (Configuration cfg) {
        System.out.println ("     MyProperty: "+cfg.get ("MyProperty"));
        System.out.println ("AnotherProperty: "+cfg.get ("AnotherProperty"));
    }
}
```

Now modify `test.xml` and add a `reload` attribute to your `<qsp-config>` element, and a `name` attribute to your `<object>` element. You should end up with something that looks like this:

```
<?xml version="1.0" ?>

<!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
<qsp-config reload="5000">
 <object name="myobject" class="MyObject">
  <property name="MyProperty" value="MyValue" />
  <property name="AnotherProperty" value="AnotherValue" />
 </object>
</qsp-config>
```

Now rebuild your MyObject class with the command:

```
    javac -classpath jpos-x.x.x.jar -d classes MyObject.java
```

...and run it. The output should look the same.

While QSP is running, open another shell (window) and modify your configuration file; for example, try to change "MyValue" to read "MyNewValue". After a couple of seconds, you should see something like this:

```
         MyProperty: MyNewValue
    AnotherProperty: AnotherValue
```

Tip

In order to be notified of configuration changes, you have to set a unique `name` attribute to your `<object>`. QSP won't re-configure an unnamed object.

3. Locating your objects

If you set a `name` attribute, QSP will register your `<object>` using `NameRegistrar`. So in our previous example, you can easily locate your `"myobject"` object with a call to:

```
    MyObject myObject = (MyObject) NameRegistrar.get ("myobject");
```

In this particular case, (`<object>` element) the name used to register the object is the same name specified in the `name` attribute. This is not true for other elements. Instead, QSP assigns them a prefix, e.g., `<mux>` uses the `"mux."` prefix and `<task>` uses a `"task."` prefix. When in doubt, take a look at the source code (`src/ext/org/jpos/apps/qsp/config/Config*.java`) for the element configurators.

4. Implementing LogSource

If your Object implements LogSource, and you provide `logger` and `realm` attributes for your `<object>` element, then that particular logger (defined elsewhere in your QSP configuration file) will be assigned to your object.

Implementing LogSource is very easy. You can extend `SimpleLogSource` or (in the situation where you are already extending another class) you can add the following code to your class:

```
import org.jpos.util.Logger;
import org.jpos.util.LogSource;

public class .... implements LogSource {
    protected Logger logger;
    protected String realm;
    ...
    ...
    public void setLogger (Logger logger, String realm) {
        this.logger = logger;
        this.realm  = realm;
    }
    public String getRealm () {
        return realm;
    }
    public Logger getLogger() {
        return logger;
    }
}
```

So back to our MyObject example...modify it so it looks like this:

```
import org.jpos.core.ReConfigurable;
import org.jpos.core.Configuration;
import org.jpos.util.Logger;
import org.jpos.util.LogSource;
import org.jpos.util.LogEvent;

public class MyObject implements ReConfigurable, LogSource {
    protected Logger logger;
    protected String realm;
    Configuration cfg;

    public MyObject() {
        super();
        cfg = null;
    }
    public void setConfiguration (Configuration cfg) {
        LogEvent evt = new LogEvent (
```

```
                this, this.cfg == null ?  "configuration" : "re-configuration"
        );
        evt.addMessage ("     MyProperty: "+cfg.get ("MyProperty"));
        evt.addMessage ("AnotherProperty: "+cfg.get ("AnotherProperty"));
        Logger.log (evt);
        this.cfg = cfg;
    }

    public void setLogger (Logger logger, String realm) {
        this.logger = logger;
        this.realm  = realm;
    }
    public String getRealm () {
        return realm;
    }
    public Logger getLogger() {
        return logger;
    }
}
```

Now modify test.xml and add a `logger` and `realm` to your `<object>` element. The result should read like this:

```
<?xml version="1.0" ?>

<!DOCTYPE qsp-config SYSTEM "qsp-config.dtd">
<qsp-config reload="1000">
 <logger name="qsp">
    <log-listener class="org.jpos.util.SimpleLogListener" />
 </logger>
 <object name="myobject" class="MyObject" logger="qsp" realm="myobject">
  <property name="MyProperty" value="MyNewValue" />
  <property name="AnotherProperty" value="AnotherValue" />
 </object>
</qsp-config>
```

Give it a try! The output should look like this:

```
<log realm="myobject" at="Tue Aug 20 19:45:39 UYT 2002.314">
  <configuration>
         MyProperty: MyNewValue
    AnotherProperty: AnotherValue
  </configuration>
</log>
```

Now 'touch' or change the `test.xml` file and you'll see a `<re-configurable>` log event.

Our object is now `ReConfigurable` and a `LogSource`. What next? You need to make it runnable.

5. Runnable object

Simply add:

```
    public class MyObject implements ReConfigurable, LogSource, Runnable
    ...
    ...
```

```
    public void run () {
        for (;;) {
            Logger.log (new LogEvent (this, "run"));
            ISOUtil.sleep (1000);
        }
    }
    ...
    ...
```

...and you'll be all set. If QSP determines that your object implements Runnable, it will start it in a new thread.

Please note that all these interfaces are optional; you can implement them if you want, but you don't have to if they're not required by your application.

6. Shutting down

If you want to be notified when the system is going down, you may want to implement `org.jpos.core.Stopable`:

```
    public interface Stopable {
       public void shutdown ();
    }
```

We won't go into great detail here, but you get the idea.

7. JMX support

If you want to control or monitor your `<object>` using a JMX agent, you may want to make your Object a standard MBean. QSP will take care of registering it with an MBean server.

This support is a very simple thing to implement. Try to add the following interface:

```
public interface MyObjectMBean {
    public String getMyProperty ();
    public String getAnotherProperty ();
}
```

...and make your MyObject implement it by adding the following methods:

```
public class MyObject implements ReConfigurable, LogSource, MyObjectMBean {
    ...
    ...
    public String getMyProperty () {
        return cfg.get ("MyProperty");
    }
    public String getAnotherProperty () {
        return cfg.get ("AnotherProperty");
    }
}
```

Now add `jmxtools.jar` to your current directory (see `ext/README` for details) and add the following configuration block to your `test.xml` file:

```
```

```
<task class="org.jpos.apps.qsp.task.HttpAdaptor" logger="qsp" realm="http-adaptor">
  <property name="port"     value="8080" />
</task>
```

(Soon, you'll see the difference between `<object>` and `<task>`).

Run QSP and point your browser to `http://localhost:8080/`. You'll be able to play with JMX's HtmlAdaptor and click on "MyProperty" and "AnotherProperty" to see their values.

Figure 13.1. MyObjectMBean

13.1.5. <logger>

Let's recall our previous example from the section about the jPOS Logger:

```
...
Logger logger = new Logger();
logger.addListener (new SimpleLogListener (System.out));
...
```

Now, let's register our newly created logger into the NameRegistrar:

```
...
logger.setName ("qsp");
...
```

You can do exactly that by using the QSP configuration file. The configuration block would look like this:

```
<logger name="qsp">
   <log-listener class="org.jpos.util.SimpleLogListener" />
</logger>
```

If you look at qsp-config.dtd (see Appendix or the qsp-config.dtd file), you'll notice:

```
<!-- logger -->
<!ELEMENT logger (log-listener*)>
<!ATTLIST logger name ID #REQUIRED>

<!-- log-listener -->
<!ELEMENT log-listener (property*)>
<!ATTLIST log-listener class NMTOKEN #REQUIRED>
<!ATTLIST log-listener name  CDATA #IMPLIED>

<!-- property -->
<!ELEMENT property EMPTY>
<!ATTLIST property name  CDATA #REQUIRED>
<!ATTLIST property value CDATA #IMPLIED>
<!ATTLIST property file  CDATA #IMPLIED>
```

What does this mean?

It means that every `<logger>` element requires a `name` attribute and can have any number of nested `<log-listener>` elements.

A `<log-listener>` requires a `class` attribute, an optional `name` and any number of nested `<property>` elements, which in turn require a `name` attribute and also can have `value` or `file` attributes.

Our SimpleLogListener does not require any properties at all, but more sophisticated LogListener implementations do. Let's have a look at RotateLogListener. We have to specify where to place the log file, when to switch logs, what's the maximum log size, etc.

1. RotateLogListener

A RotateLogListener can be configured with a block like this:

```
   <log-listener class="org.jpos.util.RotateLogListener">
     <property name="file" value="/var/log/qsp.log" />
     <property name="window" value="86400" />
     <property name="copies" value="10" />
     <property name="maxsize" value="10000000" />
   </log-listener>
```

Now, we can add a RotateLogListener to our previous configuration:

```
...
...
<logger name="qsp">
   <log-listener class="org.jpos.util.SimpleLogListener" />
   <log-listener class="org.jpos.util.RotateLogListener">
     <property name="file" value="/var/log/qsp.log" />
     <!-- one log file every 24 hours -->
     <property name="window" value="86400" />

     <!-- keep 10 copies              -->
     <property name="copies" value="10" />

     <!-- rotate if size greater than  10MB -->
     <property name="maxsize" value="10000000" />
   </log-listener>
</logger>
...
...
```

2. ProtectedLogListener

A LogListener has the opportunity to tweak a given LogEvent so that the subsequent listeners in the list are called with a modified version of the event. This feature is used by `org.jpos.util.ProtectedLogListener` which takes care of protecting sensitive information (such as credit card numbers, PIN blocks, etc.)

So you can have a configuration like this:

```
 ...
 ...
 <logger name="qsp">
    <log-listener class="org.jpos.util.SimpleLogListener" />
    <log-listener class="org.jpos.util.ProtectedLogListener">
      <property name="protect" value="2 35 45 55" />
      <property name="wipe"    value="48" />
    </log-listener>
    <log-listener class="org.jpos.util.RotateLogListener">
      <property name="file" value="/var/log/qsp.log" />
      <property name="window" value="86400" />  <!-- one log file every 24 hours -->
      <property name="copies" value="10" />     <!-- keep 10 copies              -->
      <property name="maxsize" value="10000000" /> <!-- rotate if size greater than 10MB -->
```

```
    </log-listener>
  </logger>
  ...
  ...
```

In the previous example, log messages dumped to stdout will continue to contain full information (useful for debugging), while those messages written to disk will be pre-processed by `ProtectedLogListener`, which will convert an actual message like this:

```
<isomsg>
 <field id="0" value="0200"/>
 <field id="11" value="000001"/>
 <field id="18" value="0001"/>
 <field id="35" value="4123451234512345=020645678901234"/>
 <field id="41" value="29110001"/>
 <field id="42" value="00000001001"/>
 <field id="48" value="00001"/>
 <field id="49" value="840"/>
 <field id="60" value="jPOS unittest"/>
</isomsg>
```

...into a modified, protected message like this:

```
<isomsg>
 <field id="0" value="0200"/>
 <field id="11" value="000001"/>
 <field id="18" value="0001"/>
 <field id="35" value="412345______0005=0206___________"/>
 <field id="41" value="29110001"/>
 <field id="42" value="00000001001"/>
 <field id="48" value="[WIPED]"/>
 <field id="49" value="840"/>
 <field id="60" value="jPOS unittest"/>
</isomsg>
```

3. OperatorLogListener

OperatorLogListener is a very useful LogListener implementation which takes care of sending batches of log messages via SMTP. It is configured as follows:

```
<log-listener class="org.jpos.util.OperatorLogListener">
 <property name="jpos.operator.from" value="jpos-logger@cs.com.uy" />
 <property name="jpos.operator.to" value="qsp-logger@jpos.org" />
 <property name="jpos.operator.subject.prefix" value="[jPOS] " />
 <property name="jpos.operator.tags" value="SystemMonitor Operator" />
 <property name="jpos.operator.delay" value="60000" />
 <property name="jpos.mail.smtp.host" value="mail.jpos.org" />
</log-listener>
```

[NOTE: You have to add `mail.jar` as well as `activation.jar` to your CLASSPATH. Take a look at `ext/README` for details.]

In the previous example, OperatorLogListener will send e-mail messages when it receives LogEvents tagged as "SystemMonitor" or "Operator". Once the first message has been sent, OperatorLogListener will wait for a minute before sending the next message. If more LogEvents matching the given tag(s) are received during that period of time, OperatorLogListener will buffer them and send them using MIME e-mail (one MIME attachment per received event). This approach is very important because it prevents an e-mail storm from blitzing operator if something wrong is going on for an extended period.

Once you have configured an OperatorLogListener, it's very easy to send information and debugging messages to him/her using code like this:

Example 13.1. E-mailing your operator

```
...
...
if (!channel.isConnected()) {
    LogEvent evt = new LogEvent (this, "Operator");
    evt.addMessage ("Please note Channel XYZ has been disconnected");
    Logger.log (evt);
}
...
...
```

13.1.6. On-the-fly reconfiguration

As we've seen in the previous `MyObject` example, it is possible for a given component to implement `org.jpos.core.ReConfigurable` and receive calls to its `setConfiguration (Configuration)` method whenever something changes in QSP's configuration file.

That said, we're not doing magic [yet], as you cannot completely change a configuration file, nor can you add or remove components without restarting QSP. However, property changes for components that are already configured work just fine. These on-the-fly change capabilities are very useful in 24/7 installations.

This feature, when combined with QSP's JMX support, lets you create highly manageable applications.

13.1.7. <task>

QSP's `<task>s` are handled almost the same as `<object>s`, with the following minor differences:

- Order of initialization:

  `<task>s` are the last type of components to be initialized by QSP.

- NameRegistrar's prefix:

  `<task>s` are prefixed by the string `"task."`

The `qsp-config.dtd` related to the `<task>` element is shown here:

```
<!-- task -->
```

```
<!ELEMENT task (property)*>
<!ATTLIST task class NMTOKEN #REQUIRED>
<!ATTLIST task logger IDREF #IMPLIED>
<!ATTLIST task realm CDATA #IMPLIED>
<!ATTLIST task name ID #IMPLIED>
<!ATTLIST task connection-pool IDREF #IMPLIED>
```

As with `<object>`s, `<task>`s can be just simple Objects, or they can also implement:

- `LogSource`
- `[Re]Configurable`
- `Runnable`
- `Stopable`
- `*MBean`

13.1.8. <channel>

Channels are heavily used in most jPOS applications; QSP was designed to allow a channel configuration to represent that channel's interaction with other components, such as filters, servers and muxes.

Let's look at a sample configuration:

```
<channel name="channel"
        class="org.jpos.iso.channel.XMLChannel"
        packager="org.jpos.iso.packager.XMLPackager"
        type="client" connect="no" logger="qsp" realm="channel" >
 <property name="host" value="127.0.0.1" />
 <property name="port" value="8001" />
</channel>
```

It is very simple to add a little filter to our channel. We have to place the filter configuration as a child node:

```
<channel name="channel"
        class="org.jpos.iso.channel.XMLChannel"
        packager="org.jpos.iso.packager.XMLPackager"
        type="client" connect="no" logger="qsp" realm="channel" >
 <property name="host" value="127.0.0.1" />
 <property name="port" value="8001" />
 <filter class="org.jpos.iso.filter.MacroFilter" direction="outgoing">
  <property name="sequencer" value="sequencer" />
  <property name="srcid" value="123456" />
  <property name="valid" value="0 3 7 11 32 41" />
  <property name="unset" value="42" />
 </filter>
 <filter class="my.company.myFilter" direction="incoming">
   <property name="status" value="link.nac.mvdeo:UP" />
 </filter>
</channel>
```

```
```

The same goes for parent nodes like muxes and servers: it's very easy to configure a mux that uses this channel. The configuration would look like this:

```
<mux name="mux" logger="qsp" realm="mux" connect="yes">
 <channel name="channel"
         class="org.jpos.iso.channel.XMLChannel"
         packager="org.jpos.iso.packager.XMLPackager"
         type="client" connect="no" logger="qsp" realm="channel" >
  <property name="host" value="127.0.0.1" />
  <property name="port" value="8001" />
  <filter class="org.jpos.iso.filter.MacroFilter" direction="outgoing">
   <property name="sequencer" value="sequencer" />
   <property name="srcid" value="123456" />
   <property name="valid" value="0 3 7 11 32 41" />
   <property name="unset" value="42" />
  </filter>
  <filter class="my.company.myFilter" direction="incoming">
    <property name="status" value="link.nac.mvdeo:UP" />
  </filter>
 </channel>
</mux>
```

The `qsp-config.dtd` related to the `channel` element looks like this:

```
<!-- channel -->
<!ELEMENT channel (property*, filter*)*>
<!ATTLIST channel name ID #REQUIRED>
<!ATTLIST channel class NMTOKEN #REQUIRED>
<!ATTLIST channel connect (yes|no) 'no'>
<!ATTLIST channel logger IDREF #IMPLIED>
<!ATTLIST channel realm CDATA #IMPLIED>
<!ATTLIST channel header CDATA #IMPLIED>
<!ATTLIST channel packager CDATA #IMPLIED>
<!ATTLIST channel panel IDREF #IMPLIED>
<!ATTLIST channel type (client|server) #IMPLIED>
<!ATTLIST channel timeout CDATA #IMPLIED>
<!ATTLIST channel packager-config CDATA #IMPLIED>
<!ATTLIST channel packager-logger CDATA #IMPLIED>
<!ATTLIST channel packager-realm  CDATA #IMPLIED>
<!ATTLIST channel socket-factory  CDATA #IMPLIED>
```

Channels accept child <property> elements. These properties will be passed on as part of a Configuration object to the channel implementation (provided it implements `org.jpos.core.Configurable`).

The same goes for <filter> children elements. You can add as many filters as required. Here is an example:

Table 13.1. Channel attributes

Attribute	Description	Type
name	Used to register this channel in the NameRegistrar. A prefix `"channel."` will be used.	Required
class	Channel implementation class name	Required
connect	Either `yes` or `no`	Defaults to `yes`
logger	Optional reference to a logger	Optional
realm	Logger's realm	Optional
header	QSP would call your optional `channel.setHeader(String);`	Optional
packager	Default packager for this channel	Optional but usually used
panel	Optional reference to a panel; QSP will add a GUI observer.	Optional
type	`client` or `server`. Many channel implementations (i.e: those derived from `BaseChannel`) have the ability to operate either as a client or server channel.	Implied
timeout	Timeout in milliseconds	Optional
packager-config	Attribute forwarded to the underlying packager	Optional
packager-logger	Optional logging at the packager level	Optional
packager-realm	Logger's realm	Optional
socket-factory	Optional Socket Factory used, e.g., in SSL-based channels (see ISOServerSocketFactory and ISOClientSocketFactory)	Optional

13.1.8.1. <filter>

The `filter` element is used by QSP to configure `ISOFilters`. The DTD looks like this:

```
<!-- filter -->
<!ELEMENT filter (property*,calculate*,SQL*)*>
<!ATTLIST filter name ID #IMPLIED>
<!ATTLIST filter class NMTOKEN #REQUIRED>
<!ATTLIST filter direction (incoming|outgoing|both) 'both'>
```

Table 13.2. Filter attributes

Attribute	Description	Type
name	Used to register this filter in the NameRegistrar.	Optional
class	Channel implementation class name	Required
direction	either `incoming`, `outgoing` or `both`	Optional; defaults to `both`

13.1.8.1.1. Sample filter

DelayFilter is a very simple 'demo' filter sometimes useful during development. It simply imposes a delay on all traffic passing through it (either incoming and/or outgoing) thereby allowing you to simulate a remote host or network latency, or giving you the ability to inject a 'slow down' into your test so that you can 'eyeball' and interpret the message flow in real time.

DelayFilter requires a single property named `delay`, which specifies the desired delay in milliseconds:

Example 13.2. DelayFilter configuration

```
<channel ... >
 ...
 ...
 <filter class="org.jpos.iso.filter.DelayFilter" direction="incoming">
  <property name="delay" value="1000" />
 </filter>
 ...
 ...
</channel>
```

> **Note**
> Have a look at the **filter implementations** chapter for additional information about available filters

13.1.9. <mux>

QSP's `<mux>` blocks are used instantiate and configure `ISOMUX` objects.

The DTD fragment looks like this:

```
<!-- mux -->
<!ELEMENT mux (property*,channel,request-listener*)*>
<!ATTLIST mux name ID #REQUIRED>
<!ATTLIST mux logger IDREF #IMPLIED>
<!ATTLIST mux realm CDATA #IMPLIED>
<!ATTLIST mux connect (yes|no) 'yes'>
<!ATTLIST mux class NMTOKEN #IMPLIED>
```

```
```

This feature requires: a unique `name` attribute that is used to register the `ISOMUX` with the NameRegistrar; an optional `logger/realm` pair; and an optional `class` attribute that you can use if you've created a custom ISOMUX (the class defaults to the `org.jpos.iso.ISOMUX` class).

A typical `<mux>` is represented here:

```
  <mux ...>
    <channel ... >
      ...
      ...
      <filter ... />
      <filter ... />
      ...
      ...
    </channel>
    <request-listener ... />
  </mux>
```

e.g.:

```
<mux name="mux" logger="qsp" realm="mux" connect="yes">
 <property name="tracenofield" value="11" />
 <channel name="channel"
         class="org.jpos.iso.channel.CSChannel"
      packager="org.jpos.iso.packager.GenericPackager"
      packager-config="cfg/iso93binary.xml"
        logger="qsp" realm="channel">
  <property name="host" value="192.168.0.1" />
  <property name="port" value="8000" />
 </channel>
 <request-listener
   class="org.jpos.apps.qsp.RequestDispatcher"
   logger="qsp" realm="dispatcher">
   <property name="prefix" value="my.company.jpos.Handler_" />
   <property name="timeout" value="60000" />
 </request-listener>
</mux>
```

If you want to obtain a reference to the QSP-instantiated MUX, you can do so easily by calling the `ISOMUX.getMUX (String name)` static method.

The optional attribute `connect` is honored by QSP's on-the-fly reconfiguration. As a result, you can easily disconnect a running mux by setting the `connect="no"` attribute.

Tip

Before changing a channel's host/port, you have to disconnect your mux and give QSP a chance to reload its configuration (see qsp-config's `reload` attribute).

13.1.10. <server>

QSP's `<server>` blocks are used to instantiate and configure `ISOServer` objects.

The DTD fragment looks like this:

```
<!-- server  -->
<!ELEMENT server (property*,channel,request-listener*)*>
<!ATTLIST server name ID #REQUIRED>
<!ATTLIST server port CDATA #REQUIRED>
<!ATTLIST server logger IDREF #IMPLIED>
<!ATTLIST server realm CDATA #IMPLIED>
<!ATTLIST server maxSessions CDATA #IMPLIED>
<!ATTLIST server panel IDREF #IMPLIED>
<!ATTLIST server scroll (yes|no) 'no'>
<!ATTLIST server refresh CDATA #IMPLIED>
<!ATTLIST server thread-pool IDREF #IMPLIED>
```

It requires a unique `name` attribute which is used to register our `ISOServer` instance with the central NameRegistrar as well as with a `port` number where ISOServer will listen for incoming connections.

As many other components, the server element supports an optional `logger/realm` pair.

A typical `<server>` configuration is represented here:

```
  <server ... port="nnnn" ...>
    <channel ... >
      ...
      ...
      <filter ... />
      <filter ... />
      ...
      ...
    </channel>
    <request-listener ... />
  </server>
```

i.e.:

```
<server name="nac.server" logger="qsp" realm="nac.server" port="5000">
 <channel name="nac.channel"
         class="org.jpos.iso.channel.NACChannel"
         packager="org.jpos.iso.packager.ISO87BPackager"
         type="server" logger="qsp" realm="nac.server.channel"
         header="6000000000" timeout="300000">
 </channel>
 <request-listener
   class="org.jpos.apps.qsp.RequestDispatcher"
   logger="qsp" realm="nac.dispatcher">
   <property name="prefix" value="com.mycompany.jpos.Incoming_" />
 </request-listener>
</server>
```

In order to get a reference to the QSP instantiated ISOServer, you can call the `ISOServer.getServer(String name)` static method.

The optional attribute `maxSessions` is used to specify an upper bound to the number of simultaneous connec-

tions handled by this server instance.

> **Note**
> jPOS/ISOServer uses a ThreadPool in order to accept simultaneous connections. Every thread in the pool handles incoming messages, and then forwards them to the registered ISORequestListener(s) in a synchronous way (i.e., single-threaded). It is the responsibility of the ISORequestListener to use another ThreadPool in order to handle simultaneous, long-lived transactions.

You can also assign a panel (using the `panel` attribute) that will monitor incoming/outgoing traffic.

13.1.11. <panel> and <control-panel>

ISOChannels, ISOMUXes and ISOServers are observable objects that can be observed by ISOMeter components. In order to lay out these ISOMeters in a grid, QSP supports a couple of very simple configuration blocks, namely `<panel>` and `<control-panel>`.

The DTD fragment looks like this:

```
<!-- control-panel -->
<!ELEMENT control-panel (panel)*>
<!ATTLIST control-panel rows CDATA #REQUIRED>
<!ATTLIST control-panel cols CDATA #REQUIRED>

<!-- panel -->
<!ELEMENT panel EMPTY>
<!ATTLIST panel name ID #REQUIRED>
```

A `control-panel` has `rows` and `cols` attributes that are used to specify the number of rows and columns in the grid. Inside the `control-panel` you have to place an appropiate number of named `panel` elements, e.g.:

```
<control-panel rows="4" cols="4">
 <panel name="P1" />
 <panel name="P2" />
 <panel name="P3" />
 <panel name="P4" />
 <panel name="P5" />
 <panel name="P6" />
 <panel name="P7" />
 <panel name="P8" />
</control-panel>
```

In the previous example, we have created eight panels (P1 to P8). These panel names can be used in channels, muxes and servers using the `panel` attribute, e.g.:

```
  <server name="NAC-1" logger="qsp" realm="server" port="8001" panel="P1">
   ...
   ...
  </server>
   ...
   ...
  <server name="NAC-2" logger="qsp" realm="server" port="8002" panel="P2">
   ...
```

```
  ...
</server>
```

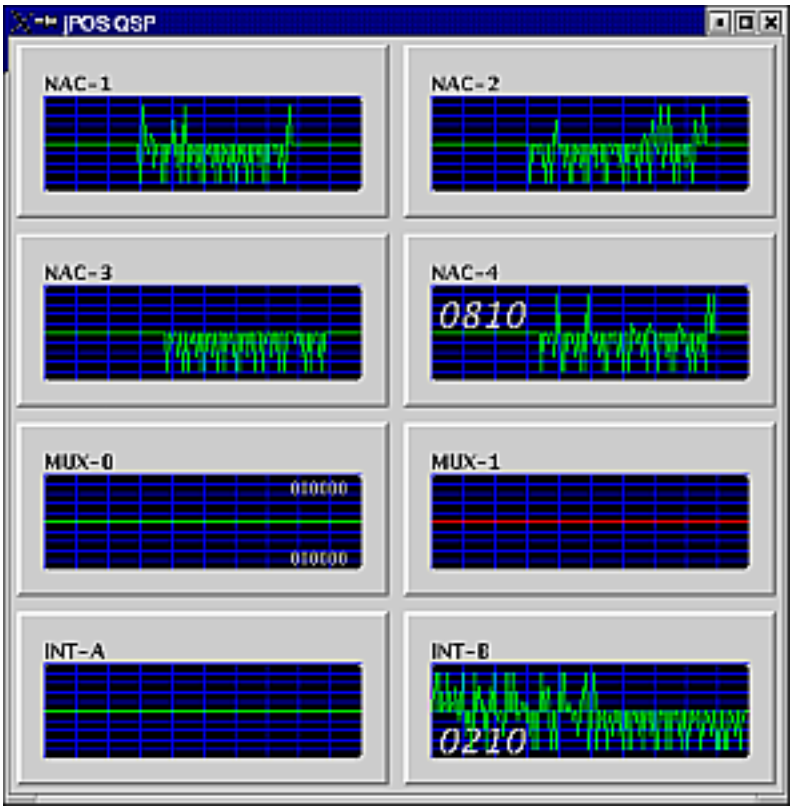


Figure 13.2. QSP Control Panel

13.1.12. <thread-pool>

Several components (such as the ISOServer and ISORequestListener implementations) rely on the use of ThreadPool in order to control system load.

Although you can create as many ThreadPools as you want, we have found it most convenient under certain situations that you share the same pool(s) among several components.

With QSP, you can configure and instantiate ThreadPool(s) and store them in the NameRegistrar so that other components can access them.

Extracted from qsp-config.dtd, the thread-pool fragment looks like this:

```
<!-- thread-pool -->
<!ELEMENT thread-pool (property)*>
<!ATTLIST thread-pool name ID #REQUIRED>
<!ATTLIST thread-pool logger IDREF #IMPLIED>
<!ATTLIST thread-pool realm CDATA #IMPLIED>
<!ATTLIST thread-pool max-size NMTOKEN '100'>
<!ATTLIST thread-pool initial-size NMTOKEN '1'>
```

Table 13.3. ThreadPool attributes

Attribute	Description
name	A unique name to be used to register this ThreadPool with the NameRegistrar. The name will be automatically prefixed by the string `"thread.pool."` in order to preserve namespaces.
logger	An optional reference to a logger
realm	Logger's realm
max-size	Maximum number of threads handled by this pool
initial-size	Initial number of threads

Example 13.3. QSP ThreadPool

```
<thread-pool name="MyPool" logger="qsp" realm="system"
  max-size="40" initial-size="10" />
```

13.1.13. <sequencer>

QSP's `<sequencer>`s blocks are used to instantiate and configure `org.jpos.core.Sequencer` implementations.

Rhe `qsp-config.dtd` related to the `<sequencer>` element is shown here:

```
<!-- sequencer -->
<!ELEMENT sequencer (property)*>
<!ATTLIST sequencer name ID #REQUIRED>
<!ATTLIST sequencer class NMTOKEN #REQUIRED>
<!ATTLIST sequencer logger IDREF #IMPLIED>
<!ATTLIST sequencer realm CDATA #IMPLIED>
```

If you recall our explanation of `<object>`s configuration block, these attributes may sound familiar. You see we have a `name` used to register our sequencer into the `NameRegistrar` (with a "sequencer." prefix), a `class` attribute used to instantiate any given `Sequencer` implementation, as well as a `logger` and a `realm`.

Example 13.4. VolatileSequencer

```
<sequencer name="volatile"
           class="org.jpos.core.VolatileSequencer"
           logger="qsp" realm="my-volatile-sequencer" />
```

```
```

In order to get a reference to our sequencer, we use code like this:

```
    import org.jpos.core.Sequencer;
    ...
    ...
    Sequencer seq = (Sequencer) NameRegistrar.get ("sequencer.volatile");
    ...
    ...
```

Example 13.5. ReliableSequencer

```
<sequencer name="reliable"
           class="org.jpos.core.ReliableSequencer"
           logger="qsp" realm="my-reliable-sequencer" >
 <property name="logdir" value="log/sequencer" />
</sequencer>
```

13.1.14. <connection-pool>

As with many other jPOS components, ConnectionPool can be configured by means of a QSP configuration block.

The DTD looks like this:

```
<!-- connection-pool -->
<!ELEMENT connection-pool (property)*>
<!ATTLIST connection-pool name ID #REQUIRED>
<!ATTLIST connection-pool logger IDREF #IMPLIED>
<!ATTLIST connection-pool realm CDATA #IMPLIED>
```

So it basically has a `name` that would be prefixed by the string `"connection.pool."` and would accept some configuration properties, namely:

Table 13.4. ConnectionPool properties

Property	Description
jdbc.driver	Driver class name (e.g., `"org.postgresql.Driver"`)
jdbc.url	JDBC connection's URL (e.g., `"jdbc:postgresql:jpos"`)
jdbc.user	Username
jdbc.password	Password

Property	Description
initial-connections	Number of initial JDBC connections
max-connections	Maximum number of JDBC connections handled by this pool
wait-if-busy	([yes|no|true|false]) If wait-if-busy is true and there are no available connections in the pool, a call to getConnection() would block until a new connection is released and placed back in the pool by another thread.

Example 13.6. QSP ConnectionPool

```
<connection-pool name="db-connector" logger="logger" realm="connection-pool">
  <property name="jdbc.driver" value="org.gjt.mm.mysql.Driver" />
  <property name="jdbc.url"
            value="jdbc:mysql://localhost:3306/test?autoReconnect=true" />
  <property name="jdbc.user" value="jpos" />
  <property name="jdbc.password" value="EasyToGuess" />
  <property name="initial-connections" value="1" />
  <property name="max-connections" value="20" />
  <property name="wait-if-busy" value="true" />
</connection-pool>
```

13.1.15. <persistent-engine>

`<persistent-engine>` is used to configure `org.jpos.tpl.PersistentEngine` objects.

The DTD looks like this:

```
<!-- persistent-engine -->
<!ELEMENT persistent-engine (property)*>
<!ATTLIST persistent-engine name ID #REQUIRED>
<!ATTLIST persistent-engine logger IDREF #IMPLIED>
<!ATTLIST persistent-engine realm CDATA #IMPLIED>
<!ATTLIST persistent-engine class NMTOKEN #IMPLIED>
```

`name` is prefixed by the literal `"persistent.engine."` so that you can get a reference to a PersistentEngine instance with code like this:

```
   ...
   ...
   try {
       ...
       PersistentEngine engine = (PersistentEngine)
       NameRegistrar.get ("persistent.engine.my-engine-name");
   } catch (NameRegistrar.NotFoundException e) {
```

```
        ...
        ...
    }
    ...
    ...
```

It accepts optional `logger` and `realm` attributes, as well as `class` (which defaults to `org.jpos.tpl.PersistentEngine`.

Table 13.5. PersistentEngine properties

Property	Description
jdbc.driver	Driver class name (e.g., `"org.postgresql.Driver"`)
jdbc.url	The JDBC connection's URL (e.g., `"jdbc:postgresql:jpos"`)
jdbc.user	Username
jdbc.password	Password

Note

As of version 1.4.4, jPOS's PersistentEngine does not leverage its own ConnectionPool (which was contributed recently). It's in our To-Do list to add that. The good news is that the change, when it comes, will be transparent to most users.

Example 13.7. QSP PersistentEngine

```
<persistent-engine name="jposdb" logger="qsp" realm="persistent-engine">
 <property name="jdbc.driver" value="org.gjt.mm.mysql.Driver" />
 <property name="jdbc.url"
          value="jdbc:mysql://localhost:3306/jpos?autoReconnect=true" />
 <property name="jdbc.user" value="jpos" />
 <property name="jdbc.password" value="DifficultToGuess" />
</persistent-engine>
```

13.1.16. <dir-poll>

The <dir-poll> element is used to configure a `DirPoll` object.

Here is dir-poll's DTD:

```
<!-- directory-poll -->
<!ELEMENT dir-poll (property)*>
<!ATTLIST dir-poll name ID #IMPLIED>
<!ATTLIST dir-poll path CDATA #REQUIRED>
<!ATTLIST dir-poll create (yes|no) 'no'>
<!ATTLIST dir-poll poolsize CDATA #IMPLIED>
```

```
<!ATTLIST dir-poll interval CDATA #IMPLIED>
<!ATTLIST dir-poll priorities CDATA #IMPLIED>
<!ATTLIST dir-poll processor CDATA #REQUIRED>
<!ATTLIST dir-poll logger IDREF #IMPLIED>
<!ATTLIST dir-poll realm  CDATA #IMPLIED>
```

Table 13.6. DirPoll attributes

Attribute	Description
name	A unique name to be used to optionally register DirPoll with the NameRegistrar. The name will be automatically prefixed by the string `"dir.poll."` in order to preserve namespaces.
logger	An optional reference to a logger
realm	Logger's realm
path	The directory path where DirPoll should poll for requests
create	A "yes" value here indicates that you want jPOS to create the directory structure required by DirPoll (request/response/tmp/bad/run) if it doesn't exist. This attribute defaults to "no".
poolsize	Indicates the maximum number of concurrent requests that are to be delegated to DirPoll's processor simultaneously.
interval	Poll time expressed in milliseconds.
priorities	An optional attribute consisting of a space-separated list of file extensions to poll first.
processor	A class implementing either org.jpos.util.DirPoll.Processor or org.jpos.util.DirPoll.FileProcessor interface.

Example 13.8. QSP DirPoll

```
<dir-poll path="/tmp/dirpoll" create="yes" name="test"
         logger="qsp" realm="dirpoll"
         poolsize="10"
         interval="1000"
         priorities=".A .B .C *"
         processor="qsp.dummydirpoll.DummyProcessor" />
```

Tip

Take a look at src/ext-examples/qsp/dummydirpoll.

13.1.17. <daily-task>

`<daily-task>`s are simple runnable objects that QSP takes care of calling at a given time of day on a daily basis. Its configuration block is very similar to that of a `<task>` and looks like this:

```
<daily-task name="db.cleanup"
   class="org.jpos.ee.batch.Cleanup"
   logger="qsp" realm="database-cleanup"
   start="06:00:00" poolsize="1">
 <property name="myProperty" value="myValue" />
</daily-task>
```

The `qsp-config.dtd` related to the `<daily-task>` element is shown here:

```
<!-- daily-task -->
<!ELEMENT daily-task (property)*>
<!ATTLIST daily-task class NMTOKEN #REQUIRED>
<!ATTLIST daily-task logger IDREF #IMPLIED>
<!ATTLIST daily-task realm CDATA #IMPLIED>
<!ATTLIST daily-task name ID #IMPLIED>
<!ATTLIST daily-task start CDATA #REQUIRED>
<!ATTLIST daily-task poolsize CDATA #IMPLIED>
```

Note

A daily task has 24 hours to run, so a poolsize="1" is the recommended value; however, you may want to increase that value in certain situations or operating environments.

13.1.18. <card-agent>

QSP provides a handy way to instantiate, configure and register your `CardAgent` implementations under the `CardSelector` using a DTD like this:

```
<!-- card-agent -->
<!ELEMENT card-agent (property)*>
<!ATTLIST card-agent class NMTOKEN #REQUIRED>
<!ATTLIST card-agent logger IDREF #IMPLIED>
<!ATTLIST card-agent realm CDATA #IMPLIED>
<!ATTLIST card-agent persistent-engine CDATA #IMPLIED>
```

A typical configuration block would look like this:

```
<card-agent class="org.jpos.core.agents.Uruguay.VISANetAgent"
            logger="qsp" realm="visanet.agent"
            persistent-engine="jposdb">
 <property name="mux"                value="visanet" />
 <property name="sequencer"          value="default-sequencer" />
```

```
 <property name="timeout"             value="20000" />
 <property file="cfg/diners.cfg" />
</card-agent>

<card-agent class="org.jpos.core.agents.Uruguay.DinersAgent"
            logger="qsp" realm="diners.agent"
            persistent-engine="jposdb">
 <property name="mux"                 value="diners" />
 <property name="sequencer"           value="default-sequencer" />
 <property name="timeout"             value="20000" />
 <property file="cfg/diners.cfg" />
</card-agent>
...
...
```

Note

CardAgents encapsulate the business logic required to connect with acquiring institutions by creating an abstraction layer that uses high level contracts defined by the `org.jpos.core.CardAgent` interface.

13.1.19. Security Module

1. <s-m-adapter>

The Security Module adapter can be configured using an <s-m-adapter> configuration block, which is governed by the following DTD:

```
<!-- s-m-adapter -->
<!ELEMENT s-m-adapter (property)*>
<!ATTLIST s-m-adapter class NMTOKEN #REQUIRED>
<!ATTLIST s-m-adapter logger IDREF #IMPLIED>
<!ATTLIST s-m-adapter realm CDATA #IMPLIED>
<!ATTLIST s-m-adapter name ID #IMPLIED>
```

jPOS provides a software-based security module adapter implementation called `org.jpos.security.jceadapter.JCESecurityModule`. This adapter can be used to simulate a hardware-based Tamper-Resistant Security Module ('TRSM') in software.

Note

`JCESecurityModule` will probably be renamed to `SSM` (which stands for Software Security Module) starting in 1.4.9. SSM will provide support for ANSI X9.24 DUKPT.

2. <secure-key-store>

KeyStores are plain-text files used to store secure keys already encrypted by a security module under its local master keys.

The QSP config DTD is described here:

```
<!-- secure-key-store -->
<!ELEMENT secure-key-store (property)*>
<!ATTLIST secure-key-store class NMTOKEN #REQUIRED>
<!ATTLIST secure-key-store logger IDREF #IMPLIED>
<!ATTLIST secure-key-store realm CDATA #IMPLIED>
<!ATTLIST secure-key-store name ID #IMPLIED>
```

Using QSP, you can define several key stores and then locate them using jPOS NameRegistrar.

Example 13.9. Software-based security module configuration

```
<s-m-adapter class="org.jpos.security.jceadapter.JCESecurityModule"
       logger="qsp" realm="sm" name="sm">
 <property name="provider" value="com.sun.crypto.provider.SunJCE" />
 <property name="lmk" value="cfg/lmk-test" />
</s-m-adapter>

<secure-key-store name="ks" class="org.jpos.security.SimpleKeyFile"
       logger="qsp" realm="ks">
  <property name="key-file" value="cfg/keys-test"/>
</secure-key-store>
```

13.1.20. Sequencers

Applications usually require some kind of sequence of numbers, and ISO-8583-based applications are no exception. We need a sequence of numbers to uniquely identify messages (i.e. trace number, retrieval reference numbers, voucher numbers).

jPOS comes with a `Sequencer` and a couple of implementations, namely:

- VolatileSequencer
- ReliableSequencer

```
package org.jpos.core;
public interface Sequencer {
    public int get (String counterName);            // increments by 1
    public int get (String counterName, int add);   // increments by 'add'
    public int set (String counterName, int value); // sets a given value
}
```

`VolatileSequencer`, as the name implies, is a memory-based lightweight sequencer. All information is lost whenever you restart your JVM.

By contrast, `ReliableSequencer` stores all sequence state information in a reliable way (journal/snapshot technique) on disk.

Example 13.10. VolatileSequencer

The following example shows how a Sequencer can be used to send echo messages.

```
Sequencer seq = new VolatileSequencer ();
ISOChannel channel = ....
for (;;) {
    ISOMsg m = new ISOMsg ();
    m.setMTI ("0800");
    m.set (3, "000000");
    m.set (11, Integer.toString (seq.get ("traceno") % 1000000));
    m.set (70, "000");
    ...
    ...
    channel.send (m);
    ...
    ...
    Thread.sleep (60000);
}
```

In this scenario, you would typically use a sequencer name associated somehow with your given channel (i.e.: "acquirer-XYZ-link" instead of "traceno").

13.1.21. BlockingQueue

org.jpos.util.BlockingQueue, as the name implies, is a very simple, in-memory blocking queue.

It's used internally by other components such as ThreadPool, LoopbackChannel, etc. You might use it for your own purposes as well.

Most used methods are shown below: (please have a look at the source code or javadocs [http://www.jpos.org/doc/javadoc/org/jpos/util/BlockingQueue.html] for full list of available methods)

```
public synchronized void enqueue (Object o);
public synchronized void requeue (Object o);
public synchronized Object dequeue();
public synchronized Object dequeue (long timeout);
```

Tip

As of jPOS version 2.0 and beyond, you can use a more sophisticated way of synchronizing threads than this BlockingQueue. Take a look at `org.jpos.space` package.

Example 13.11. BlockingQueue

LoopbackChannel uses a BlockingQueue to support its implementation; a heavily simplified source is shown below:

```
public class LoopbackChannel {
    BlockingQueue queue;
    public LoopbackChannel () {
        super();
        queue = new BlockingQueue();
```

```
        }
        ...
        ...
        public void send (ISOMsg m) {
            queue.enqueue (m);
        }
        ...
        ...
        public ISOMsg receive() {
            return (ISOMsg) queue.dequeue();
        }
        ...
        ...
    }
```

13.1.22. LockManager

org.jpos.util.LockManager is a very simple way of locking things for a given period of time.

The LockManager interface is very simple:

```
    public interface LockManager {
        public Ticket lock (String resourceName, long duration, long wait);
    }
```

You call its lock method and you get either a Ticket or null.

LockManager.Ticket interface looks like this:

```
    public interface Ticket {
        public boolean renew (long duration);
        public long getExpiration();
        public boolean isExpired();
        public void cancel();
    }
```

LockManager can be very useful when coding ISO-8583-based applications. Under certain circumstances you may need to lock a given terminal for a given period of time (e.g., when settlement/reconcilliation is taking place) and then renew that lock (e.g., after each ISO-8583 request/response takes place).

As of jPOS 1.4.x we provide an in-JVM LockManager implementation called **org.jpos.util.SimpleLockManager**. Future versions might include a distributed multi-JVM capable LockManager (probably based in the new `org.jpos.space` architecture).

O/R mapping is not one of jPOS project's goals, so you should not expect TPL to evolve too much in the future. There are plenty of excellent O/R mapping projects and technologies out there, such as Castor [http://castor.exolab.org], Hibernate [http://sf.net/projects/hibernate], JDO, EJB's BMP/CMP, etc.

Regardless, our TPL is small, it is fast, and it works. As a reuslt, many developers have choosen it over more powerful alternatives.

13.1.23. PersistentPeer

TPL uses the peer pattern. Every class that has to be persisted requires a Peer that takes care of its life cycle.

```
    public interface PersistentPeer {
        public void setPersistentEngine (PersistentEngine engine);
        public void create (Object obj) throws SQLException;
        public void load   (Object obj) throws SQLException, NotFoundException;
        public void update (Object obj) throws SQLException, NotFoundException;
        public void remove (Object obj) throws SQLException, NotFoundException;
    }
```

(See javadocs [http://jpos.org/doc/javadoc/org/jpos/tpl/PersistentPeer.html] for details).

Let's look at a real world example. Here is a CaptureEntry where we manage POS authorization transactions:

Example 13.12. PersistentPeer - CaptureEntry

```
public class CaptureEntry implements Loggeable {
    long oid, parentOID;
    int  flags;
    int agentid, cardid;
    String localTerminal, remoteTerminal;
    Date transactionDate, txDate, closedDate, originalDate;
    CardHolder cardHolder;
    BigDecimal amount, additionalAmount;
    int plan, payments, entryMode, currency, batchNumber, traceNumber;
    String retCode, autNumber, voucher, originalVoucher, rrn,
           autMessage, merchant, ci;
    Set dirtySet;

    public static final int VERSION       = 0;

    public static final int MANUAL        = 1;
    public static final int SWIPED        = 2;

    // Transaction class
    public static final int AUTHORIZATION       = 0x00000001;
    public static final int FINANCIAL           = 0x00000002;

    // Transaction subclass
    public static final int PURCHASE            = 0x00000004;
    public static final int VOID                = 0x00000008;
    public static final int REFUND              = 0x00000010;
    public static final int DEPOSIT             = 0x00000020;
    public static final int ADJUST              = 0x00000040;

    // Transaction attributes
    public static final int ADVICE              = 0x00000080;
    public static final int REFERENCED          = 0x00000100;

    public static final int CLOSE_REPORTED      = 0x00000200;

    // post-transaction facts
    public static final int VOIDED              = 0x00000400;
    public static final int ADJUSTED            = 0x00000800;
    public static final int REVERSED            = 0x00001000;
    public static final int UNPRINTED           = 0x00002000;
    public static final int CLOSED              = 0x00004000;

    // Host notification
    public static final int HOST_NOTIFIED       = 0x00008000;

    // New post-transaction fact ...
    public static final int REFUNDED            = 0x00010000;

    // Runtime flags
```

```
    public static final int NEED_REVERSE        = 0x00020000;
    public static final int AUTHORITATIVE       = 0x00040000;
    public static final int INVALID             = 0x00080000;

    public CaptureEntry () {
        super();
        sync();
        setAmount (new BigDecimal (0));
        setTransactionDate (new Date());
        setAdditionalAmount (new BigDecimal (0));
    }
    public CaptureEntry (int flags) {
        this();
        setFlags (flags);
    }
    public void sync() {
        dirtySet = new HashSet();
    }
    public void setDirty (String key) {
        dirtySet.add (key);
    }
    public boolean isDirty (String key) {
        return dirtySet.contains (key) || dirtySet.contains ("*");
    }
    public boolean hasDirtyFields () {
        return !dirtySet.isEmpty();
    }
    public void setOID (long oid) {
        this.oid = oid;
    }
    public long getOID () {
        return oid;
    }
    public void setParentOID (long parentOID) {
        this.parentOID = parentOID;
        setDirty ("parentoid");
    }
    public long getParentOID () {
        return parentOID;
    }
    public void setFlags (int flags) {
        this.flags = flags;
        setDirty ("flags");
    }
    public int getFlags() {
        return flags;
    }
    public CaptureEntry or (int mask) {
        flags |= mask;
        setDirty ("flags");
        return this;
    }
    public CaptureEntry and (int mask) {
        flags &= mask;
        setDirty ("flags");
        return this;
    }
    public void setAgentID (int agentid) {
        this.agentid = agentid;
        setDirty ("agentid");
    }
    public int getAgentID() {
        return agentid;
    }
    public void setCardID (int cardid) {
        this.cardid = cardid;
        setDirty ("cardid");
    }
    public int getCardID() {
        return cardid;
    }
```

```
    public void setLocalTerminal (String localTerminal) {
        this.localTerminal = localTerminal;
        setDirty ("localterm");
    }
    public String getLocalTerminal () {
        return localTerminal;
    }
    public void setTransactionDate (Date d) {
        transactionDate = d;
        setDirty ("transdate");
    }
    public void setOriginalDate (Date d) {
        originalDate = d;
        setDirty ("origdate");
    }
    public void setTransmissionDate (Date d) {
        txDate = d;
        setDirty ("txdate");
    }
    public void setClosedDate (Date d) {
        closedDate = d;
        setDirty ("closed");
    }
    public Date getTransactionDate () {
        return transactionDate;
    }
    public Date getTransmissionDate () {
        return txDate;
    }
    public Date getClosedDate() {
        return closedDate;
    }
    public Date getOriginalDate() {
        return originalDate;
    }
    public void setCardHolder (CardHolder cardHolder) {
        this.cardHolder = cardHolder;
        setDirty ("pan");
        setDirty ("exp");
        setDirty ("trailler");
        setDirty ("sec");
    }
    public CardHolder getCardHolder () {
        return cardHolder;
    }
    public BigDecimal getAmount () {
        return amount;
    }
    public void setAmount (BigDecimal amount) {
        this.amount = amount.setScale (2, BigDecimal.ROUND_HALF_UP);
        setDirty ("amount");
    }
    public BigDecimal getAdditionalAmount () {
        return additionalAmount;
    }
    public void setAdditionalAmount (BigDecimal amount) {
        this.additionalAmount = amount.setScale (2, BigDecimal.ROUND_HALF_UP);
        setDirty ("additional");
    }
    public void setPurchasePlan (int plan) {
        this.plan = plan;
        setDirty ("plan");
    }
    public int getPurchasePlan () {
        return plan;
    }
    public int getTraceNumber() {
        return traceNumber;
    }
    public void setTraceNumber (int traceNumber) {
        this.traceNumber = traceNumber;
```

```
        setDirty ("traceno");
    }
    public String getRRN() {
        return rrn;
    }
    public void setRRN (String rrn) {
        this.rrn = rrn;
        setDirty ("rrn");
    }
    public void setNumberOfPayments (int payments) {
        this.payments = payments;
        setDirty ("cuotas");
    }
    public int getNumberOfPayments() {
        return payments;
    }
    public int getEntryMode() {
        return entryMode;
    }
    public void setEntryMode (int entryMode) {
        this.entryMode = entryMode;
        setDirty ("entrymode");
    }
    public void setRetCode (String retCode) {
        this.retCode = retCode;
        setDirty ("retcode");
    }
    public void setRetCode (String retCode, String autMessage) {
        this.retCode = retCode;
        this.autMessage = autMessage;
        setDirty ("retcode");
        setDirty ("autmessage");
    }
    public String getRetCode () {
        return retCode;
    }
    public void setAutNumber (String autNumber) {
        this.autNumber = autNumber;
        setDirty ("autnumber");
    }
    public String getAutNumber() {
        return autNumber;
    }
    public void setAutMessage (String autMessage) {
        this.autMessage = autMessage;
        setDirty ("autmessage");
    }
    public String getAutMessage() {
        return autMessage;
    }
    public String getVoucher() {
        return voucher;
    }
    /**
     * @param maxlen maximun number of characters
     * @return at most maxlen chars
     */
    public String getVoucher(int maxlen) {
        return voucher.length() > maxlen ?
            voucher.substring (voucher.length() - maxlen) :
            voucher;
    }
    public void setVoucher (String voucher) {
        this.voucher = voucher;
        setDirty ("voucher");
    }
    public String getOriginalVoucher() {
        return originalVoucher;
    }
    /**
     * @param maxlen maximun number of characters
```

```
     * @return at most maxlen chars
     */
    public String getOriginalVoucher(int maxlen) {
        return originalVoucher.length() > maxlen ?
            originalVoucher.substring (originalVoucher.length() - maxlen) :
            originalVoucher;
    }
    public void setOriginalVoucher (String voucher) {
        this.originalVoucher = voucher;
        setDirty ("origvoucher");
    }
    public void setRemoteTerminal (String remoteTerminal) {
        this.remoteTerminal = remoteTerminal;;
        setDirty ("remoteterm");
    }
    public String getRemoteTerminal() {
        return remoteTerminal;
    }
    public void setMerchant (String merchant) {
        this.merchant = merchant;
        setDirty ("merchant");
    }
    public String getMerchant () {
        return merchant;
    }
    public void setCI (String ci) {
        this.ci = ci;
        setDirty ("ci");
    }
    public String getCI () {
        return ci;
    }
    public void setCurrency (int currency) {
        this.currency = currency;
        setDirty ("currency");
    }
    public int getCurrency () {
        return currency;
    }
    public void setBatchNumber (int batchNumber) {
        this.batchNumber = batchNumber;
        setDirty ("batchno");
    }
    public int getBatchNumber() {
        return batchNumber;
    }
    public void dump (PrintStream p, String indent) {
        String inner = indent + "  ";
        p.println (indent + "<capture-entry>");
        p.println (inner  + "<oid>" + oid + "</oid>");
        p.println (inner  + "<parent-oid>" + parentOID + "</parent-oid>");
        p.println (inner  + "<flags>0x"+Integer.toString(flags,16)+"</flags>");
        if (isAuthoritative())
            p.println (inner  + "<authoritative/>");

        cardHolder.dump (p, inner);
        p.println (inner  + "<ci>" + ci + "</ci>");
        p.println (inner  + "<local-terminal>" + localTerminal
                          + "</local-terminal>");
        p.println (inner  + "<remote-terminal>" + remoteTerminal
                          + "</remote-terminal>");
        p.println (inner  + "<merchant>" + merchant + "</merchant>");
        p.println (inner  + "<amount>" + amount + "</amount>");
        p.println (inner  + "<amount-additional>" + additionalAmount
                          +"</amount-additional>");

        p.println (inner  + "<plan>" + plan + "</plan>");
        p.println (inner  + "<payments>" + payments + "</payments>");
        p.println (inner  + "<entry-mode>" + entryMode+ "</entry-mode>");
        p.println (inner  + "<currency>" + currency + "</currency>");
        p.println (inner  + "<batch-number>" + batchNumber+ "</batch-number>");
```

```
        p.println (inner  + "<trace-number>" + traceNumber+ "</trace-number>");

        p.println (inner  + "<rrn>" + rrn + "</rrn>");
        p.println (inner  + "<ret-code>" + retCode+ "</ret-code>");
        p.println (inner  + "<aut-number>" + autNumber+ "</aut-number>");
        p.println (inner  + "<aut-message>" + autMessage + "</aut-message>");
        p.println (inner  + "<voucher>" + voucher + "</voucher>");
        if (originalVoucher != null)
            p.println (inner  +
                "<original-voucher>" + originalVoucher + "</original-voucher>");
        if (originalDate != null)
            p.println (inner  +
                "<original-date>" + originalDate + "</original-date>");
        p.println (indent + "</capture-entry>");
    }

    public boolean isOnline() {
        return (flags & HOST_NOTIFIED ) != 0;
    }
    public boolean isAuthoritative() {
        return (flags & AUTHORITATIVE) != 0;
    }
    public boolean isFinancial() {
        return (flags & FINANCIAL ) != 0;
    }
    public boolean isValid() {
        return (flags & INVALID) == 0;
    }
    public boolean isClosed() {
        return (flags & CLOSED) == 0;
    }
    public boolean needReverse() {
        return (flags & NEED_REVERSE) != 0;
    }
    public boolean isCancelable(CaptureEntry e) {
        boolean rc = false;
        String s;
        if ((s=getRetCode()) != null && s.equals ("00")) {
            rc = (checkAnyOn (PURCHASE | REFUND) &&
                checkOff (VOIDED | ADJUSTED | REVERSED | NEED_REVERSE |
                          REFUNDED | INVALID | CLOSED) &&
                getCardHolder().equals(e.getCardHolder()) &&
                getAmount().equals (e.getAmount()) &&
                getPurchasePlan()     == e.getPurchasePlan() &&
                getNumberOfPayments() == e.getNumberOfPayments() &&
                getCurrency() == e.getCurrency());
        }
        return rc;
    }
    public boolean isRefundable(CaptureEntry e) {
        boolean rc = false;
        String s;
        if ((s=getRetCode()) != null && s.equals ("00")) {
            rc = (checkOn (PURCHASE) &&
                checkOff (VOIDED | ADJUSTED | REVERSED | NEED_REVERSE |
                          REFUNDED | INVALID) &&
                getCardHolder().equals(e.getCardHolder()) &&
                e.getAmount().compareTo(getAmount()) <= 0 &&
                getPurchasePlan()     == e.getPurchasePlan() &&
                getNumberOfPayments() == e.getNumberOfPayments() &&
                getCurrency() == e.getCurrency());
        }
        return rc;
    }
    public boolean checkOn (int on) {
        return (flags & on) == on;
    }
    public boolean checkAnyOn (int on) {
        return (flags & on) != 0;
    }
    public boolean checkOff (int off) {
```

```
        return (flags & off) == 0;
    }
    public String getDescription () {
        StringBuffer sb = new StringBuffer();
        if ((flags & VOID) != 0)
            sb.append ("AN ");
        if ((flags & PURCHASE) != 0)
            sb.append ("CO");
        else if ((flags & REFUND) != 0)
            sb.append ("DE");
        if ((flags & ADVICE) != 0)
            sb.append (" OFF");
        if ((flags & (VOIDED | REVERSED | CLOSED | REFUNDED)) != 0) {
            sb.append (" (");
            if ((flags & CLOSED) != 0)
                sb.append ('C');
            if ((flags & VOIDED) != 0)
                sb.append ('A');
            if ((flags & REVERSED) != 0)
                sb.append ('R');
            if ((flags & REFUNDED) != 0)
                sb.append ('D');
            sb.append (')');
        }
        return sb.toString();
    }
}
```

Then we have CaptureEntryPeer, which takes care of CaptureEntries lifecycle:

```
public class CaptureEntryPeer implements PersistentPeer {
    PersistentEngine engine;
    public CaptureEntryPeer () {
        super();
    }
    public CaptureEntryPeer (PersistentEngine engine) {
        super();
        setPersistentEngine (engine);
    }
    public void setPersistentEngine (PersistentEngine engine) {
        this.engine = engine;
    }
    public void create (Object obj) throws SQLException {
        CaptureEntry entry = (CaptureEntry) obj;
        Connection conn = engine.getConnection();
        try {
            engine.executeUpdate (getCreateSql (entry), conn);
            entry.setOID (engine.getOID (conn));
            entry.sync();
        } finally {
            engine.releaseConnection (conn);
        }
    }
    public void load (Object obj) throws SQLException, NotFoundException {
        ResultSet rs = null;
        Connection conn = engine.getConnection();
        try {
            rs = engine.executeQuery (getSelectSql ((CaptureEntry) obj), conn);
            if (!rs.next())
                throw new NotFoundException();
            load (obj, rs);
        } finally {
            if (rs != null)
                rs.close();
```

```
            engine.releaseConnection (conn);
        }
    }
    public void update (Object obj) throws SQLException {
        CaptureEntry entry = (CaptureEntry) obj;
        engine.executeUpdate (getUpdateSql (entry));
        entry.sync();
    }
    public void remove (Object obj) throws SQLException {
        CaptureEntry entry = (CaptureEntry) obj;
        engine.executeUpdate (getRemoveSql (entry.getOID()));
    }
    private void load (Object obj, ResultSet rs) throws SQLException
    {
        CaptureEntry entry = (CaptureEntry) obj;

        entry.setOID              (rs.getInt ("oid"));
        entry.setFlags            (rs.getInt ("flags"));
        entry.setParentOID        (rs.getInt ("parentoid"));
        entry.setAgentID          (rs.getInt ("agentid"));
        entry.setCardID           (rs.getInt ("cardid"));
        entry.setLocalTerminal    (rs.getString ("localterm"));
        entry.setTransactionDate  (rs.getTimestamp ("transdate"));
        entry.setTransmissionDate (rs.getTimestamp ("txdate"));
        entry.setClosedDate       (rs.getTimestamp ("closed"));
        entry.setAmount           (new BigDecimal(rs.getDouble ("amount")));
        entry.setAdditionalAmount (new BigDecimal(rs.getDouble ("additional")));
        entry.setPurchasePlan     (rs.getInt ("plan"));
        entry.setNumberOfPayments (rs.getInt ("cuotas"));
        entry.setEntryMode        (rs.getInt ("entrymode"));
        entry.setRetCode          (rs.getString ("retcode"));
        entry.setAutNumber        (rs.getString ("autnumber"));
        entry.setAutMessage       (rs.getString ("autmessage"));
        entry.setVoucher          (rs.getString ("voucher"));
        entry.setRemoteTerminal   (rs.getString ("remoteterm"));
        entry.setMerchant         (rs.getString ("merchant"));
        entry.setCI               (rs.getString ("ci"));
        entry.setCurrency         (rs.getInt ("currency"));
        entry.setRRN              (rs.getString ("rrn"));
        entry.setTraceNumber      (rs.getInt ("traceno"));
        entry.setBatchNumber      (rs.getInt ("batchno"));
        entry.setOriginalVoucher  (rs.getString ("origvoucher"));
        entry.setOriginalDate     (rs.getDate ("origdate"));

        try {
            CardHolder ch = new CardHolder(
                ISOUtil.trim (rs.getString ("pan")), rs.getString ("exp"));
            ch.setTrailler (ISOUtil.trim (rs.getString ("trailler")));
            ch.setSecurityCode (ISOUtil.trim (rs.getString ("sec")));
            entry.setCardHolder (ch);
        } catch (InvalidCardException e) {
            throw new SQLException (e.toString());
        }
        entry.sync();
    }

    // ============================================================
    // Finders
    // ============================================================

    // ============================================================
    // private helper methods
    // ============================================================
    private void set (CaptureEntry o, RowMap map, String key, String value) {
        if (o.isDirty (key))
            map.set (key, value);
    }
    private void set (CaptureEntry o, RowMap map, String key, BigDecimal value)
    {
        if (o.isDirty (key))
            map.set (key, value);
```

```
    }
    private void set (CaptureEntry o, RowMap map, String key, Date value)
    {
        if (o.isDirty (key))
            map.set (key, value);
    }
    private void set (CaptureEntry o, RowMap map, String key, long value) {
        if (o.isDirty (key))
            map.set (key, value);
    }
    private RowMap createMap (CaptureEntry o) {
        RowMap map = new RowMap();
        map.set ("flags", o.getFlags());  // always set flags
        set (o, map, "parentoid",  o.getParentOID());
        set (o, map, "agentid",    o.getAgentID());
        set (o, map, "cardid",     o.getCardID());
        set (o, map, "localterm",  o.getLocalTerminal());
        set (o, map, "transdate",  o.getTransactionDate());
        set (o, map, "txdate",     o.getTransmissionDate());
        set (o, map, "closed",     o.getClosedDate());
        set (o, map, "amount",     o.getAmount());
        set (o, map, "additional", o.getAdditionalAmount());
        set (o, map, "plan",       o.getPurchasePlan());
        set (o, map, "cuotas",     o.getNumberOfPayments());
        set (o, map, "entrymode",  o.getEntryMode());
        set (o, map, "retcode",    o.getRetCode());
        set (o, map, "autnumber",  o.getAutNumber());
        set (o, map, "autmessage", o.getAutMessage());
        set (o, map, "voucher",    o.getVoucher());
        set (o, map, "remoteterm", o.getRemoteTerminal());
        set (o, map, "merchant",   o.getMerchant());
        set (o, map, "ci",         o.getCI());
        set (o, map, "currency",   o.getCurrency());
        set (o, map, "rrn",        o.getRRN());
        set (o, map, "traceno",    o.getTraceNumber());
        set (o, map, "batchno",    o.getBatchNumber());
        set (o, map, "origvoucher",o.getOriginalVoucher());
        set (o, map, "origdate",   o.getOriginalDate());
        CardHolder ch = o.getCardHolder();
        if (ch == null) {
            map.set ("pan",      (String) null);
            map.set ("exp",      (String) null);
            map.set ("trailler", (String) null);
            map.set ("sec",      (String) null);
        } else {
            set (o, map, "pan",      ch.getPAN());
            set (o, map, "exp",      ch.getEXP());
            set (o, map, "trailler", ch.getTrailler());
            set (o, map, "sec",      ch.getSecurityCode());
        }
        return map;
    }
    private String getCreateSql (CaptureEntry o) {
        RowMap map = createMap (o);
        return map.getInsertSql("capture");
    }
    private String getUpdateSql (CaptureEntry o) {
        RowMap map = createMap (o);
        return map.getUpdateSql("capture", "oid = "+Long.toString (o.getOID()));
    }
    private String getSelectSql (CaptureEntry o) {
        return "SELECT * from capture WHERE oid = "
            +Long.toString (o.getOID());
    }
    private String getSelectByParentOIDSql (long oid) {
        return "SELECT * from capture WHERE parentoid = "
            +Long.toString (oid);
    }
    private String getSelectByFlag
        (int agentid, String remoteTerm, int maskOn, int maskOff)
    {
```


The book exclusively prepared for Courtesy Edgardo Regodon Jr <eregodon@aeoncredit.com.ph> - 07/2009 (r2966)

```
        return "SELECT * from capture WHERE agentid = "
            + Integer.toString (agentid)
            + " AND remoteterm = '"
            + remoteTerm + "' AND (flags & 0x" + Integer.toString (maskOn,16)
            + ") and not (flags & 0x" + Integer.toString (maskOff, 16)
            + ") ORDER BY oid";
    }
    private String getSelectByVoucher (String voucher)
    {
        return "SELECT * from capture WHERE voucher = '" + voucher
            + "' ORDER BY oid";
    }
    private String getSelectByVoucherDesc (String voucher)
    {
        return "SELECT * from capture WHERE voucher = '" + voucher
            + "' ORDER BY oid DESC";
    }
    private String getSelectByOpenBatch
        (String remoteTerm, int agentid, int currency)
    {
        return "SELECT * from capture WHERE remoteterm = '"
            + remoteTerm + "' AND agentid = " + Integer.toString (agentid)
            + " AND NOT (flags & 0x84001) AND retcode = '00'"
            + (currency == 0 ?
                "" : (" AND currency = "+Integer.toString (currency))
            + " ORDER BY oid"
        );
    }
    private String getRemoveSql (long oid) {
        return "DELETE FROM capture WHERE oid = "+Long.toString (oid);
    }

    private Collection createCollection (String query) throws SQLException {
        Collection c = new ArrayList();
        ResultSet rs = null;
        Connection conn = engine.getConnection();
        try {
            rs = engine.executeQuery (query, conn);
            while (rs.next()) {
                CaptureEntry entry = new CaptureEntry();
                load (entry, rs);
                c.add (entry);
            }
        } finally {
            if (rs != null)
                rs.close();
            engine.releaseConnection (conn);
        }
        return c;
    }

    //----------------------------------------------------------------
    // Peer finders
    //----------------------------------------------------------------
    public CaptureEntry findByOID (long oid)
        throws SQLException, NotFoundException
    {
        CaptureEntry entry = new CaptureEntry();
        entry.setOID (oid);
        load (entry);
        return entry;
    }
    public Collection findByParentOID (long oid) throws SQLException {
        return createCollection (getSelectByParentOIDSql (oid));
    }
    public Collection findByFlags
        (int agentid, String remoteTerm, int maskOn, int maskOff)
        throws SQLException
    {
        return createCollection(
            getSelectByFlag (agentid, remoteTerm, maskOn, maskOff)
```

```
        );
    }
    public Collection findByVoucher (String voucher)
        throws SQLException
    {
        return createCollection(getSelectByVoucher (voucher));
    }
    public Collection findByVoucherDesc (String voucher)
        throws SQLException
    {
        return createCollection(getSelectByVoucherDesc (voucher));
    }
    public Collection findOpenBatch
        (String remoteTerm, int agentid, int currency)
        throws SQLException
    {
        return createCollection(
            getSelectByOpenBatch(remoteTerm, agentid, currency)
        );
    }
}
```

13.1.24. PersistentEngine

PersistentEngine lets you "create", "load", "update" and "remove" objects:

```
    ...
    public void create (Object o) throws NoPeerException, SQLException
    public void load   (Object o) throws NoPeerException, SQLException, NotFoundException
    public void update (Object o) throws NoPeerException, SQLException, NotFoundException
    public void remove (Object o) throws NoPeerException, SQLException, NotFoundException
    ...
```

Once you've got a reference to a `PersistentEngine` instance, you can easily create, load, update and remove objects. `PersistentEngine` takes care of instantiating a Peer and delegating calls to interact with an SQL back-end database.

13.1.25. ConnectionPool

ConnectionPool, as its name implies, is a general purpose JDBC connection pool.

Once instantiated, you can call its `Connection getConnection()` and `free (Connection)` methods.

ConnectionPool can be configured using its detailed constructor:

```
    public ConnectionPool(String driver, String url,
                          String username, String password,
                          int initialConnections,
                          int maxConnections,
                          boolean waitIfBusy)
        throws SQLException
```

[See javadocs [http://jpos.org/doc/javadoc/org/jpos/tpl/ConnectionPool.html] for details.]

However, in most cases you may want to use the default no-args constructor and then call its `setConfiguration (Configuration)` method, which accepts the following properties:

- jdbc.driver
- jdbc.url
- jdbc.user
- jdbc.password
- initial-connections
- max-connections
- wait-if-busy

13.1.26. V24

Modem, SimpleDialupModem, LeasedLineModem

jPOS provides a small layer on top of the **JavaCOMM API** (`javax.comm.*`) that can be used to deal with asynchronous communications, modems, and legacy wire protocols such as VISA-1, etc.

The main class is called `org.jpos.util.V24` which adds several handy methods on top of JavaCOMM API's `SerialPort` class.

Example 13.13. Using V24

```
public final String[] resultCodes = {
    "OK\r",
    "CONNECT\r",
    "RING\r",
    "NO CARRIER\r",
    "ERROR\r",
    "CONNECT",
    "NO DIALTONE\r",
    "BUSY\r",
    "NO ANSWER\r",
    "NO DIAL TONE\r",
};

V24 v24 = new V24 ("/dev/ttyS0", myLogger, "v24");
v24.dtr (true);
v24.send ("AT\r");
int rc = v24.waitfor (resultCodes, 10000);
```

> **Note**
> A full description of V24's methods can be found in jPOS's javadocs [http://jpos.org/doc/javadoc/org/jpos/util/V24.html].

In addition to V24, jPOS provides an interface called Modem:

```
public interface Modem {
    public void dial (String number, long aproxTimeout) throws IOException;
    public void hangup () throws IOException;
```

```
        public void answer () throws IOException;
        public boolean isConnected();
    }
```

...as well as a couple of implementations:

- LeasedLineModem

  Can be used to connect a leased line or null [dumb] modem

- SimpleDialupModem

  Implements a Hayes[tm]-compatible command set.

1. VISA1Link

`VISA1Link` can be used to send and receive packets using a VISA-1 protocol (ENQ/ACK/NAK).

In order to integrate VISA-1-based interchanges with ISO-8583-based components, we use a pseudo-packager called `org.jpos.iso.packager.VISA1Packager` which in turn uses an implementation of `VISA1ResponseFilter`:

```
    public interface VISA1ResponseFilter {
        public String guessAutNumber (String response);
    }
```

Example 13.14. Creating a VISA-1 Link

```
    public V24 createV24 ()
        throws  IOException, PortInUseException,
                UnsupportedCommOperationException
    {
        String portName  = cfg.get    ("port");
        int    baud      = cfg.getInt ("baud");
        String sParity   = cfg.get    ("parity");
        int    databits  = cfg.getInt ("databits");

        int parity = SerialPort.PARITY_NONE;
        if (sParity.equalsIgnoreCase ("even"))
            parity = SerialPort.PARITY_EVEN;
        else if (sParity.equalsIgnoreCase ("odd"))
            parity = SerialPort.PARITY_ODD;

        V24 v24 = new V24 (portName, getLogger(), "V24."+portName);
        v24.setSpeed (baud,
                databits == 7 ? SerialPort.DATABITS_7 : SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                parity, 0
        );
        return v24;
    }
    public void createVISA1Link () {
        try {
            timeout = cfg.getInt ("timeout");
            V24 v24 = createV24 ();
            Modem mdm = new LeasedLineModem (v24);
```

```
            VISA1Packager packager = new VISA1Packager
                (getFieldSequence(), 63, "05", cfg.get ("pattern"));
            packager.setLogger (getLogger(), "VISA1Packager");
            link = new VISA1Link (v24, mdm, packager);
            link.setWaitENQ (cfg.getBoolean ("waitenq"));
            link.setTimeout (timeout);
            link.setLogger (getLogger(), "VISA1Link");
            if (((Object)this) instanceof VISA1ResponseFilter) {
                packager.setVISA1ResponseFilter ((VISA1ResponseFilter) this);
            }
            Thread t = new Thread (link);
            t.start();
        } catch (Exception e) {
            Logger.log (new LogEvent (this, prefix, e));
        }
    }
    public int[] getFieldSequence() {
        int[] sequence = { 63, 35, 4, 48, 48, 62 };
        return sequence;
    }
```

Glossary

BSH	BeanShell (JSR-274) script (http://www.beanshell.org, http://jcp.org/en/jsr/detail?id=274).
CLI	Q2's Command Line Interface.
JCP	Java Community Process (http://jcp.org).
MID	Merchant Identifier
MTI	Message Type Indicator
TID	Terminal Identifier

Appendix A. Getting involved

The best way to get involved is to browse the jpos-dev [http://groups.yahoo.com/group/jpos-dev] mailing list archives and then join the list and introduce yourself.

There you'll find over one thousand jPOS users and developers sharing useful information about jPOS and related technology, use cases as well as success stories.

Most of the action in jPOS' development takes place in the CVS HEAD version (available in SourceForge [http://sf.net/projects/jpos]). If you want to get your hands dirty with jPOS internals, that's the right version to get.

Figure A.1. The team

Tip

You can also find us online in `irc.freenode.net`, on the `#jpos` channel.

You may also consider registering yourself in our WhoIsUsing [http://jpos.org/wiki/index.php/WhoIsUsing] page.

Appendix B. Bibliography

Bibliography

[Design Patterns] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Copyright © 1995 Addison-Wesley Longman, Inc. 0-201-63361-2. Addison-Wesley professional computing series. *Design Patterns, elements of Reusable Object-Oriented Software* .

[Applied Cryptography] Bruce Schneier. Copyright © 1993 Wiley. 0-471-59756-2. Wiley. *Applied Cryptography, Protocols, Algorithms, and Source Code in C* .

[Data Communications] Uyless Black. Copyright © 1987 Prentice-Hall. 0-8359-1209-4. Prentice-Hall Internatinal. *Data communications and distributed networks* .

[Mastering RMI] Rickard Oberg. Copyright © 2001 Wiley. 0-471-38940-4. Wiley. *Mastering RMI* .

[The Linda Coordination Language] Nicholas Carriero and David Gelernter. Copyright © 1988 Yale University. Wiley. *The Linda Coordination Language* .

[JavaSpaces Principles, Patterns, and Practice] Eric Freeman, Susanne Hupfer, and Ken Arnold. Copyright © 1999 Addison-Wesley Longman, Inc. 0-201-30955-6. Addison-Wesley professional computing series. *JavaSpaces Principles, Patterns, and Practice* .

[JMX] Juha Lindfors and Marc Fleury. Copyright © 1999-2001 The JBoss Group. 0-672-32288-9. SAMS. *JMX, Managing J2EE with Java® Management Extensions* .

Appendix C. ISOFieldPackagers

Naming convention for ISOFieldPackager is the following:

- IF - all ISOFieldPackagers starts with the prefix IF (ISO Field)
- A for ASCII representation
- B for BCD representation
- E for EBCDIC
- LL for variable length <100
- LLL for variable length <1000
- LLLL for variable length <10000
- LLLLL for variable length <100000
- LLH for variable length <100 (length is hex value)
- LLLH for variable length <1000 (length is hex value)
- F for fixed length

Table C.1. ISOFieldPackagers

Name	Purpose
IFA_AMOUNT	Fixed length amount field (ASCII)
IFA_BINARY	Fixed length binary field stored as an hex string (requires double length)
IFA_BITMAP	Bitmap stored as an hex string (requires double length)
IFA_FLLCHAR	ASCII alphanumeric field, variable length (<100) with trailling filler
IFA_FLLNUM	Variable length numeric field with trailling filler
IFA_LLBINARY	Variable length (<100) binary field stored as an hex-String (requires double length)
IFA_LLCHAR	Variable length (<100) alphanumeric field (ASCII)
IFA_LLLBINARY	Variable length (<1000) binary field stored as an hex-String (requires double length)
IFA_LLLCHAR	Variable length (<1000) alphanumeric field (ASCII)
IFA_LLLLCHAR	Variable length (<10000) alphanumeric field (ASCII)
IFA_LLLLLCHAR	Variable length (<100000) alphanumeric field (ASCII)

Name	Purpose
IFA_LLLNUM	Variable length (<1000) numeric field (ASCII)
IFA_LLNUM	Variable length (<100) numeric field (ASCII)
IFA_LLBNUM	Variable length (<100) numeric field (BCD)
IFA_NUMERIC	Fixed length numeric field (ASCII)
IFB_AMOUNT	Fixed length amount (BCD)
IFB_BINARY	Fixed length, raw bytes
IFB_BITMAP	raw bitmap
IF_CHAR	Fixed length ASCII
IFB_LLBINARY	Variable length (<100) raw bytes
IFB_LLCHAR	Variable length (<100) ASCII
IFB_LLHBINARY	Variable length (<100) raw bytes, length is raw too
IFB_LLHCHAR	Variable length (<100) alphanumeric (ASCII), length is raw value
IFB_LLHECHAR	Variable length (<100) alphanumeric (EBCDIC), length is raw value
IFB_LLHFBINARY	Variable length (<100) binary padded to fixed length. length is raw value
IFB_LLHNUM	Variable length (<100) numeric (BCD)
IFB_LLLBINARY	Variable length (<1000) binary
IFB_LLLCHAR	Variable length (<1000) alphanumeric
IFB_LLLNUM	Variable length (<1000) numeric (BCD). Length is represented as a BCD.
IFB_LLNUM	Variable length (<100) numeric (BCD). Length is represented as a BCD.
IFB_NUMERIC	Fixed length numeric (BCD)
IFEP_LLCHAR	Variable length (<100) alphanumeric (as used by EuroPay).
IFE_CHAR	Fixed length, alphanumeric (EBCDIC)
IFE_LLCHAR	Variable length (<100) alphanumeric (EBCDIC).
IFE_LLLCHAR	Variable length (<1000) alphanumeric (EBCDIC).
IFE_LLNUM	Variable length (<100) numeric (EBCDIC).
IFEB_LLNUM	Variable length (<100) numeric (length in EBCDIC) with special padding used by GICC packager
IFEB_LLLNUM	Variable length (<1000) numeric (length in EBCDIC) with special padding used by GICC packager
IFE_NUMERIC	Fixed length numeric (EBCDIC).

Name	Purpose
IF_CHAR	fixed length, alphanumeric (ASCII)
IF_ECHAR	fixed length, alphanumeric (EBCDIC)
IF_NOP	No operation
IF_CHAR	Fixed length alphanumeric (ASCII)
IFE_CHAR	Fixed length alphanumeric (EBCDIC)
IFA_NUMERIC	Fixed length numeric (ASCII)
IFE_NUMERIC	Fixed length numeric (EBCDIC)
IFB_NUMERIC	Fixed length numeric (BCD)
IFB_LLNUM	Variable length numeric (BCD, maxlength=99)
IFB_LLLNUM	Variable length numeric (BCD, maxlength=999)

Note

From time to time we find some new esoteric way of representing an ISO-8583 field. If you feel you need a new field definition, please check our ChangeLog [http://jpos.org/wiki/ChangeLog] first, you may be lucky and your field packager may be already there.

Appendix D. qsp-config.dtd

QSP's configuration file sits in `src/config/qsp/qsp-config.dtd`. As of this writing, its content looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT qsp-config (logger*,mux*,object*,task*,daily-task*,server*,
                      control-panel*,sequencer*,card-agent*,channel*,
                      persistent-engine*, connection-pool*, dir-poll*,
                      s-m-adapter*, secure-key-store*,
                      property*)*>
<!ATTLIST qsp-config logger IDREF #IMPLIED>
<!ATTLIST qsp-config realm  CDATA #IMPLIED>
<!ATTLIST qsp-config reload CDATA #IMPLIED>
<!ATTLIST qsp-config name   CDATA #IMPLIED>

<!-- logger -->
<!ELEMENT logger (log-listener*)>
<!ATTLIST logger name ID #REQUIRED>

<!-- log-listener -->
<!ELEMENT log-listener (property*)>
<!ATTLIST log-listener class NMTOKEN #REQUIRED>
<!ATTLIST log-listener name  CDATA #IMPLIED>

<!-- channel -->
<!ELEMENT channel (property*, filter*)*>
<!ATTLIST channel name ID #REQUIRED>
<!ATTLIST channel class NMTOKEN #REQUIRED>
<!ATTLIST channel connect (yes|no) 'no'>
<!ATTLIST channel logger IDREF #IMPLIED>
<!ATTLIST channel realm CDATA #IMPLIED>
<!ATTLIST channel header CDATA #IMPLIED>
<!ATTLIST channel packager CDATA #IMPLIED>
<!ATTLIST channel panel IDREF #IMPLIED>
<!ATTLIST channel type (client|server) #IMPLIED>
<!ATTLIST channel timeout CDATA #IMPLIED>
<!ATTLIST channel packager-config CDATA #IMPLIED>
<!ATTLIST channel packager-logger CDATA #IMPLIED>
<!ATTLIST channel packager-realm  CDATA #IMPLIED>
<!ATTLIST channel socket-factory  CDATA #IMPLIED>

<!-- filter -->
<!ELEMENT filter (property*,calculate*,SQL*)*>
<!ATTLIST filter name ID #IMPLIED>
<!ATTLIST filter class NMTOKEN #REQUIRED>
<!ATTLIST filter direction (incoming|outgoing|both) 'both'>

<!-- SQL -->
<!ELEMENT SQL EMPTY>
<!ATTLIST SQL switch CDATA #REQUIRED>
<!ATTLIST SQL statement CDATA #REQUIRED>

<!-- calculate -->
<!ELEMENT calculate EMPTY>
<!ATTLIST calculate switch  CDATA #REQUIRED>
<!ATTLIST calculate field CDATA #REQUIRED>
<!ATTLIST calculate statement CDATA #REQUIRED>

<!-- router -->
<!ELEMENT router EMPTY>
<!ATTLIST router switch CDATA #REQUIRED>
<!ATTLIST router type CDATA #REQUIRED>
<!ATTLIST router destination IDREF #REQUIRED>
<!ATTLIST router timeout CDATA #IMPLIED>
<!ATTLIST router bounce CDATA #IMPLIED>
```

```
<!-- mux -->
<!ELEMENT mux (property*,channel,request-listener*)*>
<!ATTLIST mux name ID #REQUIRED>
<!ATTLIST mux logger IDREF #IMPLIED>
<!ATTLIST mux realm CDATA #IMPLIED>
<!ATTLIST mux connect (yes|no) 'yes'>
<!ATTLIST mux class NMTOKEN #IMPLIED>

<!-- object -->
<!ELEMENT object (property)*>
<!ATTLIST object class NMTOKEN #REQUIRED>
<!ATTLIST object logger IDREF #IMPLIED>
<!ATTLIST object realm CDATA #IMPLIED>
<!ATTLIST object name ID #IMPLIED>
<!ATTLIST object connection-pool IDREF #IMPLIED>

<!-- task -->
<!ELEMENT task (property)*>
<!ATTLIST task class NMTOKEN #REQUIRED>
<!ATTLIST task logger IDREF #IMPLIED>
<!ATTLIST task realm CDATA #IMPLIED>
<!ATTLIST task name ID #IMPLIED>
<!ATTLIST task connection-pool IDREF #IMPLIED>

<!-- property -->
<!ELEMENT property EMPTY>
<!ATTLIST property name  CDATA #REQUIRED>
<!ATTLIST property value CDATA #IMPLIED>
<!ATTLIST property file  CDATA #IMPLIED>

<!-- request-listener -->
<!ELEMENT request-listener (property*,router*)*>
<!ATTLIST request-listener class NMTOKEN #REQUIRED>
<!ATTLIST request-listener logger IDREF #IMPLIED>
<!ATTLIST request-listener realm CDATA #IMPLIED>

<!-- server  -->
<!ELEMENT server (channel,request-listener*)*>
<!ATTLIST server name ID #REQUIRED>
<!ATTLIST server port CDATA #REQUIRED>
<!ATTLIST server logger IDREF #IMPLIED>
<!ATTLIST server realm CDATA #IMPLIED>
<!ATTLIST server maxSessions CDATA #IMPLIED>
<!ATTLIST server panel IDREF #IMPLIED>

<!-- control-panel -->
<!ELEMENT control-panel (panel)*>
<!ATTLIST control-panel rows CDATA #REQUIRED>
<!ATTLIST control-panel cols CDATA #REQUIRED>

<!-- panel -->
<!ELEMENT panel EMPTY>
<!ATTLIST panel name ID #REQUIRED>

<!-- sequencer -->
<!ELEMENT sequencer (property)*>
<!ATTLIST sequencer name ID #REQUIRED>
<!ATTLIST sequencer class NMTOKEN #REQUIRED>
<!ATTLIST sequencer logger IDREF #IMPLIED>
<!ATTLIST sequencer realm CDATA #IMPLIED>

<!-- card-agent -->
<!ELEMENT card-agent (property)*>
<!ATTLIST card-agent class NMTOKEN #REQUIRED>
<!ATTLIST card-agent logger IDREF #IMPLIED>
<!ATTLIST card-agent realm CDATA #IMPLIED>
<!ATTLIST card-agent persistent-engine CDATA #IMPLIED>

<!-- persistent-engine -->
<!ELEMENT persistent-engine (property)*>
```

```
<!ATTLIST persistent-engine name ID #REQUIRED>
<!ATTLIST persistent-engine logger IDREF #IMPLIED>
<!ATTLIST persistent-engine realm CDATA #IMPLIED>

<!-- connection-pool -->
<!ELEMENT connection-pool (property)*>
<!ATTLIST connection-pool name ID #REQUIRED>
<!ATTLIST connection-pool logger IDREF #IMPLIED>
<!ATTLIST connection-pool realm CDATA #IMPLIED>

<!-- directory-poll -->
<!ELEMENT dir-poll (property)*>
<!ATTLIST dir-poll name ID #IMPLIED>
<!ATTLIST dir-poll path CDATA #REQUIRED>
<!ATTLIST dir-poll create (yes|no) 'no'>
<!ATTLIST dir-poll poolsize CDATA #IMPLIED>
<!ATTLIST dir-poll interval CDATA #IMPLIED>
<!ATTLIST dir-poll priorities CDATA #IMPLIED>
<!ATTLIST dir-poll processor CDATA #REQUIRED>
<!ATTLIST dir-poll logger IDREF #IMPLIED>
<!ATTLIST dir-poll realm  CDATA #IMPLIED>

<!-- daily-task -->
<!ELEMENT daily-task (property)*>
<!ATTLIST daily-task class NMTOKEN #REQUIRED>
<!ATTLIST daily-task logger IDREF #IMPLIED>
<!ATTLIST daily-task realm CDATA #IMPLIED>
<!ATTLIST daily-task name ID #IMPLIED>
<!ATTLIST daily-task start CDATA #REQUIRED>
<!ATTLIST daily-task poolsize CDATA #IMPLIED>

<!-- s-m-adapter -->
<!ELEMENT s-m-adapter (property)*>
<!ATTLIST s-m-adapter class NMTOKEN #REQUIRED>
<!ATTLIST s-m-adapter logger IDREF #IMPLIED>
<!ATTLIST s-m-adapter realm CDATA #IMPLIED>
<!ATTLIST s-m-adapter name ID #IMPLIED>

<!-- secure-key-store -->
<!ELEMENT secure-key-store (property)*>
<!ATTLIST secure-key-store class NMTOKEN #REQUIRED>
<!ATTLIST secure-key-store logger IDREF #IMPLIED>
<!ATTLIST secure-key-store realm CDATA #IMPLIED>
<!ATTLIST secure-key-store name ID #IMPLIED>
```

Tip

Get latest version here [http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/jpos/jpos/src/config/qsp/]

Appendix E. QTest and QTestMBean source code

Example E.1. QTest

```
package org.jpos.test;

import org.jdom.Element;
import org.jdom.Comment;
import org.jpos.iso.ISOUtil;
import org.jpos.q2.Q2;
import org.jpos.q2.QBean;
import org.jpos.q2.QBeanSupport;

import org.jpos.util.Log;
import org.jpos.util.Logger;

public class QTest extends QBeanSupport implements Runnable, QTestMBean {
    long tickInterval = 1000;

    public QTest () {
        super();
        log.info ("constructor");
    }
    public void init () {
        log.info ("init");
        super.init ();
    }
    public void start() {
        log.info ("start");
        super.start ();
    }
    public void stop () {
        log.info ("stop");
        super.stop ();
    }
    public void destroy () {
        log.info ("destroy");
        log = null;
    }
    public void setPersist (Element e) {
        log.info ("setPersist");
        super.setPersist (e);
    }
    public Element getPersist () {
        setModified (false);
        log.info ("getPersist");
        return createElement ("qtest", QTestMBean.class);
    }
    public void setName (String name) {
        log.info ("setName " + name);
        super.setName (name);
    }
    public void setTickInterval (long tickInterval) {
        this.tickInterval = tickInterval;
        setModified (true);
    }
    public long getTickInterval () {
        return tickInterval;
    }
    public void run () {
        for (int tickCount=0; running (); tickCount++) {
            log.info ("tick " + tickCount);
            ISOUtil.sleep (tickInterval);
```

```
        }
    }
    public void startService() {
        new Thread(this).start();
    }
}
```

Example E.2. QBeanMBean

```
package org.jpos.test;

import org.jdom.Element;

public interface QTestMBean extends org.jpos.q2.QBeanSupportMBean {
    public Element getPersist ();
    public void setTickInterval(long tickInterval) ;
    public long getTickInterval() ;
}
```

Example E.3. 10_qtest.xml

```
<qtest class="org.jpos.test.QTest">
 <attr name="tickInterval" type="java.lang.Long">5000</attr>
</qtest>
```

Appendix F. ChangeLog

The list of changes is oficially maintained in the wiki page ChangeLog [http://jpos.org/wiki/ChangeLog]. You are encouraged to subscribe to it using its RSS feed or at least regularly check the wiki's Recent Changes Page [http://jpos.org/w/index.php/Special:Recentchanges].

F.1. jPOS 1.4.9

- new package org.jpos.tlv (TLVMsg, TLVList)
- Added HEXChannel
- LogEvent new constructors (no need for LogSource), added toString()
- FSDMsg supports 'B' type (Binary data)
- DirPoll - ability to archive processed requests
- BaseChannel.disconnect - set SO_LINGER to zero
- #423 - ISODate.parseISODate - force milliseconds down to zero.
- Changed sendMessageTrailler signature in order to make packed image available.
- Added interface ReadableConfiguration
- Added Configuration.put
- JDBMSpace and TSpace performs automatic gc by means of a Timer task
- Added LocalSpace.addListener(key,listener,timeout)
- New TSpace implementation
- Added ObjectTemplate
- bugfix in SpaceProxy jndi binding
- BSHRequestListener whitelist support
- Added BSHTransactionParticipant and BSHGroupSelector
- Added Xm* ckage org.jpos.tlv (TLVMsg, TLVList)
- Added HEXChannel
- LogEvent new constructors (no need for LogSource), added toString()
- FSDMsg supports 'B' type (Binary data)
- DirPoll - ability to archive processed requests
- BaseChannel.disconnect - set SO_LINGER to zero

- #423 - ISODate.parseISODate - force milliseconds down to zero.
- Changed sendMessageTrailler signature in order to make packed image available.
- Added interface ReadableConfiguration
- Added Configuration.put
- JDBMSpace and TSpace performs automatic gc by means of a Timer task
- Added LocalSpace.addListener(key,listener,timeout)
- New TSpace implementation
- Added ObjectTemplate
- bugfix in SpaceProxy jndi binding
- BSHRequestListener whitelist support
- Added BSHTransactionParticipant and BSHGroupSelector
- Added XmlConfigurable to jpos-util.jar
- Q2Configurable replaced by org.jpos.core.XmlConfigurable
- Q2ConfigurationException replaced by ConfigurationException
- ISOException accepts throwable as nested parameter (instead of Exception)
- Bugfix in ISOUtil ebcdic tables
- Enhanced support for SSL in ChannelAdaptor and QServer
- Added IFB_LLLHCHAR
- Bugfix in RowMap (escape """) lConfigurable to jpos-util.jar
- Q2Configurable replaced by org.jpos.core.XmlConfigurable
- Q2ConfigurationException replaced by ConfigurationException
- ISOException accepts throwable as nested parameter (instead of Exception)
- Bugfix in ISOUtil ebcdic tables
- Enhanced support for SSL in ChannelAdaptor and QServer
- Added IFB_LLLHCHAR
- Bugfix in RowMap (escape """)

F.2. jPOS 1.5.0

jPOS was released in 09/18/2005 and tagged as v1_5_0. This was the last release available as a binary distribution. From there one we have been working directly over the CVS (now Subversion) repository.

F.3. jPOS 1.5.1

- org.jpos.util.DefaultTimer - set daemon flag
- FSDMsg: use TSpace instead of TransientSpace
- BASE1Header.unpack doesn't clone byte array
- ChannelAdaptor now registers itself in the NameRegistrar
- IFB_BITMAP, deal with primary bitmaps where bit 0 turned on doesn't mean we have a secondary bitmap.
- FSDMsg, bugfix: read was breaking on FS, even if fs=false
- ConnectionPool - Altered getConnection to wait if maxConnections is not reached and a new connection is pending even if waitIfBusy is false.
- Added QBean based SystemMonitor (org.jpos.q2.qbean).
- BSHFilter honors returned ISOMsg
- Q2 prevents exiting the JVM when not running in standalone mode (i.e. inside an appserver)
- org.jpos.ui.action.Exit now accepts an optional exit code
- q2mod_jpos added ConnectionPoolAdaptor
- BaseChannel.sendMessage (byte[] b, int offset, int len) was not honoring offset and length - fixed.
- QMUX register itself with the NameRegistrar at init time instead of start
- NPE in ISOUtil.trimf
- IF_NOP.pack now returns a zero-length byte array instead of null
- EuroSubFieldPackager.pack fixed problem with ebcdic subfields
- Added support for postpack F127.26
- Fixed postpack fields 127 and 52
- BSHTransactionParticipant implements AbortPartipant - added optional trace attribute
- ISOBasePackager.unpack raises exception if field packager not configured
- Added SunJSSESocketFactory.getConfiguration()
- BaseChannel now honors local-iface and local-port properties
- Fixed bug in ISODate.parseISODate, added unit tests
- ChannelAdaptor doesn't disconnect on VetoException
- BSHFilter handles VetoException
- TLVList allow starting offset on unpack

- FSDMsg.dump more YAML-ish and less XML-ish
- q2modules - added OneShotChannelAdaptor
- TLVList allow data stream being unpacked to end with padding (0x00 or 0xFF)
- Bug fix in BCDInterpreter (when using RIGHT_PADDED_F)
- Added GZIPChannel
- ISOMsgPanel: Ask the message to be displayed for the highest field, rather than assume 128
- ConnectionPool retries connections on errors (r2260)
- GenericSubFieldPackager skip over null fields, this allows fields to be non-sequential in number. (r2262)
- Q2 - Include thrown Errors in the problem renaming processing (.ENV extension)

F.4. jPOS 1.5.2

Tagged as v1_5_2 - only available via subversion. 1.5.2 is the last version in the 1.5.x series. The next version is 1.6.0 that we call **jPOS 6**

> **Note**
> Please note that jPOS 6 has a new license (see Appendix H, *GNU Affero General Public License version 3* for details).

F.5. jPOS 1.6.0

(aka jPOS 6)

jPOS 6 Is a major reorganization of the jPOS project that has accumulated the following changes since 1.5.2:

- Upgrade to jdom-1.0.jar
- Changed LogEvent to support new jdom version
- Upgraded build system (uses jPOS-EE's approach
- Added DailyLogListener
- Use crimson instead of xerces
- Created compat_1_5_2 module
- Upgraded jdbm to 1.0
- transient:* and persistent:* Spaces 'urns' now maps to TSpace and JDBMSpace
- ISOMsg writeExternal and readExternal now honors packager if available
- ISOMsg.set(int,String) can be used for ISOBinaryFields provided there's a packager associated with the ISOMsg (the string is interpreted as hex)

- Added appropriate thread names
- TransactionManager stuff moved to its own module 'txnmgr'
- BaseChannel.disconnect closes socket before closing streams (better SSL behaveour)
- ISOServer do not drop connection on VetoException
- bugfix in Log4JListener (if watch=0 use default watch to avoid CPU going up to 100%
- TSpace and JDBMSpace throws NPE at out time if either key or value are null
- ISOCurrency's data moved to a ResourceBundle
- Dump hexstring of binary fields in ISOBasePackager.unpack
- Added IFE_AMOUNT
- Added TransactionManager debug property
- TransactionManager creates default persistent-space if one is not specified
- TransactionManager honors participants configured after group selectors
- TransactionManager now support RETRY action at prepare time (r2359)
- QServer.stop - avoid npe on unstarted/misconfigured servers (r2360)
- ISOChannelPanel now supports wipe and protect attributes (r2363)
- Added IFE_SIGNED_NUMERIC and SignedEbcdicNumberInterpreter
- Added sendKeepAlive() method to BaseChannel
- New ´keep-alive´ element in ChannelAdaptor
- FSDMsg trunk fields bigger than length
- Reworked 3rd bitmap support
- QMUX now support multiple ready indicators
- NPE in Q2 startup when deploy directory is not available/readable
- ISOServer now honors new keep-channels property (true|false)
- Added bind-address and backlog properties to ISOServer
- ISOServer: close ThreadPool on shutdown
- FSDMsg truncate A fields larger than length
- synchronized NameRegistrar
- GenericSubFieldPackager - pack dummy fields if field is not available and inner message has no bitmap
- QMUX now supports new <key>..</key> configuration element (defaults to <key>41, 11</key>).

- Added org.jpos.transaction.participant.HasEntry general purpose Group Selector
- TransactionManager attempts to call setTransactionManager(TransactionManager) on participants
- Bugfix in IF_TCHAR (unpack offset - r2410)
- Added Q2.shutdown (boolean join) (r2414)
- Added Join participant (r2416)
- Added keep-alive property to BaseChannel (r2417) - feature request #404
- Include a call to tx & rx.interrupt so that an idle ISOMUX will honour terminate requests.
- Added setOverrideHeader(boolean) and isOverrideHeader() to BaseChannel.
- ISOUtil.normalize uses UTF-8 encoding (r2420).
- BaseChannel: added protected int getHeaderLength(ISOMsg m).
- TransactionManager.stop() wait (via join) for sessions to terminate (r2422)
- ISOServer/ThreadPool: Added socket closure in the event of ThreadPool exhaustion. Combined with a +tightening of count of active, working and available Threads in ThreadPool itself (r2423).
- Added IFA_LLLLLLCHAR field packager (r2424)
- Bugfix in DailyTaskAdaptor in order to deal with daylight saving change (r2425)
- XMLPackager honors optional <header>...</header> element at unpack time. (r2426)
- bugfix in ChannelAdaptor: honor reconnect delay on peer-disconnect exceptions (r2429)
- Added new <ignore-rc> configuration element to QMUX (r2432)
- QMUX added removeISORequestListener (r2435)
- IFB_LLLHCHAR, IFB_LLLHECHAR and IFB_LLLHBINARY length can go up to 65535 (r2437)
- ISOBasePackager - add support for composite MTIs (r2438)
- BaseChannel accepts new max-packet-length optional property - new default size 100k (r2441)
- bugfix: ISOServer clones max-packet-length (r2442)
- Added new MUXPool component (r2443)
- TransactionManager minor optimization - do not take snapshot if psp is not actually persistent (r2444)
- SpaceUtil new wipeAndOut helpers (r2447)
- TransactionManager - new PAUSE support (r2448) - Continuations
- bugfix: GZipChannel compressed/uncompressed length (r2458)
- TransactionManager new method getOutstandingTransactions (r2459)
- TransactionManager new method getOutstandingTransactions (r2459)

- added org.jpos.transaction.participant.Forward (r2459)
- added org.jpos.transaction.participant.CheckPoint (r2460)
- QFactory.properties added txnmgr and transaction-manager aliases (r2462)

F.6. jPOS 1.6.1

(aka jPOS 6.1)

- added ISOCurrency.getCurrency(String) (r2466)
- added ISOPackager.createISOMsg() factory method (r2469)
- new SpaceTap class (r2475)
- Added IFA_LLLLBINARY (r2480)

Appendix G. Software Copyright

```
jPOS Project [http://jpos.org]
Copyright (C) 2000-2008 Alejandro P. Revilla

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Appendix H. GNU Affero General Public License version 3

Version 3, 19 November 2007

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU Affero General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you

or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment

not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work

with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a “Source” link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see http://www.gnu.org/licenses/.

Appendix I. Acknowledgments

We'd like to express our gratitute to the following projects and individuals

- The DocBook-xsl project crew.
- Christian Bauer from the Hibernate project for his work dealing with DocBook-xsl configuration.
- Dave Bergert, from Tranvia Inc. for his extensive review and multiple corrections.
- Alwyn Schoeman (jPOS team member) for his review, feedback and multiple corrections.
- Brian Berkness (Countrywide.com), Mark Salter (mark_salter@sourceforge.net) and David Rosenstark (corp.idt.net) for their feedback and corrections.
- My partner Daniel Larrosa (dflc@cs.com.uy) for his extensive support and for keeping our production systems running and our customers happy while I was writting this documentation.
- Special thanks go to Andy Orrock [http://andyorrock.typepad.com/] for his continuous support and feedback to the project and for his extensive review of its documentation.
- Victor Salaman, Eoin Flood, Arun Kumar, Bharavi Gade, Rajal Shah, Hani S. Kirollos, Jonathan O'Connor, Cameron Young, Alireza Taherkordi, Andres Alcarraz, Alwyn Schoeman, Carlos Quiroz, Kris Leite, Luar Roji, Dave Bergert, Jeff Gordy, Thomas L. Kjeldsen, Adrian Marques, Anthony Schexnaildre, Mark Salter, Mike Trank, Jonathan Easterling, Matias Salvador, Julien Moebs, Zhiyu Tang, Henry Chan, Jose Eduardo Leon, Christopher Harris, Bernardinus, Niclas Hedhman, Stefano Arosio, George McKinney, Matias Crespillo, Vincent Green, Leonard Thomas, Bassam Al Arab, Murtuza Chhil, Mladen Mrkic, Hitesh Sharma, Chris Lemper, Andrew R. Rothwell, Matthew Milliss, Morgan Hankins, Ozzy Espaillat, Alexei Gladkov, Alexander Fedorenko, Nathan Brice Mcafee, Bulent Erdemir, Nigel Smith, Paul W. Cowan, Norbert Kroemeke, Alan Honczar, Robbie Robinson and the jPOS community of over 1300 developers that actively participate in our mailing lists.
- The OpenSource community in general and the Apache Software foundation in particular, for providing the supporting tools and libraries on which we rely.

Index

B

BaseChannel, 51
Blog, 1
Building
  Q2, 62
  Q2 modules, 62
Building jPOS, 4

C

ChangeLog, 176
  1.4.9, 176
  1.5.0, 177
  1.5.1, 178
  1.5.2, 179
  jPOS 1.6.0, 179
  jPOS 1.6.1, 182
ChannelAdaptor, 75
ChannelPool, 54
Configurable, 39
Configuration, 39
Copyright, 1, 183

D

Deprecated, 117
Directory
  Polling, 43
Directory Structure, 2
DirPoll, 43
  Adaptor, 80
Downloading jPOS, 1

E

Eclipse, 5

F

Field types, 9
FilteredChannel, 26
Flow, Message, 13

G

GenericPackager, 47

I

IDE, 5
IDEA, 5
ISO-8583, 7
  Message Flow, 13
  Message Structure, 7
  Wire Protocol, 12
ISOChannel, 22, 51
ISOClientSocketFactory, 54
ISOField, 16
ISOFieldPackager, 16
ISOFilter, 26
ISOMsg, 16
ISOMUX, 30
ISOPackager, 20, 46
ISOServer, 27
ISOServerSocketFactory, 54

L

Legacy, 117
License, 1, 1, 184
LogEvent, 34
Logger, 34
LogSource, 34

M

Maling lists, 1
Message Flow, 13
Monitor
  SystemMonitor, 40
Multiplexer, 30
MUX, 30, 78

N

NameRegistrar, 37
NetBeans, 5

P

Profiler, 42

Q

Q2, 62, 62, 62
  Primer, 62
  XML Descriptors, 69
QBean Descriptors, 69
QMUX, 30, 78
QServer, 79

R

ReConfigurable, 39
RSS feeds, 1
Running jPOS, 6

S

Server, 27
SimpleConfiguration, 39
singleton, 37
Spaces, 56

SpaceTap, 60
SSL, 54
SystemMonitor, 40

T

ThreadPool, 45

W

Wire Protocol, 12

X

XML QBean Descriptors, 69