CAROL Library User Manual

CAROL (Common Architecture for RMI ObjectWeb Layer), a RMI manager

Guillaume Riviere

Guilllaume.Riviere@inrialpes.fr

CAROL Library User Manual: CAROL (Common Architecture for RMI ObjectWeb Layer), a RMI manager

by Guillaume Riviere

CAROL is a library allowing to use different RMI implementations. Thanks to CAROL, a Java server application can be independent of RMI implementations and accessible simultaneously by RMI clients using different RMI implementations. CAROL allows to design, implement, compile, package, deploy, and execute distributed applications compliant with the RMI model.

Table of Contents

List of Tables

List of Figures

List of Examples

Chapter 1. Introduction

1.1. About this manual

This manual was meant as a tutorial that can give you an introduction on how to use the CAROL RMI IIOP library.

> **Note:** Please note that this manual is designed to be used along with, not instead of, the RMI IIOP Tutorial (http://java.sun.com/j2se/1.4/docs/guide/rmi-iiop/tutorial.html) and the JNDI tutorial (http://java.sun.com/products/jndi/tutorial). There are a number of cases where it is much easier to refer to the rather RMI IIOP and JNDI tutorials than trying to rehash what it already covers.

This manual will teach you the general way to use the CAROL abstraction in order to manipulate remote object on multi-RMI architecture. You will learn in particulars:

- The CAROL configuration rules for each RMI architecture,
- the RMI IIOP general mechanism and programming rules,
- the JNDI general mechanism and programming rules,
- the Extended RMI JRMP mechanisms for implicit context propagation.

1.2. What is CAROL?

CAROL is a library allowing to use different RMI implementations. Thanks to CAROL, a Java server application can be independent of RMI implementations and accessible simultaneously by RMI clients using different RMI implementations. CAROL allows to design, implement, compile, package, deploy, and execute distributed applications compliant with the RMI model. CAROL provide tools for accessing to a Java server, in the same time, through the ObjectWeb JEREMIE RMI like RPC, through the JAVA standard RMI RPC and through a CORBA RPC (via a RMI IIOP). Therefore, a Java server using CAROL manipulates remote object only through RMI IIOP API classes and interfaces and never through CAROL classes or interface. So, CAROL allows a Java server to be independent, by configuration, of the RMI architecture and provider.

The CAROL library basically provides support (CAROL basic SPIs) for the following RMI implementations:

- ObjectWeb JEREMIE (JRMP 1.1 and 1.2)
- Sun RMI JRMP (JRMP 1.1 and 1.2)
- Sun JDK 1.4 RMI IIOP
- CAROL LMI implementation
- CAROL CMI implementation

The CAROL library provides also non standard tools for RMI and JNDI architecture:

- a set of mechanisms for implicit context propagation in RMI JRMP,
- a set of mechanisms for Referenceable and Reference objects binding in a CosNaming.
- a set of mechanisms for RMI Registry, Jeremie Registry and CosNaming management.

Please see the Non standard CAROL tools and mechanisms chapter for more information.

1.3. Why CAROL?

CAROL is basically design to be a solution for implementing J2EE specifications on interoperability and implicit context propagation. This library allows a J2EE server to be accessible, at the same time, by IIOP and JRMP clients.

Chapter 2. CAROL overview

2.1. Presentation of the CAROL library

This section describes the general CAROL architecture and development rules. CAROL is based on an API/SPI mechanism for export and registering RMI objects. This section describes which API are used by CAROL and how to develop a server using this API. This section is supposed to be used with the RMI IIOP tutorial (http://java.sun.com/j2se/1.4/docs/guide/rmi-iiop/tutorial.html) and the JNDI tutorial (http://java.sun.com/products/jndi/tutorial).

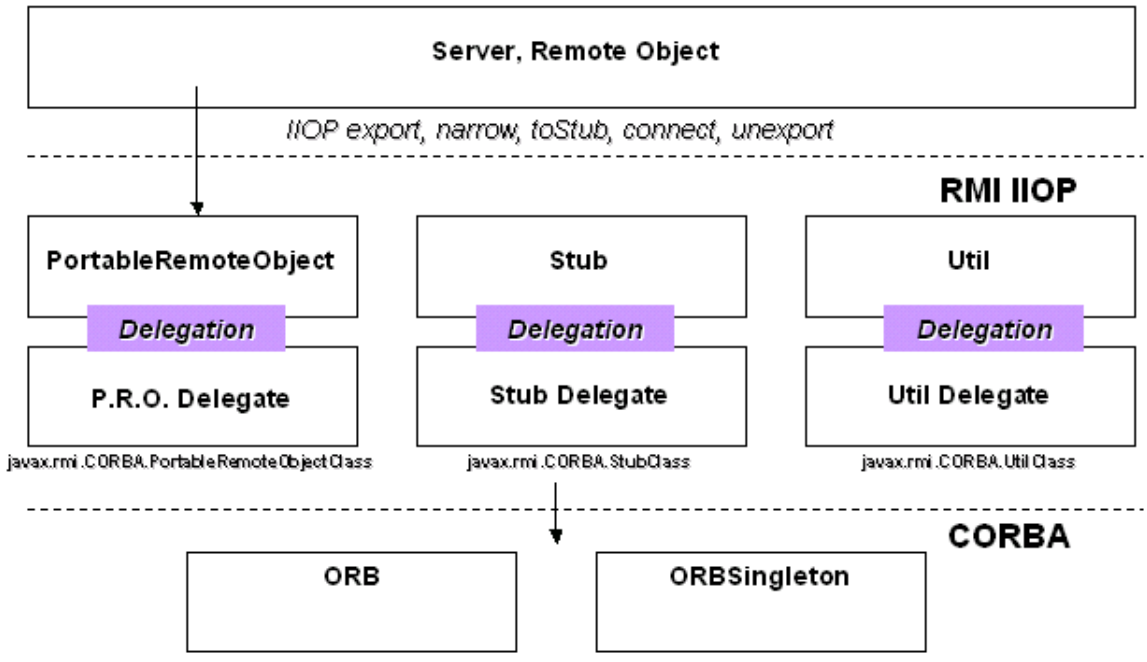
2.2. CAROL standard architecture

2.2.1. CAROL architecture

CAROL is behind the standard RMI IIOP and JNDI API. A Java server using CAROL have to be a standard RMI IIOP server and use only the JNDI interfaces for name service connections (see the section RMI IIOP Development rules and the JNDI Development rules section). A standard RMI IIOP server is required to migrate to the CAROL library without any code modification. Using CAROL library, in this case, is only a configuration manipulation. CAROL simulates a standard RMI IIOP PortableRemoteObjectDelegate and a standard JNDI context factory for interceptions and manipulations of the RPC and naming mechanism. CAROL allows any RMI IIOP remote object to be manipulate by a server on different RMI architectures and different naming services, in the same time, without code modification on the server or on the client side.

CAROL uses the standard RMI IIOP PortableRemoteObject to abstract the export mechanism. The figure 2.1 shows that the server only manipulate remote object via the RMI IIOP PortableRemoteObject and this PortableRemoteObject is a delegation to a "configured by system properties" PortableRemoteObjectDelegate class.

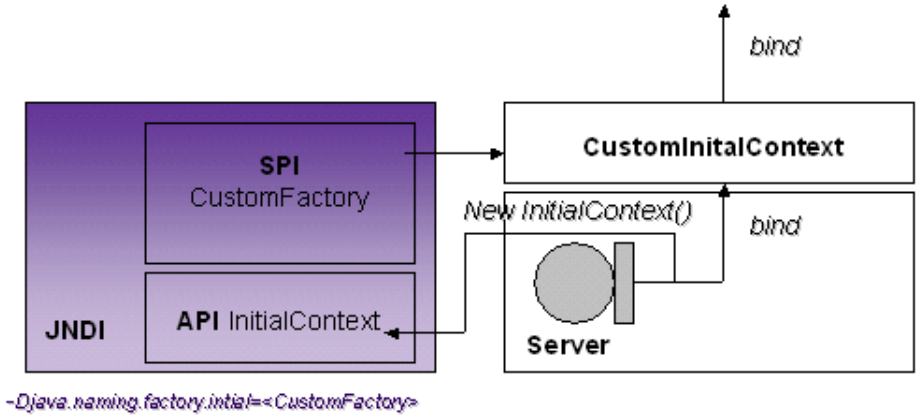
Figure 2-1. RMI IIOP mechanism



In fact, the CAROL remote object API is the standard RMI IIOP API. A Java server using CAROL is supposed to use only the java.rmi.* and the javax.rmi.* classes and never to call directly the CAROL library classes.

The same mechanism is used for objects registering through JNDI: a CAROL server is supposed to use only the JNDI interface to manipulate and contact the remote object name service. So, with JNDI, a CAROL server use the InitialContext mechanism, for (un)registering object, and this InitialContext delegate the registering to a context object build by a factory "configured by system properties". In the figure 2.2 we can see that the server only manipulates remote object registered via the JNDI InitialContext API.

Figure 2-2. JNDI mechanism



2.2.2. RMI IIOP development rules

This section describes the basic rules of RMI IIOP development. For more information, see the Sun entry for *RMI IIOP Documentation* (http://java.sun.com/j2se/1.4/docs/guide/rmi-iiop/index.html). This RMI IIOP quick start guide is design for a 3 step development:

- Development of the RMI IIOP remote objects and development of the RMI server part
- Java and CAROL RMI IIOP objects compilation
- Deployment step in a distributed environment

The Java classes used in this section are:

- `java.rmi.Remote` (http://java.sun.com/j2se/1.4/docs/api/java/rmi/Remote.html)
- `javax.rmi.PortableRemoteObject` (http://java.sun.com/j2se/1.4/docs/api/javax/rmi/PortableRemoteObject.html)

2.2.2.1. RMI IIOP remote objects development step

A RMI IIOP remote object needs only to expose its remote methods in a Java interface extending `Remote`. This is exactly the same development rules than in classical RMI JRMP. In the example 2-1, the remote object `Foo` exposes its remote method `myMethod()` in the remote interface `FooRemoteInterface`.

Example 2-1. RMI basic example

```
//The foo object is a remote object
import java.rmi.RemoteException;

//The class foo implements only
//the FooRemoteInterface interface
public class Foo implement FooRemoteInterface {

    //This method is remote
    public Integer myMethod() throws RemoteException{
return new Integer(0);
    }
}


//The foo remote interface
//extends only the Remote interface
//and exposes the remote methods
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface FooRemoteInterface extends Remote {

    //This method is remote
    public Integer myMethod() throws RemoteException;

}
```

Note: The method `myMethod()` throws a RemoteException if an exceptions occurs in the remote method call.

2.2.2.2. RMI IIOP server development step

The RMI IIOP server has to manage remote objects. This section only describes the (un)export management of a RMI IIOP remote object. Please see the JNDI development rules section for the remote object (un)registering managment. One of the most important step in a remote object life cycle is the export step (and the opposite unexport step). To Exporte a remote object means to prepare this object to receive remote call. RMI IIOP abstracts the intricate CORBA implementation mechanism of this export with the API class `PortableRemoteObject`. To Export a remote object is mandatory for remote call. There is two way for this export:

- The implicit method: if the remote object class implements the `PortableRemoteObject` class, this remote object is automatically export in is creation time. In the example 2-2 the remote object is implicitly exported by inheritance. In this case, the server only needs to construct the remote object to exported it.

Example 2-2. RMI implicit export

```
//The foo object is a remote object
import java.rmi.RemoteException;
import javax.rmi.PortableRemoteObject;

//The class foo extends PortableRemoteObject
//and implements the FooRemoteInterface interface
public class Foo extends PortableRemoteObject
 implements FooRemoteInterface {

    //The constructor
    public Foo() throws RemoteException {
super();
    }

    //This method is remote
    public Integer myMethod() throws RemoteException {
return new Integer(0);
    }
}

//The foo remote object server
import java.rmi.RemoteException;
import org.objectweb.carol.util.configuration.CarolConfiguration;

public class Server {

    //The main method of this server
    public static void main(String [] args)  {
try {
    //initialize carol
    CarolConfiguration.init();

    FooRemoteInterface myFoo = new Foo();
    // the object is automatically
    // exported on RMI IIOP
} catch (RemoteException e) {
    //Foo construction problem
}

    }
}
```

- The explicit method: if the remote object class do not implement the `PortableRemoteObject` class, this remote object has to be explicitly exported by the server. The `public static void exportObject(java.rmi.Remote)` method in `PortableRemoteObject` class allow to do that. In the example 2-3 the remote object is explicitly exported by the server.

Example 2-3. RMI explicit export

```
//The foo object is a remote object
import java.rmi.RemoteException;

//The class foo implements
//only the FooRemoteInterface interface
public class Foo implement FooRemoteInterface {

    //The constructor
    public Foo() throws RemoteException {
super();
    }

    //This method is remote
    public Integer myMethod() throws RemoteException {
return new Integer(0);
    }
}

//The foo remote object server
import java.rmi.RemoteException;
import javax.rmi.PortableRemoteObject;
import org.objectweb.carol.util.configuration.CarolConfiguration;

public class Server {

    //The main method of this server
    public static void main(String [] args)  {
try {
    //initialize carol
    CarolConfiguration.init();

    FooRemoteInterface myFoo = new Foo();
    //The object is explicitly exported on RMI IIOP:
    PortableRemoteObject.exportObject(myFoo);
} catch (RemoteException e) {
    //Foo construction problem
}

    }
}
```

2.2.2.3. CAROL RMI IIOP compilation step

The compilation step is designed by Java and RMI. There is no particular compilation step in order to use CAROL. Therefore, you need to compile Java classes and to compile stubs and skeletons with each RMI provider compiler for each RMI architecture (IIOP, JRMP, JEREMIE ...).

2.2.2.4. CAROL RMI IIOP server deployment step

The 3 points below are mandatory for CAROL server deploying on multi-RMI architecture:

- There is 2 ways for carol initialization: first, the best way, is to call the org.objectweb.carol.util.configuration.CarolConfiguration.init() method. The second way is to set the 2 system property javax.rmi.CORBA.PortableRemoteObjectClass =org.objectweb.carol.rmi.multi.MultiPRODelegate and java.naming.factory.initial =org.objectweb.carol.jndi.spi.MultiOrbInitialContextFactory in the server JVM. This second method doesn't allows to switch off carol features by configuration: The properties carol.start.rmi=false and carol.start.jndi=false doesn't work with this configuration method.
- The `carol.properties` file can be configured (see the CAROL Configuration chapter) and visible in the JVM classpath.
- For each RMI architecture all remote objects stub and skeleton have to be visible in the classpath.

  Note: For the moment, in CAROL library, there is 3 remote architectures available (CAROL SPI implementation): IIOP, JRMP and JEREMIE. There is no, in those 3 architectures, stub/skeleton class conflicts. For example, if my remote object is Foo with FooItf remote interface:

  - The stub/skel name for IIOP are: `_FooItf_Stub`/`_Foo_Tie`
  - The stub/skel name for JRMP are: `FooItf_Stub`/`FooItf_Skel`
  - The stub/skel name for JEREMIE are: `FooItf_OWStub`/`FooItf_OWSkel`

  And so there is no class name conflict, those 3 RMI architectures can be available in the same JVM. The Java classes used in this section are:

  - `javax.naming.InitialContext` (http://java.sun.com/j2se/1.4/docs/api/javax/naming/InitialContext.html)
  - `org.objectweb.carol.rmi.jrmp.interceptor.JServerRequestInterceptor`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JServerRequestInfo`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JServiceContext`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JClientRequestInterceptor`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JClientRequestInfo`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JInitializer`
  - `org.objectweb.carol.rmi.jrmp.interceptor.JInitInfo`
  - `org.objectweb.carol.rmi.jrmp.interceptor.ProtocolInterceptorInitializer`
  - `org.objectweb.carol.jndi.iiop.IIOPContextWrapperFactory`
  - `org.objectweb.carol.jndi.iiop.IIOPContextWrapper`

Warning

But, be careful, there is stubs and/or skeletons class name conflicts for different providers of the same RMI architecture. For example, this is not possible, with CAROL, to deploy a remote object on two RMI provider with the same architecture (for example RMI JRMP 1.1 and RMI JRMP 1.2 or DAVID RMI IIOP and SUN JDK 1.4 RMI IIOP) because there are a stubs and/or skeletons class name conflicts in the server JVM.

2.2.3. JNDI development rules

This section describes the basic JNDI development rules. For more information, see the Sun entry for *JNDI Documentation* (http://java.sun.com/products/jndi/1.2/javadoc/). This JNDI start guide is designed for a 2 steps development:

- Development of the JNDI server part
- JNDI deployment step on a distributed environment

2.2.3.1. Development of the JNDI server part

For remote object access with CAROL, the first part is to develop and deploy RMI IIOP remote objects (see the RMI IIOP Development rules chapter) on a Java server. The second part is to register those objects in one/many name service through the standard JNDI Interface. For this, the server needs to build a `InitialContext` object and to register all remote objects in this context like in the Example 2-4:

Example 2-4. JNDI basic example

```
//The foo remote object server
import java.rmi.RemoteException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import org.objectweb.carol.util.configuration.CarolConfiguration;

public class Server {

    //The main method of this server
    public static void main(String [] args)  {
try {
    //initialize carol
    CarolConfiguration.init();

    // the object is automatically
    // exported on RMI IIOP
    FooRemoteInterface myFoo = new Foo();

    // now the server bind this object trough JNDI
    // with the name myobjectname
    InitialContext ic = new InitialContext();
    ic.rebind("myobjectname", myFoo)

} catch (RemoteException e) {
    //Foo construction problem
}catch (NamingException ne) {
    //Foo binding problem
}

    }
```

```
}
```

> **Note:** In this example, the server use a default `IntialContext` without configuration. You may want to configure your server JNDI for each name service (registry, cosnaming ...). Please use only the CAROL configuration to setup your multi JNDI name service. For this feature, a Java server, needs the CAROL JNDI context factory (see the CAROL Configuration chapter).

2.2.3.2. JNDI deployment step on a distributed environment

The 3 points below are mandatory for CAROL server deploying on multi name service architecture:

- The system property java.naming.factory.initial need to be instantiated to `org.objectweb.carol.jndi.spi.MultiOrbInitialContextFactory` in the server JVM.
- The `carol.properties` and the `jndi.properties` files need to be configured (see the CAROL Configuration chapter) and visible in the server JVM classpath.
- Each name service (registry, cosnaming, ...) can be launched in the distribute environment.

> **Warning**
>
> Be careful, the InitialContext need to be configured for CAROL with the system property java.naming.factory.initial instantiated to `org.objectweb.carol.jndi.spi.MultiOrbInitialContextFactory`. Every other properties configured directly in the server InitialContext will be lost. The important point is to understand that the `IntialContext` is an indirection to an other context, the CAROL one, which manage all the contexts for each name service.

2.3. Non standards CAROL tools and mechanisms

2.3.1. Implicit context propagation with RMI JRMP

One of the non standard features (API) provided by CAROL is the possibility to instantiate, for a Java server, an implicit context propagation. This API is very useful for security and transaction behavior. This API is a simplification of CORBA portable interceptor concept. Therefore, this feature use a server and client interceptor with an initializer registering mechanism. This mechanism is pure Java without any CORBA classes connection (only the `rt.jar` classes is needed). This mechanism works with 1.1 and 1.2 RMI version. This section explain the way to build, register and use server and client interceptor through RMI JRMP.

2.3.1.1. CAROL RMI JRMP interceptors API

Implementing JRMP interceptors is very easy. A server interceptor only need to implements the `JServerRequestInterceptor` interface and use the `JServerRequestInfo` interface to propagate a `JServiceContext`. On the client side this is the same concept with `JClientRequestInterceptor` interface and `JClientRequestInfo` interface. For propagation, a CAROL propagation context need only to be a Serializable (or Externalizable) object and to implements the interface `JServiceContext`.

2.3.1.2. CAROL RMI JRMP Client interceptor

A JRMP client interceptor is a class implements the `JClientRequestInterceptor`. All the methods in this class are executed in the same Thread than the client remote call. All JServiceContext registered in the JClientRequestInfo (in the send_* methods) are send to the Server. All JServiceContext send by the server can be found in the JClientRequestInfo (in the receive_* methods).

2.3.1.3. CAROL RMI JRMP Server interceptor

A JRMP server interceptor is a class implements the `JServerRequestInterceptor`. All the methods in this class are executed in the same Thread than the server remote call. All JServiceContext registered in the JServerRequestInfo (in the send_* methods) are send to the Client. All JServiceContext send by the client can be found in the JServerRequestInfo (in the receive_* methods).

2.3.1.4. CAROL RMI JRMP JServiceContext

For each call,A CAROL JServiceContext can be find with is context_id. A JServiceContext is just a Serializable Object. For performance reason, it can be interesting to decrease the Context size by using a Externalizable mechanism. Carol provide a tool (org.objectweb.carol .util.perfs .CarolJRMPPerfomanceHelper) with static methods to calculate the Serializable size of a Serializable object.

2.3.1.5. CAROL RMI JRMP interceptor registering

To register interceptor in CAROL is very easy. A server/client initializer implements the `JInitializer` and use the pre_init and post_init methods for registering server and client interceptors through `JInitInfo` interface. For JVM CAROL JRMP initialization, use the org.objectweb.PortableInterceptor.JRMPInitializerClass.XXX property where XXX is the JInitializer full classname (for example pass -Dorg.objectweb.PortableInterceptor.JRMPInitializerClass .org.objectweb.carol.rmi.jrmp.interceptor.ProtocolInterceptorInitializer register into CAROL the `ProtocolInterceptorInitializer` class). Register more than one JInitializer is possible with CAROL (The `ProtocolInterceptorInitializer` is mandatory for CAROL multi protocol management).

2.3.2. Referenceable an Reference binding through a RMI IIOP CosNaming

The second non standard CAROL features is a way to register Referenceable/Reference and Serializable objects in a CosNaming through JNDI. The IIOP InitialContext delivered for IIOP wrap the Referenceable/Reference or Serializable object into a standard remote object. This remote object is exported into the JNDI context bind(or rebind) method and unexported into the JNDI context unbind method. CAROL use automatically, on the server side, this mechanism with a standard CAROL IIOP configuration (you need to call the IIOP protocol 'iiop' in the `carol.properties` file see the CAROL configuration chapter).

For a JNDI java RMI IIOP client you can use the `IIOPContextWrapperFactory` by setting the -Djava.naming.factory.initial jvm properties (with the full name of the factory). This factory builds a JNDI Context based on you `jndi.properties` uses the wrapping mechanism. For other client (Non JNDI), you can re-build manually the Referenceable or Serializable object for the CosNaming wrapper remote object (see inside the `IIOPContextWrapper` class for a detailed mechanism. A CAROL server can be also an IIOP CAROL client without any extra configuration than in a classical IIOP CAROL server.

2.3.3. Name Service Management

The third non standard CAROL features is a way to start and stop automatically RMI Name Services. This mechanism is based on a API/SPI system. The API is represented by the `org.objectweb.carol.jndi.ns.NameServiceManager`. This class provide static methods for start and stop configured name services for each protocol. A CAROL configuration property can be set inside the carol.properties for automatically start all non started and configured Name Services (see the configuration chapter). Carol provide also three Name Service SPI implementation for RMI Registry, Jeremie Registry and CosNaming management. This mechanism start those Name Services on the port defined by the jndi url property.

2.4. Getting started conclusion

CAROL is only configured by system properties and files. There is no intrusion of CAROL classes in a standard RMI IIOP server. The server is RMI architecture independent but work simultaneously on different RMI architectures. The next chapter explains the general rules for this configuration.

Chapter 3. CAROL Configuration

3.1. Presentation of the CAROL library configuration

This section describes the configuration rules for different RMI and name services managed by CAROL. Currently, CAROL is distributed with tools and classes that allow to use:

- ObjectWeb/Jonathan JEREMIE (http://objectweb.org/jonathan) RMI JRMP like
- Sun RMI JRMP (http://java.sun.com/j2se/1.4/docs/guide/rmi/index.html) (version 1.1 and 1.2)
- Sun RMI IIOP (>JDK 1.4) (http://java.sun.com/j2se/1.4/docs/guide/idl/index.html)
- Local Method Invocation (LMI) for embedded server

CAROL allows to configure a remote (or local) Java server to be accessible by one,two or three of those RMI architectures, in the same time, by configuration.

3.2. CAROL configuration

3.2.1. General configuration files

CAROL configuration is based on three properties files. The `carol-defaults.properties` file, the `carol.properties` file and the `jndi.properties` file. The `carol-defaults.properties` file is mandatory to configure CAROL. This file is embedded in the carol jar file and the CAROL user is not suppose to modify those defaults properties. Carol load first this default file properties and erase all configuration property with the property find in the jndi file configuration and (after) with the property find in the carol file configuration. So, the only important file for the CAROL user is the `carol-defaults.properties`. We are going to describe, in the next section, only the content of this file.

> **Note:** If there is an (rpc-)URL property in the `jndi.properties`, the RMI name of the url is use instead of the `carol-defaults.properties` configured one (jrmp). By default, in the `carol.properties`, no configuration is needed.

The configuration described below only use the `carol.properties` file. Do not forget that the JNDI configuration (of one of the activated protocols) can be set inside the `jndi.properties` file.

3.2.2. General configuration rules for all RMI and JNDI architectures

The `carol.properties` file is a standard Java properties file. All properties, in this file, follow the rules below (we suppose that XXX is the RMI name like ’jrmp’, ’iiop’,’jeremie’,’cmi’ or ’lmi’. For all those defaults provided RMI protocol, please use those names.

Table 3-1. Carol general properties

Property name	Property value	Description	Required
carol.protocols	’XXX protocol-name’, ’YYY protocol-name’	Activated protocols names. The first (XXX) is the default protocol for CAROL. This default protocol is used by the server when there is no entrant protocol	No, default: jrmp
carol.start.ns	’true’, ’false’	CAROL will automatically start all non started and configured Name Services if this property is set to ’true’	No, default: false
carol.start.rmi	’true’, ’false’	The CAROL PortableRemoteObjectDelegate mechanism will be deactivated if this property is set to ’false’	No, default: true
carol.start.jndi	’true’, ’false’	The CAROL InitialContextFactory mechanism will be deactivated if this property is set to ’false’	No, default: true
carol.jvm.’property-name’	’property-value’	All extra JVM properties for RMI (directly pass to the JVM with ’name’ name and ’value’ value without any verifications). This is equivalent to put -D’property-name’=’property-value’ in the Java JVM option	No
carol.jndi.’property-name’	’property-value’	All JNDI properties for all RMI (directly pass to JNDI with ’name’ name and ’value’ value without any verifications). This is equivalent to put ’property-name’=’property-value’ in the JNDI properties file	No

Table 3-2. Carol RMI XXX specifics properties

Property name	Property value	Description	Required

Property name	Property value	Description	Required
carol.XXX .PortableRemoteObjectClass	'XXX portable remote object class name'	The portable remote object delegate class name for this RMI (CAROL provide implementation of those classes for RMI JRMP,JEREMIE, LMI and CMI, see below)	No, There is a default for each RMI provided by carol
carol.XXX .NameServiceClass	'XXX carol Name Service class name'	The Name Service class name for this RMI (CAROL provide implementation of those classes for RMI JRMP, RMI IIOP and JEREMIE, see below)	No, this property is only necessary for automatically start a Name Service for this protocol and there is a default for each RMI provided by carol
carol.XXX.url 'property-name'	'property-value'	JNDI url value. This property is equivalent to the jndi java.naming. provider.url property	No, there is defaults for each RMI provided by carol
carol.XXX.context. factory 'property-name'	'property-value'	JNDI initial context factory class name. This property is equivalent to the jndi java.naming. factory.initial property	No, there is defaults for each RMI provided by carol.
carol.XXX. interceptors	'interceptor 1 name','interceptor 2 name'	Interceptors initializers names	No

3.2.3. RMI JRMP configuration

One of the SPI personality provided by CAROL is the standard Sun RMI JRMP. This personality can be used with all standard RMI JRMP features. CAROL allow implicit context propagation with RMI JRMP (like a transactional or a security context) via a RMI IIOP Interceptors like mechanism. The example below explains the general way for CAROL RMI JRMP configuration:

Example 3-1. RMI JRMP `carol.properties` file.

```
# activated protocols
carol.protocols=jrmp

# Example of Interceptors initializer class
carol.jrmp.interceptors=
 org.objectweb.carol.jtests.conform.interceptor.jrmp.Initializer
```

In the file above we see a JRMP standard configuration, note that nothing is needed except custom interceptors configuration. You can customized your configuration with:

Table 3-3. Carol RMI JRMP specifics properties

Property name	Property value	Description	Required
carol.jrmp. PortableRemoteObjectClass	'JRMP portable remote object class name'	JRMP implementation of the Portable Remote Object class name	No, there is a default for JRMP
carol.jrmp. NameServiceClass	'JRMP carol Name Service class name'	JRMP implementation of the Carol name service	No, there is a default for JRMP based on the registry
carol.jvm.jrmp. server.portnumber	'jrmp port number'	All jndi properties needed for XXX name service configuration. This is equivalent to put 'property-name'='property-value' in the jndi.properties file	No, there is a default for JRMP (0)
carol.jvm.org. objectweb. PortableInterceptor. JRMPInitializerClass. 'jrmp initializer name'	'' (empty)	Initializer for JRMP inteceptors	No, there is defaults for JRMP. If there is a multi protocol configuration, CAROL automaticaly put interceptors for multi protocol management. This property is equivalent, for jrmp, to the carol.jrmp .interceptors one

3.2.4. RMI IIOP configuration

One of the SPI personality provided by CAROL is the standard Sun RMI IIOP. This personality can be used with all standard RMI IIOP features. CAROL allow implicit context propagation with RMI IIOP (like a transactional or a security context) via a RMI IIOP Interceptors mechanism. The example below explains the general way for CAROL RMI IIOP configuration:

Example 3-2. RMI IIOP `carol.properties` file.

```
# activated protocols
carol.protocols=iiop

# Example of Interceptors initializer class (class name with package)
carol.iiop.interceptors=
 org.objectweb.carol.jtests.conform.interceptor.iiop.IIOPInitializer
```

In the file above we see a IIOP standard configuration, note that nothing is needed except custom interceptors configuration. You can customized your configuration with:

Table 3-4. Carol RMI IIOP specifics properties

Property name	Property value	Description	Required
carol.iiop. PortableRemoteObjectClass	'IIOP portable remote object class name'	IIOP implementation of the Portable Remote Object class name	No, there is a default for IIOP
carol.iiop. NameServiceClass	'IIOP carol Name Service class name'	IIOP implementation of the Carol name service	No, there is a default for IIOP based on the tnameserv CosNaming
carol.jvm.org.objectweb. PortableInterceptor. IIOPInitializerClass. 'iiop initializer name'	" (empty)	Initializer for IIOP inteceptors	No, there is defaults for IIOP. If there is a multi protocol configuration, CAROL automaticaly put interceptors for multi protocol management. This property is equivalent, for iiop, to the carol.iiop .interceptors one

3.2.5. RMI JEREMIE personality configuration

One of the SPI personality available/provided by CAROL is the ObjectWeb Jonathan JEREMIE personality. This personality can be used with all standard JEREMIE features. CAROL also allow JEREMIE to propagate implicitly a context (like a transactionnal or a security context) via a JEREMIE handler mechanism. This section explains the general way for CAROL JEREMIE configuration in the two jonathan.xml and carol.properties files:

Example 3-3. JEREMIE `jonathan.xml` file

```
<?xml version="1.0"?>
<!DOCTYPE Configuration SYSTEM "configuration.dtd">
  <CONFIGURATION>
    <ELEM name="DavidCarolHandler">
      <ATOM class="org.objectweb.carol.rmi.jonathan.david.DavidCarolHandler"/>
    </ELEM>
    <ELEM name="david/orbs/iiop/services_handler_context/1534">
      <ALIAS name="/DavidCarolHandler" />
    </ELEM>
    <ELEM name="JeremieCarolHandler">
      <ATOM
class="org.objectweb.carol.rmi.jonathan.jeremie.JeremieCarolHandler"/>
    </ELEM>
    <ELEM name="jeremie/service_handler_context/1535">
      <ALIAS name="/JeremieCarolHandler" />
    </ELEM>
    <ELEM name="jeremie/stub_factories/std">
      <CONFIGURATION>
<ELEM name="Stub name extension">
  <PROPERTY type="String" value="OW"/>
</ELEM>
      </CONFIGURATION>
```

```
    </ELEM>
  </CONFIGURATION>
```

Inside The above file, JEREMIE is configured to use the OW extension for stub/skeleton and to use the CAROL protocol handler.

Example 3-4. JEREMIE `carol.properties` file

```
# activated protocols
carol.protocols=jeremie
```

In the file above we see a JEREMIE standard configuration, note that nothing is needed except custom interceptors configuration. You can customized your configuration with:

Table 3-5. Carol RMI JEREMIE specifics properties

Property name	Property value	Description	Required
carol.jeremie. PortableRemoteObjectClass	'JEREMIE portable remote object class name'	JEREMIE implementation of the Portable Remote Object class name	No, there is a default for JEREMIE
carol.jeremie. NameServiceClass	'JEREMIE carol Name Service class name'	JEREMIE implementation of the Carol name service	No, there is a default for JEREMIE based on the jeremie registry

3.2.6. LMI personality configuration

One of the SPI personality available/provided by CAROL is the CAROL LMI personality. This personality is for, an only for, local methods call. With this personality, jndi register (and return) local java references. The Referenceable mechanism is also provide in LMI context. For the moment, LMI is only tested in a non multi protocol environment.

> **Note:** This implementation is not conform to RMI specifications: There is no serialization and the server "remote" call is executed is the client thread. This is a fake implementation, faster than standard RMI but not conform.

Example 3-5. LMI `carol.properties` file

```
# activated protocols
carol.protocols=lmi
```

In the file above we see a LMI standard configuration, note that nothing is needed. You can customized your configuration with:

Table 3-6. Carol RMI LMI specifics properties

Property name	Property value	Description	Required
carol.lmi. PortableRemoteObjectClass	'LMI portable remote object class name'	LMI implementation of the Portable Remote Object class name	No, there is a default for LMI
carol.lmi. NameServiceClass	'LMI carol Name Service class name'	LMI implementation of the Carol name service (fake)	No, there is a default for LMI based on a fake registry service

3.2.7. MULTI RMI configuration

The example below describes a general RMI configuration with 3 RMI architectures configured and 2 RMI activated (RMI IIOP and JEREMIE) and with RMI IIOP default:

Example 3-6. MULTI RMI `carol.propertiess` file

```
# carol properties
# (note that for this configuration only this
# property is needed)
carol.protocols=iiop,jeremie

# start or not all non started name services
carol.start.ns=true

# use carol rmi (Multi PORD)
carol.start.rmi=true

# use carol naming (Multi JNDI)
carol.start.jndi=true

#####################################################
# Configuration for Rmi JRMP                        #
#####################################################

# portable remote object delegate class
carol.rmi.PortableRemoteObjectClass
 =org.objectweb.carol.rmi.multi.JrmpPRODelegate

# Name service class for this protocol
carol.rmi.NameServiceClass
 =org.objectweb.carol.jndi.ns.JRMPRegistry

# here, for jndi we take the jndi.properties but
# we can make some :
```

```
# configuration for rmi jrmp jndi
# java.naming.factory.initial property
carol.jrmp.context.factory
 =com.sun.jndi.rmi.registry.RegistryContextFactory
# java.naming.provider.url property
carol.jrmp.url
 =rmi://localhost:1099

# port number
carol.jvm.jrmp.server.portnumber=10

#######################################################
# Configuration for Rmi IIOP                          #
#######################################################

# portable remote object delegate class for this protocol
carol.iiop.PortableRemoteObjectClass
 =com.sun.corba.se.internal.javax.rmi.PortableRemoteObject

# Name service class for this protocol
carol.iiop.NameServiceClass
 =org.objectweb.carol.jndi.ns.IIOPCosNaming

# configuration for rmi jrmp jndi
# java.naming.factory.initial property
carol.iiop.context.factory
 =org.objectweb.carol.jndi.iiop.IIOPReferenceContextWrapperFactory
# java.naming.provider.url property
carol.iiop.url
 =iiop://localhost:2000

#######################################################
# Configuration for JEREMIE                           #
#######################################################

# portable remote object delegate class for this protocol
carol.jeremie.PortableRemoteObjectClass
 =org.objectweb.carol.rmi.multi.JeremiePRODelegate

# Name service class for this protocol
carol.jeremie.NameServiceClass
 =org.objectweb.carol.jndi.ns.JeremieRegistry

# here, for jndi we take the jndi.properties but we can make some :
# configuration for rmi jrmp jndi
# java.naming.factory.initial property
carol.jeremie.context.factory
 =org.objectweb.jeremie.libs.services.registry.jndi.JRMIInitialContextFactory
# java.naming.provider.url property
carol.jeremie.url
 =jrmi://localhost:2001

#######################################################
# Configuration for CMI                               #
#######################################################

# portable remote object delegate class for this protocol
```

```
carol.cmi.PortableRemoteObjectClass
 =org.objectweb.carol.rmi.multi.CmiPRODelegate

# Name service class for this protocol
carol.cmi.NameServiceClass
 =org.objectweb.carol.jndi.ns.CmiRegistry

# here, for jndi we take the jndi.properties but we can make some :
# configuration for rmi jrmp jndi
# java.naming.factory.initial property
carol.cmi.context.factory
 =org.objectweb.carol.cmi.jndi.CmiInitialContextFactory

# java.naming.provider.url property
carol.cmi.url
 =cmi://localhost:2002

#####################################################
# Configuration for LMI                             #
#####################################################

# portable remote object delegate class for this protocol
carol.lmi.PortableRemoteObjectClass
 =org.objectweb.carol.rmi.multi.LmiPRODelegate

# Name service class for this protocol
carol.lmi.NameServiceClass
 =org.objectweb.carol.jndi.ns.LmiRegistry

# here, for jndi we take the jndi.properties but we can make some :
# configuration for rmi jrmp jndi
# java.naming.factory.initial property
carol.lmi.context.initial
 =org.objectweb.carol.jndi.lmi.LmiInitialContextFactory

# java.naming.provider.url property
# (only for carol, no importance)
carol.lmi.url
 =lmi://nohost:0

#####################################################
# Configuration for Interceptor                     #
#####################################################

# xtra properties for the jvm (only in use in the multi protocol case)

# Protocol Interceptors initializer class
carol.jrmp.interceptors=
 org.objectweb.carol.interceptor.myJRMPInterceptorInitializer

# Protocol Interceptors initializer class
carol.iiop.interceptors=
 .org.objectweb.carol.interceptor.myIIOPInterceptorInitializer

#####################################################
# Configuration for Global JNDI                     #
#####################################################
```

```
# note that all other jndi properties than
# url and context factory can be found in
# the jndi.properties file or in the jvm
# (like this one)
carol.jndi.java.naming.factory.url.pkgs
 =org.objectweb.carol.naming
```

Chapter 4. CAROL requirements

This chapter describe the system requirements for CAROL.

Web sites index

General CAROL requirements

Java environment

- A CAROL Java server need a JDK 1.2 or greater
- A CAROL Java server need the `carol.jar` file in it's classpath

CAROL RMI JRMP requirements

Server Environement

- A CAROL RMI JRMP Java server need a JDK 1.2 or greater

Client Environement

- A CAROL RMI JRMP Java client need a JDK 1.2 or greater
- A CAROL RMI JRMP Java client need the `carol.jar` file in it's classpath

CAROL RMI IIOP requirements

Server Environement

- A CAROL RMI IIOP Java server need a JDK 1.4 or a 2.6 CORBA with RMI IIOP

Client Environement

- A CAROL IIOP client need a CORBA 2.6
- A CAROL RMI IIOP Java client need a JDK 1.4 or a 2.6 CORBA with RMI IIOP and the `carol.jar` file in it's classpath

CAROL JEREMIE requirements

Server Environement

- A CAROL JEREMIE Java server need a JDK 1.2 and a Jonathan 3.0 alpha10 or greater

Client Environement

- A CAROL JEREMIE Java client need a JDK 1.2 and a Jonathan 3.0 alpha10 or greater

Chapter 5. Links and Reference

Web site list and book reference

Web sites index

ObjectWeb web sites

ObjectWeb
- ObjectWeb main web site

CAROL
- CAROL ObjectWeb web site

JONATHAN
- JONATHAN ObjectWeb web site

Monolog
- Monolog ObjectWeb web site

SUN web sites

Java Sun
- Java Sun main web site

JDK 1.4
- Java JDK 1.4 API

RMI
- RMI documentation and tutorial web site

RMI IIOP
- RMI IIOP documentation and tutorial web site

OMG web sites

OMG
- OMG main web site

CORBA web page
- CORBA web page

PortableInterceptor
- PortableInterceptor documentation

Appendix A. Licence

This document is released under Free Documentation licence; the terms of this licence are detailed below.

A.1. Free Documentation Licence

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give

them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers

or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section
may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements"
or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http:///www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we

recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.

Glossary

RMI

(Remote Method Invocation) This is the standard specifications of the Java RPC.

RPC

(Remote Procedure Call) all remote method call protocol is a RPC.

JVM

(Java Virtual Machine) The Java virtual machine.

JDK

(Java Development Kit) A set a Java tools (compiler, jvm, library ...) for Java programs development.

API

(Application Programming Interface) Interfaces allowing to use library in programs.

SPI

(Service Provider Interface) Interface for provider library plugging in an other library.

JNDI

(Java Naming Directory Interface) Standard API/SPI for J2EE naming interface .

OMG

(Object Management Group) Industrial group for computer standard specifications.

CORBA

(Common Object Request Broker Architecture) OMG RPC specification.

IIOP

(Inter-operable Internet Object Protocol) CORBA RPC standard protocol on TCP/IP

JRMP

(Java Remote Method Protocol)Java RMI standard protocol