The Ivy C++ and java library guide

CENA NT02-819

Yannick Jestin

jestin@cena.fr

This document is a programmer's guide that describes how to use the Ivy Java library to connect applications to an Ivy bus. This guide describes version 1.2 of the library. This document itself is part of the java package, available on the Ivy web site (http://www.tls.cena.fr/products/ivy/).

1. Foreword

This document was written in SGML according to the DocBook DtD, so as to be able to generate PDF and html output. However, the authors have not yet mastered the intricacies of SGML, the DocBook DtD, the DocBook Stylesheets and the related tools, which have achieved the glorious feat of being far more complex than LaTeX and Microsoft Word combined together. This explains why this document, in addition to being incomplete, is so ugly. We'll try and improve it.

The Windows ivy-c++ port has been written with the same API. Most of the documentation for the Ivy java library applies to the windows c++ library. There is a section dedicated to the description of the intrinsics. There is also a unix port of this library, which is a C++ wrapper on top of the C library. There is also a section dedicated to this port.

2. What is Ivy?

Ivy is a software bus designed at CENA (http://www.cena.fr/) (France). A software bus is a system that allows software applications to exchange information with the illusion of broadcasting that information, selection being performed by the receiving applications. Using a software bus is very similar to dealing with events in a graphical toolkit: on one side, messages are emitted without caring about who will handle them, and on the other side, one decide to handle the messages that have a certain type or follow a certain pattern. Software buses are mainly aimed at facilitating the rapid development of new agents, and at managing a dynamic collection of agents on the bus: agents show up, emit messages and receive some, then leave the bus without blocking the others.

Ivy is implemented as a collection of libaries for several languages and platforms. If you want to read more about the principles Ivy before reading this guide of the java library, please refer to *The Ivy sofware bus: a white paper*. If you want more details about the internals of Ivy, have a look at *The Ivy architecture and protocol*. And finally, if you are more interested in other languages, refer to other guides such as *The Ivy Perl library guide*, or *The Ivy C library guide*. All those documents should be available from the Ivy Web site (http://www.tls.cena.fr/products/ivy/).

3. The Ivy java library

3.1. What is it?

The Ivy java library (aka ivy-java or fr.dgac.ivy) is a java package that allows you to connect applications to an Ivy bus. You can use it to write applications in java. You can also use it to integrate any thread-safe java application. So far, this library has been tested and used on a variety of java virtual machines (from 1.1.7 to 1.4.1), and on a variety of architectures (GNU/Linux, Solaris, Windows NT,XP,2000, MacOSX).

The Ivy java library was originally developed by François-Régis Colin and then by Yannick Jestin at CENA. It is maintained by a group at CENA (Toulouse, France)

3.2. Getting and installing the Ivy Java library

You can get the latest versions of the Ivy C library from the Ivy web site (http://www.tls.cena.fr/products/ivy/). It is packaged either as a jar file or as a debian package. We plan to package it according to different distribution formats, such as .msi (Windows) or .rpm (Redhat and Mandrake linux).

The package is mainly distributed as a JAR file. In order to use it, either add it in your CLASSPATH, or put the jar in your $JAVA_HOME/jre/lib/ext/ directory, if you use a java 2 virtual machine. If running windows, be sure to add it to the right place for runtime (C:\Program Files\JavaSoft\...). The package contains the documentation, the sources and the class files for the fr.dgac.ivy package.

In order to test the presence of Ivy on your system once installed, run the following command:

```
$ java fr.dgac.ivy.Probe
```

If it spawns a line about broadcasting on a strange address, this is OK, it is ready and working. If it complains about a missing class (java.lang.NoClassDefFoundError), then you have not pointed your virtual machine to the jar file.

4. Your first Ivy application

We are going to write a "Hello world translater" for an Ivy bus. The application will subscribe to all messages starting with "Hello", and re-emit them after translating "Hello" into "Bonjour". In addition, the application will quit when it receives any message containing exactly "Bye".

4.1. The code

Here is the code of "ivyTranslater.java":

```
import fr.dgac.ivy.* ;

class ivyTranslater implements IvyMessageListener {

  private Ivy bus;

  ivyTranslater() {
    // initialization
    bus = new Ivy("IvyTranslater","Hello le monde",null);
    bus.bindMsg("^Hello(.*)",this);
    bus.bindMsg("^Bye$",new IvyMessageListener() {
      // callback for "Bye" message
      public void receive(IvyClient client, String[] args) {System.exit(0);}
    });
    try {
       // starts the bus on the default domain or IVY_DOMAIN property
       bus.start(null);
    } catch (IvyException ie) {
      System.err.println("can't run the Ivy bus" + ie.getMessage());
    }
  }

  // callback associated to the "Hello" messages"
  public void receive(IvyClient client, String[] args) {
    bus.sendMsg("Bonjour"+((args.length>0)?args[0]:""));
  }

  public static void main(String args[]) { new ivyTranslater(); }
}
```

4.2. Compiling it

You should be able to compile the application with the following command (if the ivy-java jar is in your development classpath):

```
$ javac ivyTranslater.java
```

```
$
```

4.3. Testing

We are going to test our application with **fr.dgac.ivy.Probe**. In a terminal window, launch **ivyTranslater**.

```
$ java ivyTranslater
```

Then in another terminal window, launch **java fr.dgac.ivy.Probe '(.*)'**. You are then ready to start. Type "Hello Paul", and you should get "Bonjour Paul". Type "Bye", and your application should quit:

```
$ java fr.dgac.ivy.Probe '(.*)'
you want to subscribe to (.*)
broadcasting on 127.255.255.255:2010
IvyTranslater connected
IvyTranslater subscribes to ^Bye$
IvyTranslater subscribes to ^Hello(.*)
IvyTranslater sent 'Hello le monde'
Hello Paul
-> Sent to 1 peers
IvyTranslater sent 'Bonjour Paul'
Bye
-> Sent to 1 peers
IvyTranslater disconnected
<Ctrl-D>
$
```

5. Basic functions

5.1. Initialization and Ivy threads

Initializing a java Ivy agent is a two step process. First of all, you must create an `fr.dgac.ivy.Ivy` object. Once this object is created, you can add subscriptions to Ivy events, be it messaged, arrival or departure of other agents, etc, but your agent is still not connected. In order to connect, you should call the `start()` method on your Ivy object. This will run two threads that will remain active until you call the `stop()` method on your Ivy object. Once the `start()` method has been called, your agent is ready to handle messages on the bus !

Here are more details on those functions:

```
fr.dgac.ivy.Ivy(String name,String message, IvyApplicationListener appcb)
```

This constructor readies the structures for the software bus connexion. It is possible to have different busses at the same time in an application, be it on the same bus or on different ivy busses. The *`name`* is the name of the application on the bus, and will by transmitted to other application, and possibly be used by them. The *`message`* is the first message that will be sent to other applications, with a slightly different broadcasting scheme than the normal one (see *The Ivy architecture and procotol* document for more information). If *`message`* is null, nothing will be sent. *`appcb`*, if non null, is an object implementing the IvyApplicationListener interface. Its different methods will be called upon arrival or departure of an agent on the bus, when your application itself will leave the bus, or when a direct message will be sent to your application.

```
public void start(String domainbus) throws IvyException
```

This method connects the Ivy bus to a domain or list of domains. *`domainbus`* is a string of the form 10.0.0:1234, it is similar to the netmask without the trailing .255. This will determine the meeting point of the different applications. Right now, this is done with an UDP broadcast. Beware of routing problems ! You can also use a comma separated list of domains, for instance "10.0.0.1234,192.168:3456". If the domain is *`null`*, the API will check for the property *`IVY_DOMAIN`*, if not present, it will use the default bus, which is 127.255.255.255:2010, and requires a loopback interface to be active on your system. This method will spawn two threads, one listening to broadcasts from other agents, and one listening on the service UDP socket, where remote agent will come and connect. If an IvyException is thrown, your application is not able to talk to the domain bus.

```
public void stop()
```

This methods stops the threads, closes the sockets and performs some clean-up. You can reconnect to the bus by calling `start()` once again.

5.2. Emitting messages

Emitting a message is much like writing a string on a output stream. The message will be sent if you are connected to the bus and somebody is interested in its content.

```
public int sendMsg(String message)
```

Will send each remote agent the substring in case there is a regexp matching. The int result is the number of messages actually sent. The main issue here is that the sender ivy agent is the one who takes care of the regexp matching, so that only useful information are conveyed on the network.

5.3. Subscribing to messages

Subscribing to messages consists in binding a callback function to a message pattern. Patterns are described by regular expressions with captures. When a message matching the regular expression is detected on the bus, the callback function is called. The captures (ie the bits of the message that match the parts of regular expression delimited by brackets) are passed to the callback function much like

options are passed to main. Use the `bindMsg()` method to bind a callback to a pattern, and the `unbindMsg` method to delete the binding.

```
public int bindMsg(String regex, IvyMessageListener callback);
public void unBindMsg(int id);
```

The *`regex`* follows the gnu.regexp regular expression syntax. Grouping is done with parenthesis. The *`callback`* is an object implementing the IvyMessageListener interface, with the `receive` method. The thread listening on the connexion with the sending agent will execute the callback.

6. Advanced functions

6.1. fr.dgac.ivy.Probe utility

fr.dgac.ivy.Probe is your swiss army knife as an Ivy java developper. Use it to try your regular expressions, to check the installation of the system, to log the messages, etc.

The command line options (available with the --help switch) are the following:

- -b allows you to specify the ivy bus. This overrides the -DIVY_BUS java property. The default value is 127.255.255.255:2010.
- -n NAME allows you to specify the name of this probe agent on the bus. It defaults to JPROBE, but it might be difficult to differenciate which jprobe sent which message with a handful of agents with the same name
- -q allows you to spawn a silent jprobe, with no terminal output.
- -d allows you to use JPROBE on debug mode. It is the same as setting the VY_DEBUG property (java -DIVY_DEBUG fr.dgac.ivy.Probe is the same as java fr.dgac.ivy.Probe -d)
- -h dumps the command line options help.

The run time commands are preceded by a single dot (.) at the beginning of the line. Issue ".help" at the prompt (without the double quotes) to have the list of availables comands. If the lines does not begin with a dot, jprobe tries to send the message to the other agents, if their subscriptions allows it. The dot commands are the following

- .die CLIENTNAME issues an ivy die command, presumably forcing the first agent with this name to leave the bus
- .bye (or .quit) forces the JPROBE application to exit. This is the same as issing an end of file character on a single input line (^D).
- .direct client id message sends the direct message to the remote client, using the numeric id
- .list gives the list of clients seen on the ivy bus

- .ping issues a ping request. This is only available for ivy java clients so far, and allows you to try and send a packet to a remote agent, in order to check the connectivity.

6.2. fr.dgac.ivy.IvyDaemon utility

As the launching and quitting of an ivy bus is a bit slow, it is not convenient to spawn an Ivy client each time we want to send a simple message. To do so, we can use the IvyDaemon, which is a TCP daemon sitting and waiting on the port 3456, and also connected on the default bus. Each time a remote application connects to this port, every line read until EOF will be forwarded on the bus. The standard port and bus domain can be overriden by command line switches. (java fr.dgac.ivy.IvyDaemon -h).

First, spawn an ivy Damon: $ java fr.dgac.ivy.IvyDaemon

then, within your shell scripts, use a short tcp connexion (for instance netcat) : $ echo "hello world" | nc -q 0 localhost 3456

The message will be sent on the default Ivy Bus.

6.3. Direct messages

Direct messages is an ivy feature allowing the exchange of information between two ivy clients. It overrides the subscription mechanism, making the exchange faster (there is no regexp matching, etc). However, this features breaks the software bus metaphor, and should be replaced with the relevant bounded regexps, at the cost of a small CPU overhead. The full direct message mechanism in java has been made available since the ivy-java-1.2.3.

7. Ivy c++ Windows port

The API is very similiar to the java port, that's why we include this little section within the ivy java documentation. The author is not familiar with windows programming or C++ programming so that this documentation might be inaccurate. Here is a sample listing that might be useful:

```
// ivytest.cpp : Defines the entry point for the console application.
#include <iostream.h>

#include <stdlib.h>

#include "ivy.h"
#include "IvyApplication.h"

static bool TheGrassIsGreenAndTheWindBlows = true;
```

```
class cIvyTranslater : public IvyApplicationCallback
{
public:
    cIvyTranslater(void);
protected:
void OnApplicationConnected   ( IvyApplication *app );
void OnApplicationDisconnected( IvyApplication *app );
void HelloCallback  ( IvyApplication *app, int argc, const char **argv );
void ByeCallback    ( IvyApplication *app, int argc, const char **argv );
Ivy *bus;
};



cIvyTranslater::cIvyTranslater(void)
{
    // initialization
bus = new Ivy( "cIvyTranslater","cIvyTranslater READY",this,FALSE);

    int count;
    count = bus->BindMsg( "^Hello(.*)", BUS_CALLBACK_OF(cIvyTranslater, HelloCallback ));
    count = bus->BindMsg( "^Bye$",      BUS_CALLBACK_OF(cIvyTranslater, ByeCallback ));

    bus->start("127.255.255.255:2010");
}

void cIvyTranslater::HelloCallback(IvyApplication *app, int argc, const char **argv)
{
    const char* arg = (argc < 1) ? "" : argv[0];
cout << "cIvyTranslater received msg: Hello'" << arg
<< "'" << endl;
    bus->SendMsg( "Bonjour%s!", arg );
}

void cIvyTranslater::ByeCallback(IvyApplication *app, int argc, const char **argv)
{
cout << "cIvyTranslater stops bus" << endl;
    if (bus) {
        TheGrassIsGreenAndTheWindBlows = false;
        bus->stop();
        delete bus; // This statement is never reached! Don't know why!
    }

}

void cIvyTranslater::OnApplicationConnected(IvyApplication *app)
{
cout << "cIvyTranslater is ready to accept messages from "
<< app->GetName() << endl;
}

void cIvyTranslater::OnApplicationDisconnected(IvyApplication *app)
```

```
{
cout << "cIvyTranslater good buy '" << app->GetName()
<< "'" << endl;
}


void main(int argc, char* argv[])
{
    cIvyTranslater aIvyTL;

    while (TheGrassIsGreenAndTheWindBlows) {
        Sleep(2000);
        cout << "new cycle..." << endl;
    }
cout << "Good buy, world\n";
}
```

7.1. Win32 API

8. programmer's style guide

[to be written]

9. Contacting the authors

The Ivy java library is now maintained by Yannick Jestin. For bug reports or comments on the library itself or about this document, please send him an email at <jestin@cena.fr>. For comments and ideas about Ivy itself (protocol, applications, etc), please join and use the Ivy mailing list: <ivy@tls.cena.fr>.