PIMR 1.0.0: Administration Guide
Version1.0.0
www.gradsoft.kiev.ua

October 24, 2002

Contents

1 General information

1.1 Package Structure And Destination

PIMR Package ("Implementation Repository ", just " Repository " below) is intended to raise CORBA - services ("Implementations") accessibility . Besides you can use Repository instead of nameservice to simplify configuration of several interfacing CORBA - services . Repository functions are to trace services accessibility and to redirect requests to a running service. For this purpose Repository supports services register. Repository works under one of Unix-like OS or Windows NT. It's written in C++ using GradC++ToolBox.

1.2 Where You Can Get It

PIMR is always available in CVS ("COS/PIMR" module).

1.3 Authorship

PIMR package is developed by GradSoft LTD, homepage: http://www.gradsoft.com.ua

2 Installation

2.1 General Information

PIMR package is distributed in source codes, and to install it you should do the next :

1. Compilation -compile source code in user's environment.
2. Installation - copy executable module and other files to directories specified by user .
3. Tuning (see below).

Using make command with certain options (see below) you can:

1. compile the package;
2. install previously compiled package;
3. remove objects, created in the compiling;
4. remove copied files

2.2 Required Software

1. C++ compiler:
   - Unix: gcc
   - Windows NT: Microsoft Visual C 6.0 or higher
2. make utility:
   - Unix: gnu make is required
   - WindowsNT: nmake of MSVC++
3. GradC++Toolbox library 1.5.0 or higher
4. CORBA ORB:
   - Unix: ORBacus 4.0.1 or higher
   - Windows NT: ORBacus 4.0.1 or higher
5. Tcl interpreter (necessary to interpret configuration file):
   - Unix: Tcl-8.3.x or higher.
   - Windows NT: ActiveState ActiveTcl 8.3.4.2 or higher

2.3 Installation Order Under UNIX

1. Make sure that required software is installed and works.
2. Unzip pimr-1.0.0.tar.gz or pimr-1.0.0.zip in the directory you chose (further it'll be called "project root directory" , and we'll mar it as `<project_root>`).
3. Change directory to `<project_root>`.
4. Run configure with `./configure` with necessary options (you can get options list by command `./configure` with `--help` option; in particular, you can use `--prefix=<smth>` option to set installation directory)
5. Run gmake
6. Change to user root with `su`
7. Run installation with `gmake install`
8. To uninstall package use gmake uninstall

2.4 Installation Order Under Windows NT

1. Make sure that required software is installed and works. To work with MSVC you need:
   (a) INCLUDE and LIB user's environment variables to be set and to contain paths to MSVC include files and libraries accordingly.
   (b) to include paths to **nmake**, **cl**, **link** utilities into `PATH` user's environment variable.
2. Unarchive pimr-1.0.0.tar.gz or pimr-1.0.0.zip in the directory you chose (we'll call it `<project_root>`). Edit **environment.nt** file in the subdirectory `<project_root>\config\Win32`. Set values of the next nmake-varaiables:
   - PROJECT_DIR is `<project_root>`.
   - INSTALL_DIR (installation directory) - executable files will be placed in **bin**subdirectory, header files in **include** subdirectory, idl-descriptions in **idl** subdirectory, configuration files in **conf** subdirectory of the installation directory.
   - GRADCTOOLBOX_INSTALL_DIR is the directory to which GradC++Toolbox is installed. It's necessary to compile the component and PIMR with the same ORB.
   - ORB (conventional ORB name) - this variable should have one of the following values: ORBacus, OmniORB or TAO depending which ORB you have. Depending on this variable you should configure
     - ACE_ROOT - ACE root directory (in case of using TAO)
     - TAO_ROOT - TAO root directory (in case of using TAO and TAO is out of ACE directory tree)
     - OMNI_ROOT - omniORB root directory (in case of using omniORB)
     - OOC_ROOT - ORBacus root directory (in case of using ORBacus)
   - TCL_DIR - directory in which TCL is installed.
3. To compile the package change directory to `<project_root>` and run `make` (`make build`).
4. To install the package change directory to `<project_root>` and run `make install`.
5. To remove installed files change directory to `<project_root>` and run `make uninstall`.
6. To remove files created after compilation change directory to `<project_root>` and run `make clean`.

3 Setup

Before use PIMR you need to set it up by editing the configuration file and using different command line options. Information about command line argument you can get running PIMR with `--help` option. Also you can use next options:

- `--help`

  Represents embedded information about command line options.

- `--config file_name`

  Allows to load command line arguments from file.

- `--with-naming initial_object`

  Sets up initial objects.

- `--ior-stdout`

  Object Reference to repository will be printed out to standard output .

- `--ior-file-PIMR file_name`

  If you use this option repository will try to print IOR-reference to repository service into the file.

And the repository options:

- `--configfile configuration_file` Makes repository to use this configuration file. If there is no such directive the repository will try to use `pimr.cfg` file from current directory. Don't mix up this option with `--config` option.

- `--logfile log_file`

  Makes repository to write reports to this file. If there is no such directive the repository will try to use `pimr.log` file from current directory. You can find more about log-file in the "Usage" part.

You can also specify options of the object broker you are using.

For instance, if to run repository, that uses ORBACUS , with following options :

```
pimr.exe
   --configfile conf.pimr --logfile log.pimr
   -OAport 12345 -ORBtrace_connections 2
   --ior-stdout
```

it'll read repository configuration from "conf.pimr" file; we'll put log report to "log.pimr" file; repository ORB will use 12345 port and will print out test messages; at startup repository will print out it's IOR.

Repository configuration file is the program in TCL language. To set up PIMR several commands were added to the language. They are:

- `service {service name}`
  `--ior {ior reference}`
  `--repositoryid {idl interface}`
  `[--restart {restart command}]`
  `[--callback {ior reference}]`

  Informs repository about service and its parameters. It's necessary for repository to be able to redirect requests to service and to control its state. `--restart` option shows that if service is down it should be restarted, this argument is optional.

- `forwardfreq {number}`

  Makes repository to initiate check of services state when redirecting not rarer than every N seconds.

- `checkfreq {number}`

  Makes repository to check if services are available not rarer than every N seconds.

- `redirecttimeout {number}`

  Sets the timeout in seconds for request redirection.

- `obsoletetimeout {number}`

  Makes repository to check if service is available in N seconds after the last check.

- `restarttimeout {number}`

  Sets the timeout in seconds for service restart. If timeout is exceeded while restarting service, the service Is marked out as failed and will not be restarted any more.

Let's consider, for instance, following configuration file:

```
#
# This is TCL demo config file for PIMR
#
set thisHost "127.0.0.1"
set thisPort 11000

service  HelloService \
    --ior "corbaloc::${thisHost}:${thisPort}/HelloService" \
    --repositoryid "IDL:Demo/HelloWorlder:1.0" \

#    --restart "startHelloWorlder.bat" \  This lines are commented
#    --host "${thisHost}:${thisPort}" \   Note that ${thisHost}
#    --callback "" \                      will be computed by TCL.
```

```
forwardfreq 1
checkfreq 10
obsoletetimeout 15
restarttimeout 30
redirecttimeout 60
```

This configuration file gives information about one service named HelloService. Repository will seek for it on local host on the port 11000 by name HelloService. Repository will not restart the service, for there is no 'restart' command.

It is advisable to choose repository parameters observing the next rule: obsoletetimeout + checkfreq + restarttimeout < redirecttimeout

4 Testing

4.1 General Information

To check working capacity of the compiled package you can use test samples. You can compile test examples under Windows running following command from `<project_root>` directory:

`nmake /f makefile.nt buildtests`

To uninstall tests run:

`nmake /f makefile.nt cleantests`

Under Unix-like system use ???.

Now change directory to `<project_root>/demo/` and view "readme"-files in this directory and in subdirectories.

Test CORBA-services are in `<project_root>/demo/server*/` subdirectories. To run tests you can use verb"run.bat" files. Tests interfaces you'll find in `<project_root>/demo/*.idl` files.

`<project_root>/demo/client/` directory contains client program for these services. Also there you'll find `run_straight.bat` and `run_via_repository.bat` to start client directly and via repository accordingly.

`<project_root>/demo/repository/` directory contains repository configuration files for demo-samples and `repository*.bat` start scripts.

Scripts start service and repository on ip-address 127.0.0.1 and ports 11000 (11001), 11111 accordingly. So, before starting the programs make sure given ports are available.

4.2 Testing

- Compile repository and test samples.
- Check client and service

  Start client. It should terminate with the error saying that service is unavailable.

Start HelloWorld running `run.bat` or `"run.sh"` in `<project_root>/demo/server0/` directory.

Then run client program with `run_straight.bat` (Don't terminate HelloWorld) from `<project_root>/demo/client/` directory. Client should run without errors and "Hello, world" should be printed out in the service window .

- Test 1 (tests if the repository is capable to redirect a request.)
  - Star repository with `repository.bat` but don't start the service. Make sure that repository considers HelloWorld service as terminated .
  - Start client with `run_via_repository.bat`. Client should get `OBJECT_NOT_EXIST` exception.
  - Start HelloWorld with `run.bat` from `<project_root>/demo/server0/` directory.
  - Wait (not longer than one minute) while repository will detect that service is running.
  - Start client with `run_via_repository.bat`. Client should run without errors and "Hello, world" should be printed out in the service window.
  - Terminate the service and immediately start client. Client should get `OBJECT_NOT_EXIST` exception..
  - Stop repository.

- Test 2 (test if the repository is capable to restart a service)
  - Start repository with `repository1.bat` but don't start the service. Wait (not longer than one minute) - repository should start the service using `restart.bat` script.
  - Start client with `run_via_repository.bat`. Client should run without errors and "Hello, world" should be printed out in the service window.
  - Terminate the service and immediately start client. Not longer than in one minute repository should start the service and client should perform the request without errors.
  - Terminate repository.

- Test 3 (test if the repository is capable to accept "activated" message)
  - Start HelloWorld service from `<project_root>/demo/server0/` directory using `run.bat`.

- Start client with `run_straight.bat`. Client should run without errors and "Hello, world" should be printed out in the service window.
- Start repository using `repository.bat`.
- Wait (not more than one minute) while repository will detect the service.
- Stop the service and start it from `<project_root>/demo/server1/` directory.
- Wait (not longer than one minute) while repository will detect the service again.
- Start client using `run_via_repository.bat` . Client should run without errors and "Hello, world" should be printed out in the service window.
- Start client with `run_straight.bat`. Client should run with error saying that service is unavailable (for it runs on the other port.).
- Stop repository.

•

5 Usage

To make repository control a service you should register it at repository. You can do it using the configuration file or using the RepositoryAdmin interface . Then client will be available by the following reference:

`corbaloc::repository_host:repository_port/required_service_name`

Client sends request using this corbaloc. Repository automatically redirects request to object-service.

Repository is available by next reference:

`corbaloc:: repository_host:repository_port/Repository`

Attention! Don't use "Repository" name when registering your services. This name is used to access the repository. If you try to register service with such a name you'll get PIMR::RepositoryAdmin::NotAviable exception.

When running repository records messages into log-file. By default it's `./pimr.log` , but you can change the location of the file using `--configfile log-file` command line argument . To choose type of messages you want to record to log-file set the following variables in дМС PIMRPostConfig.h:

```
#define LOG_DEBUG_ENABLE   false
#define LOG_INFO_ENABLE    true
#define LOG_WARNING_ENABLE true
#define LOG_ERROR_ENABLE   true
#define LOG_FATAL_ENABLE   true
```

and recompile repository.

6 Document History

20.09.2002 - Initial version.