

CONTRACT NUMBER 508830

DEISA
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Second release of DCRMP: update of the DCRMP software
stack accordingly to the new software releases

Deliverable ID: DEISA-DSA3-5

Due date : April, 29, 2006

Actual delivery date: May 17, 2006

Lead contractor for this deliverable: IDRIS-CNRS, France

Project start date : May 1st, 2004

Duration: 4 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Content

Table of Content.....	2
List of Figures.....	2
List of Tables.....	2
List of Examples.....	2
1. Introduction.....	3
1.1 Executive Summary.....	3
1.2 References and Applicable Documents.....	3
1.3 Document Amendment Procedure.....	3
1.4 List of Acronyms and Abbreviations.....	3
2. Second release of the DCRMP.....	4
2.1 New Requirements.....	4
2.2 Requirements analysis.....	5
2.3 Implementation decision analysis.....	5
3. DPC documentation.....	7
3.1 Structure of a DPC Package.....	7
3.2 Pre/post/install scripts.....	7
3.3 <code>config</code> file syntax.....	8
3.4 How to create a DPC package.....	8
3.5 DPC installation process.....	9
3.6 Current status and conclusions.....	9
4. Conclusion and future direction of the DCRMP task.....	11
4.1 Future directions.....	11

List of Figures

Figure 1 - DCRMP package.....	6
Figure 2 - Package structure.....	7

List of Tables

Table 1 - List of acronyms.....	3
Table 2 - DCRMP implicit variables.....	8

List of Examples

Example 1- Example of configuration file.....	8
---	---

1. Introduction

1.1 Executive Summary

This document presents the latest developments of the DCRMP task. Its scope has been expanded to support the distribution of applications that are part of DEISA’s Common Programming Environment (CPE). User Support (SA4) has formulated the requirements that led to the development of the DCRMP Package Creator (DPC), an extension to the DCRMP software. DCP is capable to distribute a public domain software application as an executable file. This document presents the rationale that led to the current release of DCP.

It analyses the requirements expressed by SA 4 and the implementation options. A detailed documentation of DPC is provided. Finally, an outline of the future development of the DCRMP task is given.

1.2 References and Applicable Documents

- [1] <http://www.deisa.org>
- [2] DEISA Primer, <http://www.deisa.org/userscorner/primer/primer.php>
- [3] DEISA D-SA3-3 “First release of DCRMP”

1.3 Document Amendment Procedure

The initial document amendment procedure is via communication between members of the SA 3 team. The document is then submitted for review to the DEISA Executive Committee and an Executive Committee appointed DEISA reviewer. The document is then amended according to the comments received from the Executive Committee and the appointed reviewer. It is subsequently re-submitted to the DEISA Executive Committee for submission to the EU.

1.4 List of Acronyms and Abbreviations

Term/Acronym	Definition
CPE	Common Production Environment
DCRMP	DEISA Cluster Resource Management Package
DPC	DCRMP Package Creator
RPM	Red Hat Package Manager
SA 3	DEISA Service Activity n.3: Resource Management and Middleware
SA 4	DEISA Service Activity n.4: User Support

Table 1 - List of acronyms

2. Second release of the DCRMP

SA 4 is in charge of the deployment of the DEISA CPE. To carry out this task the help of an application able to distribute CPE software components could be really helpful. Moreover it could facilitate the adoption of the DEISA software stack from new partners. For these reasons SA3, exploiting its expertise and reusing some of the software already developed, worked in close collaboration with SA4 members to understand their requirements and to provide a suitable instrument for the distribution of the CPE.

DCRMP version 2 has been designed with the following objectives:

- distribution of public domain software only;
- distributing the software only among homogeneous super-clusters in binary form, when there are at least two platforms of the same architecture with an analogous CPE;
- SA 4 has interest in distributing both binary distributions and source code distributions. Source distribution will be considered in the next phase once more experience with the DCRMP framework for CPE has been gained. Source code distributions will also be considered for those sites with administrative and security local rules which forbid them to install binary files.

In the next paragraph, the requirements expressed by the SA 4 are listed followed by an analysis of the requirements. To close the chapter, the implementation decisional analysis is presented.

2.1 *New Requirements*

The CPE is composed by several software tools like, shells, compilers, libraries, etc. [2]. Below is a complete list of the current AIX CPE:

- Bash
- Tcsh
- C/C++
- Fortran
- Java
- ESSL (AIX)
- FFTW
- HDF5
- LAPACK
- MASS (AIX)
- NAG
- netCDF
- PESSL (AIX)
- ScaLAPACK
- WSMP
- Emacs
- GMake
- NEdit
- omniORB
- OpenSSH
- Perl
- Python
- Tcl/Tk
- TotalView

- CPMD
- CPMD2cube
- gOpenMol
- Torb
- WIEN2k

Essentially, SA 4 needs a standard way to distribute and install the public domain software. In fact, for each application of the CPE they specify not only the release number, but also the parameters, and the functionalities that should be invoked at compile time¹, to ensure a higher level of coherence of the software environment in the DEISA supercomputing infrastructure.

Currently, the distribution of the installation procedure is carried out by means of a written document, where a detailed description of the required steps is reported. The most obvious solution would be the translation of this procedure into shell scripts. Additionally to further simplify the installation of new packages it would be interesting to distribute these scripts along with all the other data as a self-contained file.

Finally, it should be considered that usually the installation of public domain CPE software components is performed in user mode because User Support operators does not always have root access to the system.

2.2 Requirements analysis

Firstly the characteristics of CPE software differ substantially from the ones of the software distributed by SA 3. In fact, software used by SA 3 is intrinsically distributed, i.e. it is composed by different components that usually run on different machines, while CPE software components can be mainly classified as high level executables or scientific libraries.

For this reason the previous version of the DCRMP [3] would not fully satisfy the new requirements. In fact, it was developed specifically for the installation of multi-component distributed applications, where the main concern is on maintaining the configuration coherence among the different hosts. It follows that the development of a new tool will meet the requirements more appropriately, although several ideas matured with the previous release and some parts of the source code have been reused and extended.

Concluding, the DPC consists in an extension of the previous DCRMP release developed with the strict collaboration of SA 4 representatives, able to create installation packages. Considering the substantial improvement of this new release and the unique characteristic of this new tool, it was decided to name it DCRMP Package Creator (DPC), also to underline its stand alone characteristic.

2.3 Implementation decision analysis

The use of a standard Package Manager to satisfy the requirements seemed straightforward. Unfortunately the AIX [3] standard distribution package (LPP format) was dismissed in a first instance because it was not able to maintain multiple versions of an application, and moreover it didn't allow the installation of a package without root permission. The same consideration can also be applied for the RPM, while the other Package Managers do not run on AIX operation system.

For these reasons it has been decided to extend the packaging mechanism already used by the first release of the DCRMP, which uses an executable '.bin' package to

¹ For example, support 64-bit mode when this is not the default one on an AIX system, or the support of non default or extended features (for instance, both single and double precision modes for the FFTW library, when only double one is generated by the default installation procedure).

distribute public domain software components. This type of package is composed of two parts:

- Head: a shell script in charge of extracting the data contained in the tail and of performing other preliminary operations
- Tail: all the data necessary for the package installation, wrapped up into an archive.

Figure 1 shows a graphical representation of a '.bin' package.



Figure 1 - DCRMP package

Obviously this mechanism has been improved in several ways in order to create a standalone instrument that can also be used without the other DCRMP software components. In particular the head script has been extended with several new features:

- It creates a temporary directory where it expand all the data and deletes it after the end of the installation process;
- It creates several environment variables that can be used during the installation process;
- It executes a predefined set of scripts as explained in the next section;
- It prompts the user for parameters that can be used to customize the installation (like installation directory, or other configuration parameters) or it can read them from a file.

Moreover, to simplify the creation of DPC packages, a new command has been developed. This command takes as input a set of files and directories and produces a '.bin' package. The current release of the DPC does not maintain any historical data about packages already installed on a particular host.

3. DPC documentation

In this section the documentation to install or to create a package with this tool is provided. Firstly the structure of a package is described, followed by the explanation of the pre/post/install scripts. In the paragraph 3.3 the syntax of the main configuration file is defined. The last two paragraphs report the steps necessary to create a package with the DPC, and to install a package created with the DPC.

3.1 Structure of a DPC Package

Packages created by the DPC use a fixed directory structure for their data tail (see paragraph 2.3). In the figure below there is a representation of this structure that is crucial to understand the DPC.

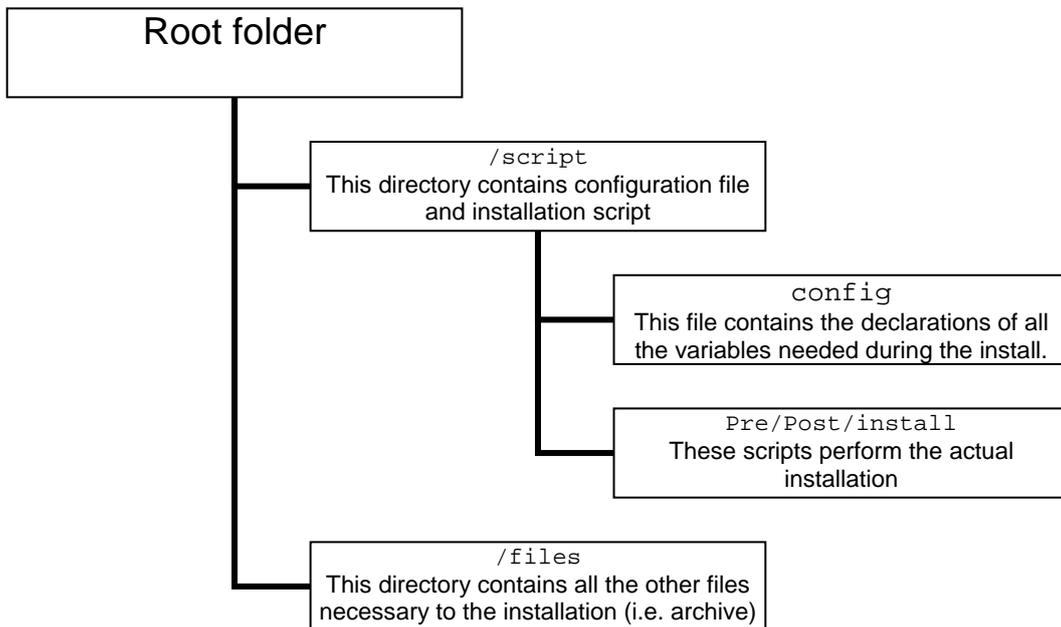


Figure 2 - Package structure

3.2 Pre/post/install scripts

The pre/post/install scripts are in charge of performing the actual installation. It is up to the package creator to create them appropriately. Before executing these scripts the DPC creates two groups of environment variables that can be used to carry out the installation:

1. Implicit variables: they are all named with a prefix DCRMPI_ and they can be used to retrieve information about the temporary directory, and the location of the different scripts during the installation process. The following table gives an overview on the implicit variables.
2. User variables: these variables contain all the configuration parameters necessary to perform the installation. They are defined in the configuration file and they hold the valued inserted by the user during the installation process.

DCRMPI_TEMP_DIR	This variable refers to the temporary directory created at the beginning of the execution. During the installation process the tail of the package is copied into this directory.
DCRMPI_SCRIPTS_DIR= "\$DCRMPI_TEMP_DIR/scripts"	This variable holds the path to the scripts directory.
DCRMPI_CONFIG_FILE= "\$DCRMPI_SCRIPTS_DIR/config"	This variable points to the configuration file.
DCRMPI_INSTALL_FILE= "\$DCRMPI_SCRIPTS_DIR/install" DCRMPI_PREINSTALL_FILE= "\$DCRMPI_SCRIPTS_DIR/pre-install" DCRMPI_POSTINSTALL_FILE= "\$DCRMPI_SCRIPTS_DIR/post-install"	These three variables point to the three scripts that are called during the installation process.
DCRMPI_FILES_DIR= "\$DCRMPI_TEMP_DIR/files"	This variable refers to the directory that contains all the files necessary for the installation process.

Table 2 - DCRMP implicit variables

3.3 config file syntax

The configuration file is used to enumerate all the variables that are necessary during the installation process of a public domain software package. The syntax of the file is based on the classic format

```
<variable name>=<default value>
```

It is important not to leave blank spaces after the variable name; additionally, to avoid collision with other existing variables, the prefix DCRMP_ should be adopted for all the names. If there is not a default value for the variable it is possible to leave the right value blank, (e.g. <variable name>=).

To display a short explanation during the installation process it is also possible to insert a brief description of the variable in question using the following syntax:

```
<variable name>.DESCRIPTION=<brief description>
```

For example, to prompt the user for the installation directory the following two lines should be written in the configuration file:

```
DCRMP_INSTALLATION_DIR=/tmp/foodir/install
DCRMP_INSTALLATION_DIR.DESCRPTION=The path where the
application will be installed
```

Example 1- Example of configuration file

3.4 How to create a DPC package

In order to create a package with the DPC:

1. It is necessary to set up a directory structure as described in the previous paragraphs.
2. It is required to write a proper configuration file that contains all the variables (see 3.3) needed during the installation process.
3. At least one of the pre/post/install scripts must be created.

4. Finally all this is packed together and by creating a package with the DPC using the `create-package.sh` shell script (it is in the root directory of the DPC distribution).

The command uses the following syntax:

```
create-package.sh {base_directory|-h} [-n file_name]
```

where:

```
base_directory is the path to the directory structure created during the step 1.  
-n is an optional flag that can be use to specify the name of the output package.  
-h print an help message
```

3.5 DPC installation process

To perform the installation of a binary package created with the DPC it is sufficient to execute it. The package can be invoked with the following arguments:

```
<name of the package> [-f parameters_file]
```

When installed, a DPC package can be executed in two different modes. The first is called *interactive mode* and it prompts the user for every needed installation variable (this is the default and does not need any parameters). While the *batch mode* (that can be activated with the flag `-f`) will load all installation variables from an input file.

A typical installation of a DPC package performs the following steps:

1. It creates a temporary directory and expands all data in the tail of the package into this directory.
2. It reads a configuration file, where all the installation variables are declared (see paragraph 3.3).
3. If the package is executed in the *interactive mode* it prompts the user for the values of the variables as defined in the configuration file. While if it is executed in batch mode, it will automatically load the variables value from the specified parameters file.
4. It exports the variables defined in the previous step as environment variables, then it also adds some other implicit variables (see 3.2).
5. It executes the pre-install, install, and post-install scripts. These scripts must contain the commands necessary to accomplish the installation.
6. It deletes the temporary directory created in step 1 and terminates.

When the package is executed in the *interactive mode* it creates a configuration file that contains all the input entered by the user during the installation process (step 3.2). This file is created in the current working directory and is called `parameters.conf`. In this way, if a user wants to install the same package again but needs to change only one variable, he/she can modify this file and execute the package in *batch mode* again. Hence the user will not have to type-in all the parameters again.

3.6 Current status and conclusions

In the last month three versions of the DPC have been released where several bugs discovered with the help of SA 4 were corrected . The current release is 0.3, and is

distributed as a DPC package. As an example, Python 2.4.1 64bit has been successfully packed and tested on the CINECA SP5 cluster and IDRIS SP4 cluster.

4. Conclusion and future direction of the DCRMP task

The current DPC release has reached a maturity level sufficient for a production environment. Obviously it does not mean that the addition of new features and the maintenance will be stopped. But rather it is time to foster the adoption of this new tool through the distribution of CPE public domain applications by means of the DPC. There have been several discussions on either distributing application with the DPC as source code or as compiled binaries. The exemplary Python version distributed as a binary executable has proved to be reliable enough, in fact the same package has been installed and tested successfully both on Power4 and on Power5 systems (IDRIS an CINECA). Considering such experience, this solution could provide simpler package (therefore with less bugs and faster to be prepared) and it should be considered to use this approach in all cases where possible.

SA4 will care about the packaging and will decide when and which components of the CPE will be distributed through the DPC.

4.1 Future directions

Improvements to the DPC are already in progress with the help of SA 4 representatives (mainly patching), although other work could be done to simplify the creation of new packages. For example, in the DPC distribution more complex pre/post/install scripts should be included with specific examples on a package installation. In this way a user can start up building a package directly modifying these scripts. Additionally, it would be useful to have a configuration file for the `create-package.sh` command where to put all the parameters that are currently passed with command line flags.