OSCAR Package Compiler Manual

COLLABORATORS

	TITLE : OSCAR Package Compiler Manual	*REFERENCE* :

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Geoffroy Vallee and Jean Parpaillon	June 22, 2007	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.2	06-22-07	Documentation for OPKG 0.2	

Contents

1 Introduction

The OSCAR Package Compiler (OPKGC) aims to generate binary packages (both RPMs and Debian packages) from OSCAR packages in order to ease their management and diffusion.

The idea of an OSCAR Package (OPKG) compiler has been defined and presented by Erich Focht during the OSCAR meeting in January 2007.

Few constraints have to be kept in mind since the OSCAR developers started to implement OPKGC:

- The implementation has to be simple in order to ease the work of the maintainers: the development team of the OSCAR project is quite small so smaller and simpler is the code, better it is.
- OSCAR core components are implemented in Perl or Python, therefore it is a real plus if OPKGC can do so.

The OPKGC works in two phases: (i) the generation of the *meta-files* for the creation of binary packages (e.g. the creation of a spec file for RPMs), and the creation of binary packages based on these meta-files.

2 Architecture

2.1 Generation of Meta-Files for the Creation of Binary Packages

2.1.1 Introduction

OSCAR packages are described by a set of files in a distribution independant way. These files contains mandatory informations for binary packages generation plus some optional informations common to distribution specific packages system. `config.xml` file contains most of informations about OPKG. For instance, it provides dependences informations, package information (such as version, author's information) and changelog. The other files are scripts executed before or after package installation or removal, plus documenation and testing scripts. A complete description of an OPKG is available on the OSCAR developpers wiki[1].

The idea is therefore to use these files to generate meta-files for binary packages. Typically, OPKGC allows developers to generate meta-files for binary packages for different distribution. For instance, they may want to execute the following command for the generation of binary package meta-files for RHEL: **opkgc --dist="rhel" <directory>**. This command will parse files in the given directory of the OPKG and generate the appropriate binary package meta-files.

2.1.2 Implementation

The core of the mechanism for the generation of binary package meta-files is divided into three parts: (i) the parsing of the `config.xml`, (ii) the transformation of this XML files to the binary package meta-files and (iii) the addition of the various scripts and documentation files into the binary packages.

To parse `config.xml` file, which is an XML document, OPKG use a standard API available in the Python language, **etree**. That will ease the OPKGC maintenance and will avoid bugs implied by the implementation of a new parser.

A naive approach for the transformation of *config.xml* into binary package meta-files is to use XSLT files. XSLT stylesheets are dedicated to XML files transformations and a language (XPATH) allows access to elements in the XML document in a concise form. The main drawback is that generated files are not XML files and XSLT does not allow to control fine enough the syntax of generated files (ie, regarding spaces, carriage returns, or other regex transformations).

Therefore, the *Cheetah*[2] template system is used for meta-files generations. It needs more code to be written, but, this is the only way to generate syntaxically correct files.

Therefore OPKGC is composed of two parts: (i) Cheetah templates for generated files meta-files, and (ii) a python code that drives the templating system, creates the right meta-files tree and generate the packages.

[1]

[2]Included into all Python distributions:

2.2 opkgc Installation

2.3 Creation Binary Packages Configuration Files for Specific Linux Distributions

OSCAR package configuration files (*config.xml*) may integrate filter for the specification of parameters for different Linux distribution. For instance, it is possible to specify that the OPKG lam-oscar depends on Red Hat Entreprise Linux on the *lam-mpi* RPM, whereas on Mandriva, it depends on the *lam* RPM. The OPKGC should then allows developers to specify the Linux distribution for which they want to generate binary packages (for instance using a command such as *opkgc -rpm="rhel" config.xml*). The specification of such dependences in the confiugration files disable the capability to have one single generic XSL template for the generation of binary packages.

Typically the way to deal with such a solution is to set a parameter when compiling an OPKC; paramater that gives the target Linux distribution. The standard mechanism for the implementation of such a solution is to combine XSLT stylesheets. A first stylesheet give a generic template and a second template define a constant (a parameter in the XSLT terminology) defining the target Linux distribution. Therefore when combining the two stylesheets, the final stylesheets includes both the generic template and a variable that sets the target Linux distribution. Therefore, only the stylesheet defining the variable for the specification of the target Linux distribution must be dynamically generated based on the OPKGC parameters (simple task).