

Design of pyCCABS

"An Automatic build system for CCA projects"

Torsten Wilde and James Kohl
Oak Ridge National Laboratory

CCA Forum Quarterly Meeting
Atlanta, GA ~ January 2005



What's this all about?

- Boyana's CCA GUI should be able to actually build projects ☺
- Generate build system structure using custom project information
- Requirements:
 - Easy to use
 - Not hard-coded for one specific build system (e.g. GNU Autotools)
 - Users should be able to extend/customize build system (e.g. configure macros)
 - Advanced users should be able to modify generated build system files if needed (user-readable output)
 - Modifications are backed-up but not transferred if file gets re-written

- Abstracting build system information
 - Work with any build system
 - Highly subjective user choice
- Automating generation of the build system is a trade-off between standardization & user flexibility
 - Define rules for project creation, management, build and installation
 - Define rules for special project types (take advantage of known additional information, e.g., CCA)
- Modular program design for easy extensibility



- Project generation tool written in python
 - Command line interface (GUI independent)
- Provides plug-in interfaces for build systems and special project types
 - Works with “any” build system
 - Takes advantage of known information for specific target development (e.g. CCA ports, components, applications ...)
 - Define standard skeleton project directory structure
- Current state:
 - Plug-ins for CCA and GNU Autotools (maybe all we ever need ☺)
 - Autotool files are generated for individual leaf directories
- Future work:
 - Generate build system files for complete tree structure
 - Run autotools from pyCCABS
 - Have pyCCABS invoke make



□ Directory + Files

□ Directory Types:

□ Package/Subpackage

- Can be installed independently
- Subpackages are configured by their parent package
- Package is the project root

□ SubDirectory

- No configure
- Used to make ONE library or executable
- Automatically included in parent -I, -L

□ File Types:

□ Source file

□ Header file

□ Script file

□ “Normal” file (e.g. README, jpg)

□ File Attributes:

□ Install

□ Distribute

□ Used for building library

□ Used for building executable



Package/Subpackage options

P
Y

C
C
A
B
S

© Torsten
Wilde

- `pyCCABS <options>`
- `--pinit=PROJECT_NAME`
 - Creates initial project root directory PROJECT_NAME and copies necessary (directory type: Package)
- `--addsubpackage=PACKAGE_NAME`
 - Creates subdirectory PACKAGE_NAME in current dir and copies necessary files into it (directory type: SubPackage)
- `--deletesubpackage= PACKAGE_NAME`
 - Removes subpackage from build and deletes PACKAGE_NAME directory
- `--disablesubpackage= PACKAGE_NAME`
 - Removes subpackage PACKAGE_NAME from build, leaves directory intact
- `--enablesubpackage= PACKAGE_NAME`
 - Adds subpackage PACKAGE_NAME back into build (for existing subpackage directory)
- `--listsubpackage`
 - Lists all subpackages for the current package



- pyCCABS <options>
- “--addfile=File_Name:[type:...]”
 - Adds file File_Name to the current directory structure (doesn't create file)
 - [type] options are:
 - source:header:script:file:exe:lib:inst:dist (in valid combinations)
- “--addfiles=File_Name:[type:...]+FILE_NAME:[type:...]”
 - Add files File_Name + File_Name to the current directory
- “--deletefile= File_Name”
 - Removes file from build and deletes it (need disable and enable option?)
- “--listfiles”
 - Lists all files for the current directory

- The “**“README_ TEMPLATE”** file
 - provides step-by-step description of using pyCCABS
 - Copied into evry new package directory

- The “**“template-config.tempin”** file
 - This file is the user input template configuration file
 - Lists all user editable system variables
 - E.g. **XXX_VARIABLE_NAME_XXX = value**
 - **XXX_MY_NAME_XXX = TestProject**



- Two parts:
 - How to use pyCCABS
 - Behind the scenes

Example: CCA application

- Task: Create a CCA application that defines ports, components and a driver
- Requirements:
 - pyCCABS (with CCA support installed – by default)
 - For GNU Autotools (GAT) support:
 - GNU Automake >1.7
 - GNU Autoconf > 2.57

pyCCABS use without GUI

- Use: pyCCABS --help to see all available options
- 1. pyCCABS --pinit=MY_PROJECT_NAME --ptype=cca --pbs=gat
 - Creates dir MY_PROJECT_NAME and default dir structure:
 - Applications (subpackage)
 - Components (subpackage)
 - Ports (subpackage)
 - Xml-repository (subdir)
- 2. Use pyCCABS to add files, subdirs or subpackages to project
 - pyCCABS --addccaport=testPort
 - pyCCABS --addfile=portsource.cxx:source:inst:lib
- 3. edit template-config.tempin for each package/subpackage
- 4. Use pyCCABS --pcreate to generate build files
 - README.in, configure.ac, Makfile.am ...
- 5. Use pyCCABS --pmake to make project
 - Configure, Makefile, README ...
 - Runs selected build system native make
 - Make options: --pmakeopt=[install, uninstall, rpm ...]



- PROJECT_NAME

- Ports/

- testPort/

- portsource.cxx, template-config.tempin, README.in, configure.ac, Makfile.am, README, configure, Makfile, portsource-version.exe, portsource.o

- Config/

- Autotool files (e.g. install.sh)

- template-config.tempin, README.in, configure.ac, Makefile.am, README, configure, Makefile

- Config/

- Autotool files (e.g. install.sh)

- Components/

- template-config.tempin, README.in, configure.ac, Makefile.am, README, configure, Makefile

- Config/

- Autotool files (e.g. install.sh)

- Applications/
 - template-config.tempin, README.in, configure.ac, Makefile.am, README, configure, Makefile
 - Config/
 - Autotool files (e.g. install.sh)
- Xml-repository/
 - template-config.tempin, README.in, configure.ac, Makefile.am, README, configure, Makefile
 - Config/
 - Autotool files (e.g. install.sh)
- AND make targets:
 - Install, uninstall, clean, distclean, rpm (generates source rpm and tar distribution)

- Boyana's GUI hides pyCCABS options from the user
- GUI dialogs need to collect information required by pyCCABS from the user
- User doesn't need to know anything about pyCCABS ☺ !

Template-config.tempin in detail

```
□ <MAIN>  
# project/package name (also autoconf package name)  
XXX_MY_NAME_XXX = (default PACKAGE_NAME)  
  
# package version  
XXX_VERSION_XXX = 0.1  
  
# contact e-mail  
XXX_CONTACT_EMAIL_XXX = wilde@ornl.gov  
  
# additional name tag for distribution  
XXX_RELEASE_INFO_XXX = (default A)  
  
# additional name tag for libraries  
XXX_LIB_INFO_XXX =  
  
#additional name tag for executable  
XXX_EXEC_INFO_XXX =
```





```
# RPM info
XXX_RPM_SUMMARY_XXX = "bla bla."
XXX_RPM_DESCRIPTION_XXX = "bla bla"
XXX_PACKAGELICENSE_XXX = LGPL
XXX_RPM_GROUP_XXX = Development
XXX_RPM_REQUIRES_XXX =

# project/package wide definitions
# additional linker flags
XXX_ADD_LINK_FLAGS_XXX =
# additional compiler flags
XXX_ADD_COMPILER_FLAGS_XXX =
# additional needed user library directories & user libraries
XXX_ADD_LIB_DIRS_XXX =
XXX_ADD_LIBS_XXX =
# additional needed user header directories
XXX_ADD_HEADER_DIRS_XXX =

# additional autoconf macros (list separated by spaces)
XXX_ADD_AUTOCONF_MACROS_XXX =

</MAIN>
```



- Two parts:
 - How to use pyCCABS
 - Behind the scenes

System File

```
<SYSTEM>
build_system = gat
make_install = 1
make_rpm = 1
parent_dir = ROOT
project_root = /home/wilde/py-builder-demo/Helloworld
project_type =
tree_changed = 1
dir_type = package
project_type_dir =

    # list of subdirectories (subpackages)
    # subpackage_name = 0/1 (not active/active)
    <SUBPACKAGES>
    </SUBPACKAGES>

    # list of subdirectories (sources)
    # subdir_name = 0/1 (not active/active)
    <SUBDIRS>
    </SUBDIRS>

    # list of files in dir
    # file_name = install sourcedist
    <DIRFILELIST>
    helloworld.cxx = source:inst:exe
    </DIRFILELIST>

    # project wide build information
    <BUILD>
    project_cpp_flags =
    </BUILD>
</SYSTEM>
```



How project type CCA was added

P
Y

C
C
A
B
S

- Wrote plug-in cmd_type_cca.py
 - Adds command line options: ccaport, ccacomponent, ccaapplication
 - Adds project type: cca to pyCCABS
 - Project type handler will refer calls on project type to handler for type found in system-template.tempin file
- Added directory structure to package definition storage dir
 - PackageTemp/
 - Gat/ (GNU Autotools build system support files)
 - Cca/ (project type cca support files)
 - Gat/ (global CCA GAT definitions)
 - Ccaport/ (specific CCA port files)
 - Gat/ (specific CCA port GAT extensions)
 - Template-config.tempin & template-system.tempin (extensions), template-config.files, template-config.files.copy
 - Template-config.tempin & template-system.tempin (extensions), template-config.files, template-config.files.copy
 - Template-config.tempin & template-system.tempin (extensions), template-config.files, template-config.files.copy

Future Features Discussion

- Save and load xml description of project tree
 - Create exact copy of project on different machine
- Move definition of directory build (exe, libso, liba) into system file
 - Creation and assembly of make file list automatically using extension definition file (e.g. headers = hh, HH, h)
 - Automatic generation of header file list
- Add user *.in files automatically to configure or add pyCCABS command line option for doing so (add, delete)
- Add priorities for packages build process in one project (use as deployment tool?)
- How to integrate build system specific features
 - GAT – autoconf macros
 - Where should CCA macros live?
 - How will pyCCABS know where to look?
 - Which macros should pyCCABS provide?
- How to do make after change in project?
 - Change flag set, go to parent package, reconfigure that package



- Probably more to do than apparent
 - Enable input of autoconf macro directories
 - Cleanup standard compile/link flags
 - Streamline and improve documentation
 - Get default values from CCAFE and Babel
- Integrate autogen.sh script into pyCCABS (run autotools)
- Implement pyCCABS –pmake
- Interfacing with Boyana's GUI
- Demo will now follow

