# Python scripts to create VS2005 files

## Extending windows builds to use VS2005

03/11/2007 08:09 -0800

### The Current system

The elements of our current system can be found in the following places

1. CMT: code implementing  the cmt build vsnet command to generate the vsnet project files for the libraries and applications, and a .sln file to define a workspace. These files are written to a folder "Visual"
2. GlastPolicy/GlastCppPolicy/cmt/fragments has 13 files used to support this: the locations are provided by a set of make_fragment statements in GlastPolicy/GlastCppPolicy/cmt/requirements
3. MRvcmt ruby code that rewrites the .sln files, which are not correct for VS 2003 or 2005.
4. The script GlastPolicy/GlastCppPolicy/cmt/glast_make_shlib.bat, which is invoked by a "custom build step" during library generation. The script uses CMT with the command "build windefs <package>" to create a ".def" file of exported symbols found in a library file, for linking to create a non-Gaudi dll. It contains code to define
5. The fragment GlastPolicy/RootCintPolicy/cmt/fragments/vcproj_library_header is used to define a custom build step for running rootcint. For such packages, and for any package depending on it, it replaces the fragment of the same name from GlastCppPolicy. Two special macros are provided to disable this when it is not required, which must be defined by such packages

### Problems with extending or modifying the current system

- Many separate elements must be coordinated!
- Running the CMT build windefs command under VS2005 fails
- The custom build step used for creating dll's is not run at the correct time, or not at all

### Objectives of the new system

- Collect all the above elements into a single place, including both the logic and data describing the output files
- Use CMT only to manage packages and define their associated macros and environment variables to set.
- Build the Visual studio files in a new folder, to be called VS2005. (CMT has the name 'Visual' wired in.) This allows simultaneous test builds in the two environments.

### Requirements

For a given CMTPATH and CMTCONFIG, the new system must implement the following

- Obtain the list of accessible packages, and the dependencies of any one
- Extract the macros and sets defined by a package. There are three categories:
    - Compile time: includes, special compilation flags
    - Link time: libraries needed and special link flags
    - run time: paths for dll's and any special environment variables needed
- Create the vcproj files for each library or application project according to the build instructions, with an extra step for obtaining external symbols for dll creation if necessary, and special compile steps for cint
- Create the sln file for each package containing references to the project files, and, if requested, the project files for all used packages. Dependencies must be included to define a proper build order.
    - If used packages are requested, it must also be possible to specify that they not be part of the build. This is done in MRvcmt by making the requirements file read-only

### Implementation

The implementation is entirely in Python, with dependencies on:

- CMT for package management and macro definition
- ROOT for rootcint and bindexplib

The above tasks are accomplished by the following python submodules, in a pycmt package

- cmt.py -- interface to CMT (class CMT); class Package represents a package properties
- files.py -- class Files converts CMT files descriptions to a list of actual files
- vcproj.py -- class Vcproj (depending on CMT and Files) creates ".vcproj" file for a project
- workspace.py -- class Workspace
- devenv.py -- class Devenv runs the Visual Studio application
- main.py -- the module command line interface

### Current status

### ScienceTools

It is possible to build the ScienceTools packages that don't depend on **cppunit.** I am currently developing and testing astro, flux, and map_tools.

### GlastRelease

There are two current problems:

1. **Gaudi,** and all the libraries that it depends on, must be rebuilt with the new compiler--this is a painstaking process, and only Heather and Tracy have the expertise (the procedure is documented here)
2. The **Event** package must be build as a shareable. The symbols in the EventModel class are declared to be static, and explicitly exported, but links against the dll fail.

## What about the Release Manager?

The RM in fact uses a system that is completely independent of what I described above. It is written entirely in perl, and is not accessible to developers at the current time. Thus an apparent requirement is that RM be also extended, and that the various external libraries be made available.

## Integration with MRvcmt (or MRstudio)

The idea is for the the ruby code in either of these to invoke the appropriate functionality of this package to either build a package, or to start Visual Studio. DataMind, the contractor that maintains these packages, has not yet agreed to this. There is a request for MRvcmt to allow specification of the CMTCONFIG variable on the command line, and an attempt has been made, (adding a command line argument -c) but not consistently implemented.