How do I use the same pkg file for debug and release builds?

Within Carbide there is a useful protocol whereby you can use three macros in your pkg file. To quote the SDK

“The following symbols are available for PKG files in order to make reuse of PKG files more efficient across build targets. These symbols are only supported within Carbide.c++. They are:

$(EPOCROOT) — the location of the epoc32 folder but not including the epoc32 folder

$(PLATFORM) — platform designator, for example: ARMI, WINSC (Emulator) , GCCE, THUMB, etc.

$(TARGET) — target designator UDEB (Debug) or UREL (Release)”

This is a good thing because you don’t need to have special pkg files for release and debug builds. Sadly this is not available in the vanilla SDKs.

Carbide is good, but there are often times we have to implement batch build systems and have to rely on the SDK. The good news is you can do a half-baked/home-baked version yourself. The steps are as follows

1. Use an Extension makefile file in your bld.inf. The line will look something like:

```
makefile ..\sis\sis.make
```

The details of Extension makefiles are in the SDK.

2. In sis.make (you can call it what you want, but it has to match the line above) you can call a Perl script. Something like:

```
pkgname= pkg1

FINAL :
 echo using EPOCROOT=
 echo $(EPOCROOT)
 echo Writing pkg
 createpkgfiles.pl

 echo make the sis file
 echo Build and sign using
 makesis $(pkgname).pkg
 signsis -v -s $(pkgname).SIS $(pkgname)signed.sis somecert.cer somekey.key
 del /q $(pkgname).SIS
 ren $(pkgname)signed.sis $(pkgname).sis
```

3. Use “createpkgfiles.pl” to write the pkg files from templates.

```
@templates=
(
['template.pkg1.pkg', 'pkg1.pkg'],
['template.pkg2.pkg', 'pkg2.pkg'],
);


sub replaceStrings
{
 print "\nreplace strings\n";
 local $/;

 #Move the template to the target file
 for $i( 0 .. $#templates)
 {
  print "\n\ncopy /Y $templates[$i][0] $templates[$i][1]\n\n";
  system("copy /Y $templates[$i][0] $templates[$i][1]");

  open(FILE, $filename) or open(FILE, "$templates[$i][1]") or die("could not find $filename");
  $_=<FILE>;
  close (FILE);

  foreach $key (sort keys(%ENV))
  {
   #print "\n$key = $ENV{$key}";
   s/\$\($key\)/$ENV{$key}/gm;
  }

  open(FILE, ">$templates[$i][1]") or die("could not find $filename for writing");
  print FILE "$_";
  close (FILE);
 }

}
########################
#Start here
########################

replaceStrings();
print "\nEnd";
```

The script will take lines like in the template file

```
$(EPOCROOT)epoc32\release\$(PLATFORM)\$(CFG)\...
```


(C) Copyright Nokia 2013. All rights reserved.

and turn them into

```
C:\Symbian\9.1\S60_3rd_MR\epoc32\include\GCCE\UREL\...
```

In the pkg file words conforming to the $(varname) syntax will be replaced with the value of the value of the environment variable.

To find out what the environment is when the build is going on you can call this function in the script:

```
sub printEnv
{
 foreach $key (sort keys(%ENV))
 {
  print "\n$key = $ENV{$key}";
 }
}
```

The good thing is that you can use any environment variable, so you can set a few of your own and use them to vary the build and/or change the version in the pkg to match your app.

So, this function:

```
sub setVersion
{
 my($versionInfoFile)=@_;
 open(VER_FILE, $versionInfoFile);

 print "Get versions from $versionInfoFile\n";
 while($line=<VER_FILE>)
 {
  @words=split(/ +/, $line);
  if($words[1] eq 'VERSION_MAJOR')
  {
   $val=$words[2];
   chomp($val);
   print "VERSION_MAJOR= $val, ";
   $ENV{'BLD_VER_MAJOR'}=$val;
  }
  if($words[1] eq 'VERSION_MINOR')
  {
   $val=$words[2];
   chomp($val);
   print "VERSION_MINOR= $val, ";
   $ENV{'BLD_VER_MINOR'}=$val;
  }
  if($words[1] eq 'VERSION_BUILD')
  {
   $val=$words[2];
   chomp($val);
   print "VERSION_BUILD= $val";
   $ENV{'BLD_VER_BUILD'}=$val;
  }
 }
 close (VER_FILE);
}
```

Will read the version numbers out of a header that looks like this:

```
#define VERSION_MAJOR 4
#define VERSION_MINOR 0
#define VERSION_BUILD 49
```

And set the environment so that the line in the pkg template:

```
#{"$( BLD_PKG_NAME)", "$(BLD_PKG_NAME)"},(0x20000xxx),$(BLD_VER_MAJOR),$(BLD_VER_MINOR),$(BLD_VER_BUILD)
```

Will yield the hoped for result (If I had a penny for every time I forgot to update the pkg version and released something, I would have about 8p)

The basic idea is that the environment variables are there to use as part of the build process. You could use another language; I just use Perl because it is already there in the SDK. You can also customize your build in all sorts of ways once you get the hang of it.

Hope this helps.

 (C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.