

---

Subject: Versioning support (hack-ishly solved)  
Posted by [slashupp](#) on Tue, 30 Jul 2019 07:33:57 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

It would be nice/useful to have theide update version-numbers with each successful compile of a package.

With version numbers in the form release.build.test, theide can increment 'test' with each successful compile in Debug-mode, increment 'build' when compile succeeds in Release mode, leaving 'release' to be adjusted by the developer - possibly in Project:[package organizer] or [Main package configuration]

Then have macros available to get these from code, to keep in a variable for later use e.g.: to version-stamp data saved to file or db so that later releases/builds can correctly interpret the data-structure.

---

Subject: Re: Versioning support  
Posted by [slashupp](#) on Wed, 07 Aug 2019 12:17:57 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

I came up with the following macro that can do version-support.

It "key-jack"s F7 and it would also be nice if it can overload the toolbar-button.

for the macro to work it needs the following header to be part of the main package files:

Name: VersionRBT.h

```
#ifndef _VersionRBT_h_
#define _VersionRBT_h_

#define V_RELEASE 0
#define V_BUILD 0
#define V_TEST 0
#define N_2_S0(x) #x
#define N_2_S(x) N_2_S0(x)
#define V_VERSION "v " N_2_S(V_RELEASE) "." N_2_S(V_BUILD) "." N_2_S(V_TEST)

#endif
```

The #defines can be used wherever needed and the V\_VERSION is just a convenient string representation.

The Versioning-macro updates the #define counts (and restore if the build fails) before the

package is compiled  
making the new version current for the compiled code.  
Just use the V\_ defines where needed, they will always be current.

To test it, add the above header and import the macro ([Setup->Macro Manager.. then R-click the "Global macros"-tree and Import..])

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;
#include "VersionRBT.h"
```

```
GUI_APP_MAIN
{
    PromptOK("tester: " V_VERSION);
}
```

---

### File Attachments

1) [Versioning.usc](#), downloaded 151 times

---

---

Subject: Re: Versioning support  
Posted by [slashupp](#) on Sun, 11 Aug 2019 09:46:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Based on mirek's answer here: [https://www.ultimatepp.org/forums/index.php?t=msg&th=10750&goto=52210&#msg\\_52210](https://www.ultimatepp.org/forums/index.php?t=msg&th=10750&goto=52210&#msg_52210)  
I've improved my macro as per attached - much neater I think.  
thx mirek

---

### File Attachments

1) [Versioning.usc](#), downloaded 163 times

---

---

Subject: Re: Versioning support (hack-ishly solved)  
Posted by [slashupp](#) on Mon, 12 Aug 2019 06:00:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Added history-keeping to the header/macro...

The new header is:

```
#ifndef _VersionRBT_h_
#define _VersionRBT_h_
```

```

#define V_RELEASE 0
#define V_BUILD 0
#define V_TEST 0
#define N_2_S0(x) #x
#define N_2_S(x) N_2_S0(x)
#define V_VERSION "v " N_2_S(V_RELEASE) "." N_2_S(V_BUILD) "." N_2_S(V_TEST)

```

```

//=====
=====

```

```

//history will be updated by the Versioning-macro..
// and can be accessed with something like:
// {
//   std::string s{"History\n"};
//   for (auto p:VersionHistory<>) { s+="  ["; s+=p.first; s+="] -> "; s+=p.second; s+="\n"; }
//   PromptOK(DeQtF(s.c_str()));
// }

```

```

#include <map>

```

```

template<typename S=std::string> struct MVHIST
{
    std::map<S, S> m;
    MVHIST(const S &k, const S &v) { m[k]=v; }
    MVHIST<S>& operator()(const S &k, const S &v) { m[k]=v; return *this; }
    operator std::map<S, S>() { return m; }
};

```

```

template<typename S=std::string> std::map<S, S> VersionHistory=MVHIST<S>
//HISTORY (insertion-point, leave this comment in place - see macro)
;

```

```

#endif

```

and the macro is attached.

## File Attachments

1) [Versioning.usc](#), downloaded 163 times

---