Subject: Possible solution of "icpp problem"
Posted by mirek on Sat, 27 Aug 2016 14:09:14 GMT
View Forum Message <> Reply to Message

Hi,

"icpp issue" is longstanding problem that prevents U++ to be used as 'library' without TheIDE.

Brief summary of issue: if file has .icpp extension, U++ builder treats it differently and always includes corresponding .obj file into executable. Thus it is possible to put initialization code into it (using fake global constructor). That way, e.g. to support gif Image loading, all that has to be done is just add plugin/gif to project. plufin/gif/gif.icpp contains the code that registers gif format in Draw package.

Now if you rename gif.icpp to gif.cpp, build process will produce .obj and it gets into library, but linker will not link it into .exe because there are no references to functions in plugin/gif.

This was for quite a long time a problem when thinking about "U++ library".

Now I got finally a good idea how to resolve this issue:

We can put static constructor instance into "gif.h" calling required initialization code. That way, #include <plugin/gif/gif.h> will be required in client code, but that is not that much problem (in most cases, such #include will be there anyway) and as library, it is just fitting to include gif header to get gif support.

I can change U++ sources so that .icpp are still supported by U++ builder, but not really required when building outside umk/theide. .icpp will just call the same initialization hook as static constructor in the header (in reality, it will be easier to just call empty function in .cpp that contains INITBLOCK with required initialization...)

What do you think? Should I go that way?

Mirek

Subject: Re: Possible solution of "icpp problem"
Posted by Mindtraveller on Sun, 28 Aug 2016 08:07:27 GMT
View Forum Message <> Reply to Message

First of all, let's think do we really need to keep icpp-compatibility at all. Or it is possible to forget about them once and for all.

Talking about the solution proposed (to inject registration calls with header - if I understand it correctly), I think this is possible way, but it has one [possible] issue with initialization order. We can't predict the way headers are included and we can't predict the sequence of initialization.

I always wondered why plugins are not initialized inside CONSOLE_APP_MAIN/GUI_APP_MAIN or with auto-initialization classes. (I mean something like that: http://zeuxcg.org/2010/10/10/death-by-static-initialization/ )

Subject: Re: Possible solution of "icpp problem"
Posted by dolik.rce on Sun, 28 Aug 2016 11:57:49 GMT
View Forum Message <> Reply to Message

Hi Mirek,

That would be great! It would also make U++ much easier to use without theide. Using other IDEs or makefiles, cmake, etc. would lower the barrier for new users and could allow many more people to adopt U++.

Mindtraveller wrote on Sun, 28 August 2016 10:07First of all, let's think do we really need to keep icpp-compatibility at all. Or it is possible to forget about them once and for all.I'm definitely for dropping icpp completely, if possible. There are other uses, not only plugin-like stuff (e.g. files with Skylark handlers), but those could be solved in similar way I presume. Could there be a simple utility class for automatic initialization that could be used in all cases, even in user code?

Mindtraveller wrote on Sun, 28 August 2016 10:07Talking about the solution proposed (to inject registration calls with header - if I understand it correctly), I think this is possible way, but it has one [possible] issue with initialization order. We can't predict the way headers are included and we can't predict the sequence of initialization.If I remember how it works now with icpp, the order is not really well defined either. I think the init blocks are called in the order in which the icpp files are passed to the compiler.

Best regards,
Honza

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Sun, 28 Aug 2016 15:41:34 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 28 August 2016 10:07First of all, let's think do we really need to keep icpp-compatibility at all. Or it is possible to forget about them once and for all.

Talking about the solution proposed (to inject registration calls with header - if I understand it correctly), I think this is possible way, but it has one [possible] issue with initialization order. We can't predict the way headers are included and we can't predict the sequence of initialization.

That is non-issue. We cannot predict the order now as well - the code has to written in a way that order does not matter. Which actually is not that hard.

Quote:
I always wondered why plugins are not initialized inside CONSOLE_APP_MAIN/GUI_APP_MAIN)

That is certainly possible, but complicates client code contract. You will have to start to remember that package needs to be registered.

Quote:
with auto-initialization classes

The link you have posted suggests highly specialized case of what we need to do IMO. Anyway, the problem with code overhead mentioned in the article is real, it is something I will have yet to think about... (and yet again, icpp seems to be the superior solution to that...)

Mirek

---

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Sun, 28 Aug 2016 15:49:23 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Sun, 28 August 2016 13:57Hi Mirek,

That would be great! It would also make U++ much easier to use without theide. Using other IDEs or makefiles, cmake, etc. would lower the barrier for new users and could allow many more people to adopt U++.

Mindtraveller wrote on Sun, 28 August 2016 10:07First of all, let's think do we really need to keep icpp-compatibility at all. Or it is possible to forget about them once and for all.I'm definitely for dropping icpp completely, if possible. There are other uses, not only plugin-like stuff (e.g. files with Skylark handlers), but those could be solved in similar way I presume. Could there be a simple utility class for automatic initialization that could be used in all cases, even in user code?

Well, first of all, let us not to forget that the problem we really need to have solved is to force linker to include some .obj files from .lib (or .o from .a ... even if none of symbols is referenced from the rest of the code. What I am proposing is basically putting a dummy call from header to such .obj, just to have it 'activated'.

Could be equivalent of

Gif.h

struct GifInitializer { GicInitializer() { GifInitializerFunction(); } };

static GifInitializer DummyName;

Gif.cpp (former Gif.icpp)

void GifInitializerFunction() {}

INITBLOCK {
  ... register GIF in Draw
}

---

Subject: Re: Possible solution of "icpp problem"
Posted by Mindtraveller on Sun, 28 Aug 2016 15:57:34 GMT
View Forum Message <> Reply to Message

If sequence is not a issue, I definitely vote for the simple solution.

---

Subject: Re: Possible solution of "icpp problem"
Posted by dolik.rce on Sun, 28 Aug 2016 18:37:37 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 28 August 2016 17:49Well, first of all, let us not to forget that the problem we really need to have solved is to force linker to include some .obj files from .lib (or .o from .a ... even if none of symbols is referenced from the rest of the code. What I am proposing is basically putting a dummy call from header to such .obj, just to have it 'activated'.
For me the issue always was "there is something non-standard that must be done when compiling/linking". If a static dummy object everything that needs to be done to turn this into "everything works with fine with all common C++ build systems" situation, than it will make me really happy

If the code stays as simple as you suggested, you could easily hide it into pair of simple macros. That way, user can just add one line to a file that would previously be .icpp and another line to header or anywhere else to force correct linking. It would be handy, for example in the Skylark case I already mentioned.

Honza

---

Subject: Re: Possible solution of "icpp problem"
Posted by koldo on Mon, 29 Aug 2016 07:18:35 GMT
View Forum Message <> Reply to Message

Good news, thank you Mirek.

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Thu, 15 Sep 2016 07:14:57 GMT
View Forum Message <> Reply to Message

Funny facts: I have now tried this with plugin/png in a large application (15MB .exe). The application size grows by 3KB and the empty dummy initialization routine gets called 170 times.

As we have about 17 icpp files, this seems to be acceptable (~40KB app size increase, about 3000 dummy calls).

Subject: Re: Possible solution of "icpp problem"
Posted by kov_serg on Thu, 22 Sep 2016 14:44:23 GMT
View Forum Message <> Reply to Message

Why not to use explicit method?

```
// upp_static_init.cpp
void upp_static_init() {
  {void plugin_gif_init(); plugin_gif_init();}
  // or this way
  #include <plugin/gif/gif.init>
  ...
}

[plugin/gif/gif.init]
{void plugin_gif_init(); plugin_gif_init();}
```

you'll have control over order of initialization and it looks like ordinary include.
And this file can be easyly generated with small tool.

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Thu, 22 Sep 2016 15:02:06 GMT
View Forum Message <> Reply to Message

kov_serg wrote on Thu, 22 September 2016 16:44Why not to use explicit method?

```
// upp_static_init.cpp
void upp_static_init() {
  {void plugin_gif_init(); plugin_gif_init();}
  // or this way
  #include <plugin/gif/gif.init>
```

```
  ...
}

[plugin/gif/gif.init]
{void plugin_gif_init(); plugin_gif_init();}
```

you'll have control over order of initialization and it looks like ordinary include.
And this file can be easyly generated with small tool.

Actually, this sounds like existing solution - notice there are autogenerated 'init' files in packages, which serve the exactly same purpose.

However, it looks like nobody likes this solution...

Mirek

---

Subject: Re: Possible solution of "icpp problem"
Posted by kov_serg on Thu, 22 Sep 2016 22:32:19 GMT
View Forum Message <> Reply to Message

May be something like this?

```
// lib1.h
#pragma once

int lib1_init();

#ifndef IMPL
int lib1_init_res=lib1_init();
#endif


// lib1.cpp
#define IMPL
#include "lib1.h"
#include <stdio.h>

int lib1_init() { printf("lib1\n");return 0; }
```

---

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Fri, 23 Sep 2016 07:26:31 GMT
View Forum Message <> Reply to Message

kov_serg wrote on Fri, 23 September 2016 00:32May be something like this?

```
// lib1.h
#pragma once

int lib1_init();

#ifndef IMPL
int lib1_init_res=lib1_init();
#endif


// lib1.cpp
#define IMPL
#include "lib1.h"
#include <stdio.h>

int lib1_init() { printf("lib1\n");return 0; }
```

Sure, this is basically what this forum thread proposes in the first post

---

Subject: Re: Possible solution of "icpp problem"
Posted by mr_ped on Sun, 25 Sep 2016 02:16:11 GMT
View Forum Message <> Reply to Message

With C++17 you can create global variable in header file like this:

```
inline gif_plugin_class gif_plugin;
```

I'm not sure how well the order of initialization can be controlled in such case, but it may be worth of some experiments.

---

Subject: Re: Possible solution of "icpp problem"
Posted by kov_serg on Mon, 26 Sep 2016 15:05:59 GMT
View Forum Message <> Reply to Message

How about another explicit method

Generate in project dir file like this

```
// static_init.cpp
#define INIT(lib) extern int lib##_init(); lib##_init();
static int init() {
    INIT(lib1)
```

```
    INIT(lib2)
    INIT(lib3)
    // ....
    return 0;
}
static int init_result=init();
```

and link it to project. Just like .def file.
This is simple and you can control order of initialization. And no need of inline valiables support.

---

Subject: Re: Possible solution of "icpp problem"
Posted by mirek on Mon, 26 Sep 2016 15:43:15 GMT
View Forum Message <> Reply to Message

kov_serg wrote on Mon, 26 September 2016 17:05How about another explicit method

Generate in project dir file like this

```
// static_init.cpp
#define INIT(lib) extern int lib##_init(); lib##_init();
static int init() {
    INIT(lib1)
    INIT(lib2)
    INIT(lib3)
    // ....
    return 0;
}
static int init_result=init();
```

and link it to project. Just like .def file.
This is simple and you can control order of initialization. And no need of inline valiables support.

We do not need to control order of initialization. That is pointless as it is easy to write order neutral initialization code.

Other than that, 'link it to the project, just like .def' creates about the same problem as original .icpp - works great if you have control about build process. .icpp is perfect solution to the problem - the only problem is that it is unusual and no other build system supports it. That is something I am trying to solve. People just expect .cpp and .h (or .h and .lib), any other 'magic' is not welcome.

Mirek

P.S.: I have thought about inline variables and I think they are not much advantage over good old global initialization.

---

Subject: Re: Possible solution of "icpp problem"
Posted by kov_serg on Mon, 26 Sep 2016 16:35:01 GMT
View Forum Message <> Reply to Message

Where will be *.h and *.cpp or *.lib and readme

And script like this

```
#!/bin/sh

lines_to_check=8
cat <<'EOF'
// static_init.cpp
static int init() {
EOF
find *.h -type f -exec head -n $lines_to_check {} + | grep -P '^//\s+static_init:\s+' | awk '{print "
extern int "$3"(); "$3"();"}'
cat <<'EOF'
  return 0;
}
static int init_result=init();
EOF
```

It look for h files this comment line in first 8 lines
// static_init: lib1_init

And then you can comment unused parts and use it like config file