Subject: operator<<= in tree control reference example
Posted by gprentice on Wed, 05 Jan 2011 09:55:53 GMT
View Forum Message <> Reply to Message

In the tree control reference example there is this code

```
  void LoadTree(int parent, const String& path, Progress& pi)
  {
    pi.SetText(DeFormat(path));
    for(FindFile ff(AppendFileName(path, "*.*")); ff; ff.Next()) {
      if(pi.StepCanceled())
        return;
      String n = ff.GetName();
      if(n != "." && n != "..") {
        edit.Add();
        edit.Top() <<= n;
```

Why does the last line use operator<<= instead of operator= ?

If there's no reason then it should be changed to operator= because it took me a while to track down the fact that it seems to go through Ctrl::operator<<= which goes through the SetData virtual function into EditField::SetData. For a newcomer or even not for a newcomer, it's not easy to find what that operator<<= does - it doesn't make any sense to use it in an example.

Also, why does the above code do an unnecessary call to the Top() function - couldn't it just be edit.Add() = n;

And lastly (off topic but related to the edit.Add function call), does U++ assume that memory allocation with new never fails - how does U++ handle failure?

Graeme

Subject: Re: operator<<= in tree control reference example
Posted by cbpporter on Wed, 05 Jan 2011 10:07:25 GMT
View Forum Message <> Reply to Message

Well <<= is pretty standard for U++, and comes often in pair with ~, which does the opposite. For controls it is SetValue/GetValue. It is just one of the conventions you need to learn and what contributes to the learning curve of U++. It is the same with standard C++ and << for streams and other conventions.

The reason "=" is not used because it would be confusing. You use "=" between different types, but only if they are near identical/easily convertible. Like different representation of the same data. But an EditField is not a Value. It has a Value. This is why we use <<=, or the "kind of assign operator, but more of s take this value and use it" operator.

Until you are comfortable with this, you can avoid operator overloading in your code.

As for memory allocation failures, it is very hard to recover for them in a reliable way. It is easy if you have a malloc, but in real and complex applications with advance memory allocation it is hard. So U++ should print an error message and fail.

Subject: Re: operator<<= in tree control reference example
Posted by gprentice on Wed, 05 Jan 2011 10:31:56 GMT
View Forum Message <> Reply to Message

Well I should have said that "edit" is an EditString and it provides an operator= for assigning a string (and why not a constructor too) so I think operator= should be used in this example.

Regarding the learning curve, do you have any idea how a newcomer is supposed to learn that operator<<= is "standard for U++". You cannot easily navigate your way to Ctrl::operator<<= using theide. U++ examples should have source code comments for obscure things like this.

You didn't respond to the question about the call to the Top() function.

Quote:So U++ should print an error message and fail.
What does this mean exactly? I just wanted to know what U++ actually does.

Graeme

Subject: Re: operator<<= in tree control reference example
Posted by cbpporter on Thu, 06 Jan 2011 09:32:03 GMT
View Forum Message <> Reply to Message

Well, the fastest way to get the answer would be if you do the test yourself. Make new fail by allocating to much memory and see what happens .

Or quote from the docs:
Quote:
We decided to ignore possibility of "out-of-memory" exceptions and recovery. If U++ application goes out of memory, it simply prints the error message and terminates. This is quite pragmatic resolution - our experience is that it is quite hard and annoying to achieve robustness here and it cannot be reliably tested. Also, most platforms with virtual memory will almost freeze long before out-of-memory problem due to intensive page swapping. Connected issue - default and copy constructors are not allowed to throw exceptions in U++ (the common reason to throw exception here was out-of-memory condition). This limitation will be removed in future releases.

As for the use of Top, I have no idea. Ask the author of the package. I would not use a Top after and Add if the Add returns a reference, and I'm not sure if there is a real reason for it.

The way newcomers learn this is by going over the code, the documentation, asking on the forum and applying the conventions themselves. It is something you will hit your head on, overcome it and learn from your experience. This solution is not idle, but there is no easy way to point people in the right direction from the start and make sure they do head that way.

Subject: Re: operator<<= in tree control reference example
Posted by gprentice on Thu, 06 Jan 2011 10:33:36 GMT
View Forum Message <> Reply to Message

Termination for out of memory is fine. Thanks for hunting out that information.

I don't like that overloaded operator much. It makes the code hard to read, you can't tell what the programmer's intention was and compilers are liable to mis-compile it because of the number of operator<<= around. I'm also wondering why Ctrl::SetData is even virtual at all. Anyway, thanks.

Graeme

Subject: Re: operator<<= in tree control reference example
Posted by gprentice on Thu, 06 Jan 2011 10:53:10 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 05 January 2011 23:07Well <<= is pretty standard for U++, and comes often in pair with ~, which does the opposite. For controls it is SetValue/GetValue. It is just one of the conventions you need to learn and what contributes to the learning curve of U++. It is the same with standard C++ and << for streams and other conventions.

The reason "=" is not used because it would be confusing. You use "=" between different types, but only if they are near identical/easily convertible. Like different representation of the same data. But an EditField is not a Value. It has a Value. This is why we use <<=, or the "kind of assign operator, but more of s take this value and use it" operator.

Actually I can't even see how that operator<<= works.
edit is declared like this
Array<EditString> edit;

and the code does

edit.Top() <<= n;
where n is a String. So the LHS is EditString&. The only operator<<= I can find is in the Ctrl base class
  const Value& operator<<=(const Value& v)   { SetData(v); return v; }

but the call to SetData (not SetValue?) is not virtual and just throws the data away - yet the code works ??? I must be missing something?

Subject: Re: operator<<= in tree control reference example
Posted by dolik.rce on Thu, 06 Jan 2011 11:12:15 GMT
View Forum Message <> Reply to Message

gprentice wrote on Thu, 06 January 2011 11:53The only operator<<= I can find is in the Ctrl base class
  const Value& operator<<=(const Value& v)   { SetData(v); return v; }

but the call to SetData (not SetValue?) is not virtual and just throws the data away - yet the code works ???

The fact you are missing is that SetData() is virtual That way, if you override it in any Ctrl derived class you get automatically the <<= operator, since it is defined in Ctrl in such way that it calls the overridden method from the child class. There is no SetValue() (except some special Ctrls, like DropList, TreeCtrl etc.), the general interface for operating value of Ctrls is made entirely by SetData(), GetData() and their respective operators <<= and ~.

Honza

Subject: Re: operator<<= in tree control reference example
Posted by gprentice on Thu, 06 Jan 2011 11:26:28 GMT
View Forum Message <> Reply to Message

yep, I know SetData is a virtual function but a call to a virtual function is "virtual" only when the call is made on a pointer or reference   - except, I've just looked up a C++ book I have and in a member function (such as operator<<=) an unqualified call is also virtual because it's equivalent to this->SetData - hmm, I probably knew that once but I've long since forgotten!

Graeme
