Peless

An example project

Where is this presentation?

- `http://http.FreeBlackPatchPanel.com/pme/linux/peless.sxi`

Peless Choices

I. Choose goal application. X11 text file lister
    A) xless, still exists
    B) gless does not come with new distros anymore.
    C) kless (not the klingon) ditto.

II. Choose programming environment=C++
    A) I have experience with C++ and I like it.
    B) I have also used Java.

III. Choose X11 library=GTKMM

IV. Implementation details, details.

V. Choose project Home

II B, I have also used Java.

1. Spellchecker
2. Bondage and disipline Language
   a) We won't let you make a mistake.
   b) Therefore, we can't let you have any freedom.
   c) The ***suits*** love it.
   d) Resources
      i. Memory (garbage collector)
      ii. Other resources
   e) Not open source!

Java memory management

```
public void myMethod() {
Date date = new Date();
// do stuff
// never worry about deleting
// no memory leaks!
}
```

C++ Resource management

- Construction is resource acquisition destruction is resource releasing

```
void f(char* name)
{
  ofstream out(name);
  out << "hello world" << endl;
  // no need to close out
};
```

C++ memory

- Static or global memory (don't use. With exceptions)
- local (stack) memory.
- dynamic memory (new) Big problem.

My C++ memory Philosophy
C++ pointers Don't use them! (almost)

- (not original)
- C++ pointers are a powerful feature in the C++ language. They allow powerful framework classes to be created, and give the programmer total control over access to memory. In C++, they are a necessary reality.
- C++ pointer should be used by experts to create basic classes like containers.

C++ pointer misuse.

- But for your typical Joe Six pack application programmer, C++ pointers are too powerful to use. There are too many subtile ways to create a memory leak or a reference through a pointer that is no longer valid.
- This leads to bugs!
- The misuse of C++ pointers is very costly, and if it is allowed to continue, the ***SUITS*** may require us to use inferior languages that don't have pointers like JAVA.

Good News: Raw C++ pointers can be hidden!

- With proper use of C++ tools almost all use of raw C++ pointers can be hidden. The typical application program should encapsulate away and hide almost all uses of C++ pointers using these tools. This discipline will eliminate memory leaks and invalid pointer references!

Three common uses for pointers.

- Pointer arithmetic to navigate arrays!
- To pass objects around from method to method or from object to object.(No ownership(=deletion responsibility) involved.)
- Object allocation.

pointer arithmetic to navigate arrays!

- Use the STL vector template instead!
- If you do need to use pointer arithmetic to navigate arrays get your code checked 3 times by an expert especially if there is inheritance involved with the base type.

To pass objects around from method to method or from object to object.

- (No ownership(=deletion responsibility) involved.)
- **Use C++ references instead.**

Object allocation

- the lifetime of the object corresponds to a program block ({ }) (or a program block that could exist.)
  - use local objects instead.
- The lifetime of the object corresponds to the lifetime of another class.
  - use contained sub-objects "hasa" instead.
- The object is truly dynamic and its life time does not correspond to the lifetime of any other object or code block. This is the hard case.
  - Keep the object inside a "owner" object.

Program Block {}

- Bad

```
{
  MyClass *
    my_instance_pointer( new MyClass() );

  mess_with_it( *my_instance_pointer );

  delete my_instance_pointer;
};
```

Program Block {}

- Good

```
{
  MyClass my_instance;
  mess_with_it(my_instance);
};
```

- The my_instance will automaticly be deleted when the program block is exited at "}" . It is even less lines of code!

Digression

If we were coding in java the above would not be good. I forgot to tell you that the destructor on MyClass pushes the control rods back into the reactor, ending the nuclear experiment! This is according to the C++ idiom object construction is resource allocation, object destruction is resource deallocation. You always push in the control rods when you are through with the nuclear reactor! (deallocating it.) When you are coding something in java and you want something to happen when you are through with an object, you damm well better code it yourself, because you might not want to wait for the garbage colector to get it. There is a finalize method, but don't forget to call it! The reactor can get hot very quickly!

Digression

The JAVA people think that since they handle the allocation/deallocation of memory so well, from their point of view, that they do not have to help handle other resorces. If they did they would have proper destructors! Memory is not the only resource! Files need to be closed, and the lights need to be turned off!

Class lifetime

```
class NewClass
{
 private:
   MyClass * my_instance_pointer;
 public:
   NewClass():my_instance_pointer(new MyClass() )
   {
     OTHER STUFF
   };
   ~NewClass()
   {
     OTHER DESTRUCTOR STUFF
     delete my_instance_pointer;
     MORE OTHER DESTRUCTOR STUFF
   };
   A LOT OF OTHER STUFF
 };
```

Class lifetime

```
class NewClass
{
 private:
   MyClass my_instance;
 public:
   NewClass(): my_instance() )
   {
     OTHER STUFF
   };
   ~NewClass()
   {
     OTHER DESTRUCTOR STUFF
     MORE OTHER DESTRUCTOR STUFF
   };
   A LOT OF OTHER STUFF
};
```

truly dynamic lifetime

- The object is truly dynamic and its life time does not correspond to the life time of any other object or code block.
- This is the hard case.

truly dynamic lifetime, bad example

- Here is an example where we must create an object, compute on it, then possibly add to a container which will take responsibility for deleting it. This happens a lot in GUI programming.

```
{
 MyClass * my_instance_pointer( new MyClass() );
 mess_with_it( *my_instance_pointer );
 my_container.add(my_instance_pointer);
  // my_container will now assume responsibility for
  // deletion.

};
```

truly dynamic lifetime, solution

- Keep the object inside a "owner" object at all times which has responsibility for deletion. This "owner" object can be a container or if nothing else will do a std::auto_ptr or boost::smart_ptr. (I think of these objects as single item "containers".)

truly dynamic lifetime, good example

```
{
 std::auto_ptr<MyClass>
  my_instance_pointer_auto( new MyClass() );

 mess_with_it( *my_instance_pointer_auto );


  my_container.add( my_instance_pointer_auto.re
  lease() );
 // ownership of the instance (deletion
 // responsibility)
 // is transfered from the auto_ptr to the
 // container.
};
```

Peless Choices

I. Choose goal application. X11 text file lister
II. Choose programming environment=C++
III. Choose X11 library=GTKMM
    A) QT
    B) GTKMM
IV. Implementation details, details.
V. Choose project Home

QT vs GTKMM

- QT was invented before the new standard and templates. It therefore, uses a non-standard way of handling slots that can now be done in standard C ++.
- QT requires raw pointers in it's interface and therefore is incompatible with the Paul Elliott memory philosophy (TM).
- GTKMM is newer with a changing interface.
- GTKMM is not perfect.
- KDE and GNOME will run both QT, and GTKMM apps!

From GTKMM FAQ:

- 1.5 Why not just use Qt if you like C++ so much?
  - gtkmm developers tend to prefer gtkmm to Qt because gtkmm does things in a more C++ way. Qt originates from a time when C++ and the standard library were not standardized or well supported by compilers. It therefore duplicates a lot of stuff that is now in the standard library, such as containers and type information. Most significantly, they modified the C++ language to provide signals, so that Qt classes can not be used easily with non-Qt classes. gtkmm was able to use standard C++ to provide signals without changing the C++ language.

Boost libraries!

- Everyone uses the boost libraries!
- Has basic classes that should be in std:: but they did not have time to put them in!
- Before developing any class that is so general that everyone would want to use it, check if it is already in the Boost libraries!
- Includes powerful functor library!
- Regular expression search library.
- shared_ptr referenced count smart pointer!

Automake, autoconf, libtool, or bjam?

- It is fashionable to criticize auto*, but have you looked at the fixed bugs history?
- It has to work in an enormous number of runtime or build environments!
- When you have come up with something that works in all these enviroments and has be as debugged as completely as the auto* tools, contact me!
- The boost library people assume you are going to use bjam! They don't provide a “boost-config”.

How I learned IO.

```
      PROGRAM TEST
      INTEGER I,J
      WRITE(61,800)'ENTER I'
      READ(60,801)I
      J = ICALC(I)
      WRITE(61,802)'THE ANSWER IS',J
  800 FORMAT(1X,A)
  801 FORMAT(I5)
  802 FORMAT(1X,A,I6)
      END
```

Asynchronous IO

- New fangled asyncronous IO programing.
- Create your main window or main dialog.
- Add your callbacks.
- Call the main event loop, which runs till the end of the freaking program!
- The freaking UI is in control and your code gets called back!
- Could be “don't call us, we'll call you programing”

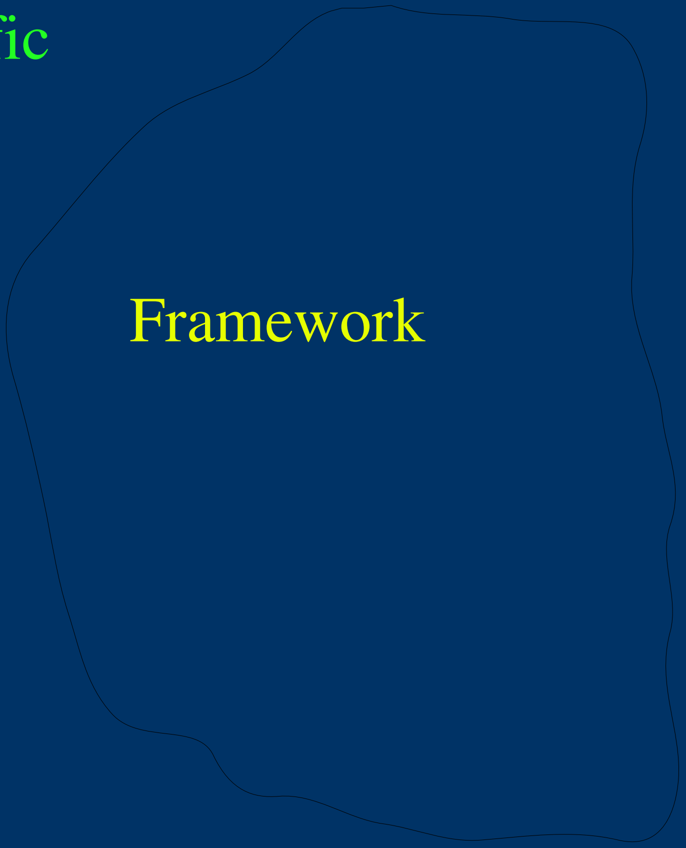
Old way: How does a window framework do a call back?

Framework written w/o specific knowledge of users' classes

Users classes derived from Framework classes.

User writes his classes later.

How can framework callback users' classes?



Old way: How can a framework callback a users classes?

- Answer it can't. The users' class has not been written yet.
- The framework calls back one of it's own classes.
- But the class called back is polymorphic i.e. with virtual functions.
- The user derives from this class, overriding the method that will be called back.
- The user passes this object to the framework as it's base class. When the framework calls back, the derived method (written by the user) gets control.

Polymorphic callback Example:

- IBM user interface class library. (Presentation manager). ICommandHandler has virtual function called “command”.
- You derive from ICommandHandler overridding command.
- Pass one of these objects to IBM UICL.
- When the callback happens your derived code gets control.
- Some people do not like this style.
- And so the concept of signals and slots was invented!

Qt signals and slots.

- QT was written before all the features of the current C++ standard was nailed down for general use.
- QT uses “const char*” for signal names and a nonstandard moc compiler for signals and slots.
- Trolltech admits this is not as fast as gtkmm's templates.
- Benchmark:
- http://libsigc.sourceforge.net/benchmark.shtml

GTKMM uses templates for signals and slots.

- Templates make many people's brains hurt!
- The gtkmm solution uses libsigc++.
- The gtkmm solution is 100% standard C++.
- Templates are here to stay get used to it!
- You can do things with templates that can not be conveniently done with other software technologies.

libSigC++

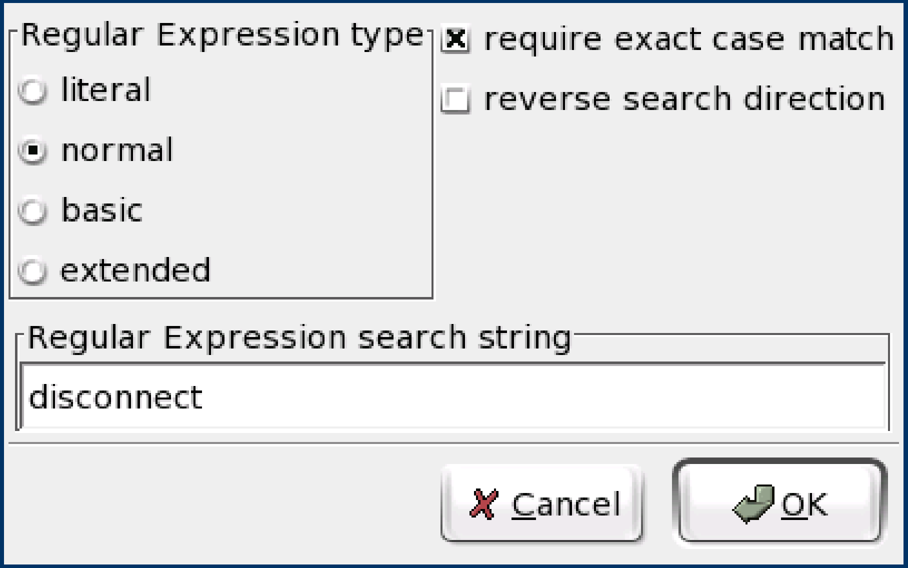
- You can connect the methods of any class.
- You can “bind” in parameters to be passed to called method.
- You can “hide” signal parameters that are not needed.
- Don't forget to disconnect the slot when the object or the parameters go away. (In the destructor.)

String searching

- can search for literals or regular expressions.
- Forward or reverse directions.
- regular expressions can be
  - Normal – like perl
  - Basic – like sed
  - Extended – like egrep.

Search dialog

- This dialog is more complicated than it looks!
- Vertical box
  - 2 Horizontal boxes
    - radio buttons
    - checkboxes
- Entry frame
  - Text entry.


Regular Expression type

literal

[✓ normal]

basic

extended

[✓ require exact case match]

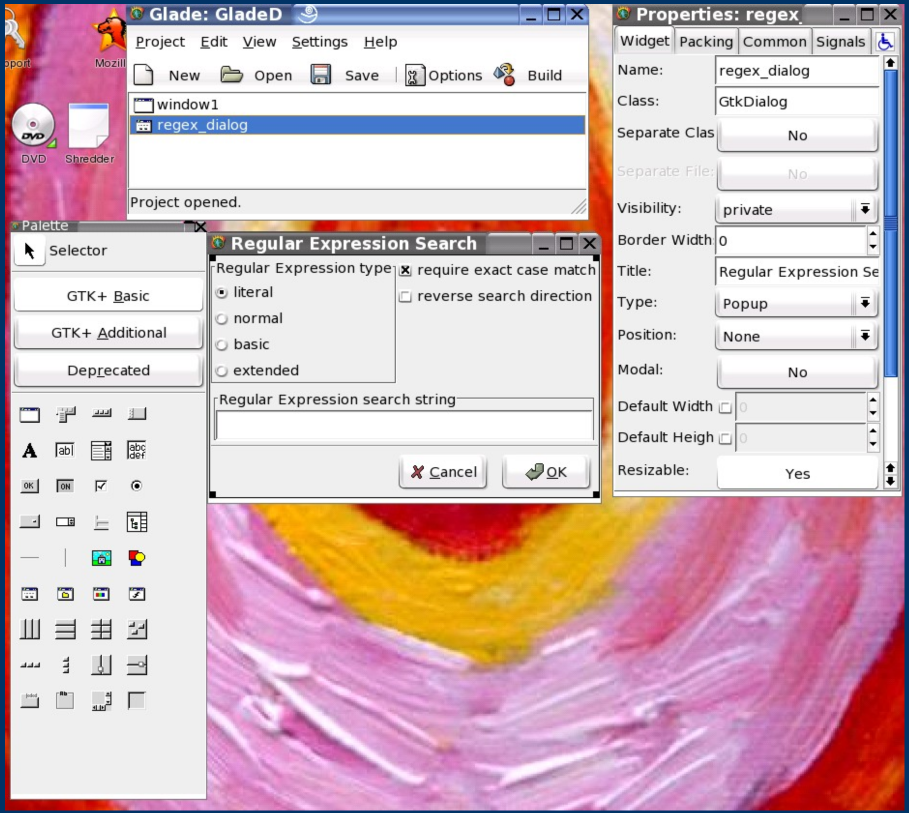
reverse search direction

Regular Expression search string

disconnect

Glade-2, a point and clicky fancy UI dialog generator.

- Unfortunately generates code with raw pointers!
- I uses glade-2 to create the code, then I Hack the hell out it, changing pointers to contained sub objects!
- This makes it compatible with the Paul Elliott Memory management philosophy (TM).


Glade: GladeD
Project Edit View Settings Help
New Open Save Options Build
Palette
Selector
GTK+ Basic
GTK+ Additional
Deprecated
Regular Expression Search
Cancel
OK
Properties: regex
Widget Packing Common Signals
DVD
Shredder

window1
regex_dialog

Project opened.

Regular Expression type
- (●) literal
- ( ) normal
- ( ) basic
- ( ) extended

[x] require exact case match
[ ] reverse search direction

Regular Expression search string

Name:	regex_dialog
Class:	GtkDialog
Separate Clas	No
Separate File:	No
Visibility:	private
Border Width	0
Title:	Regular Expression Se
Type:	Popup
Position:	None
Modal:	No
Default Width	0
Default Heigh	0
Resizable:	Yes

Boost regular expression searcher.

- Is proposed for C++ standard.
- Boost uses iterator over wchar.
- Gtkmm uses ustrings which uses iterator over gunichar.
- but gunichar staticly convert to wchar.
- convertion adapter.

You must use a source control system!

- Even if you are a lone wolf developer, you will want to be able to track your history and possibly roll-back changes.
- Unless a mad dog sues you!
- Start using one now, you will need it if anyone volunteers to help you.

CVS vs subversion.

- CVS is older and more widely used and supported.
- Subversion is new and avoids CVS's hacks.
- Subversion can rename a directory!
-
- I use subversion.
- Both systems allow you to start with a local filesystem database, and convert to network system later.
- Put your files in a “leaf” directory, never in the trunk!

Chose an open source development site.

- Requirements:
  - Free (free beer).
  - open source development host site.
  - support subversion.
- Berlios: (Berlin) Are there any others?
- http://developer.berlios.de/
-
- Read the Site documentation!

Start up your project, Berlios

- Request userid, and project. (24-48) hours.
- Choose a non guessable password.
- They will give you a shell account on shell.berlios.de.
- ssh-keygen to generate your self a ssh-key.
  - use protocol 2.
  - make password non guessable.
- For your shell account, simply copy your $HOME/.ssh/yourkey_dsa.pub to "$HOME/.ssh/authorized_keys" on shell.berlios.de.

Shell script to use BerliOS:

```
#!/bin/bash
eval `ssh-agent`
export SVN_SSH="ssh -2 -l pelliott"
if ssh-add ~/.ssh/BerliOS;
then
echo Berlios Key added.
else
exit
fi
nohup konsole -T BerliOS --name "BerliOS
  konsole" >/dev/null 2>&1 &

exit
```

SSH access.

- The previous slide will allow you to ssh, scp and svn to berlios without typing a password.

Import your existing svn tree into berlios.

- The berlios documentation describes how to import your existing tree:
- https://developer.berlios.de/docman/display_doc.php?docid=394&group_id=2#import
- Requires a 6 hour wait, but you only have to do it once.
-
- Check out your source tree from berlios and check that it is correct.

SVN usage.

- You should now be able to use svn just as on your local system. You can:
  - commit files.
  - track historys.
  - checkout files.
  - all the other svn stuff.

Checkout your files with SVN!

- `svn checkout svn+ssh://svn.berlios.de/svnroot/repos/peless`
-
- This will give you a copy of the project to work with.
-
- `svn commit --editor myeditor my-source-file`
-
- This commits your changes

Create a web page to describe your project!

- Described in
- https://developer.berlios.de/docman/display_doc.php?docid=43&group_id=2
- The web pages generated by Berlios are fixed format. This page is freeform. Describe in your own words what the heck your project is and what it is good for.
- Include a link back to your host berlios.
- http://peless.berlios.de/
-

Get some software testers!

- This step is hard.
-
- Beg and Plead, Beg and Plead some more!
-
- Any bugs that you find fix.
-
- Continue!
-
- Your project has reached the debugging phase!

Debuging with gdb!

- `CXXFLAGS=-g    ./configure your-parameters`
- `make`
- `gdb myprogram`
- `set args   parameters to program`
-
- You can step to next instruction “n”
- Step into routine “s”
- Set breakpoints run or continue!