Builder in C++

Back to **Builder** description

Builder design pattern demo

Discussion. The forte of Builder is constructing a complex object step by step. An abstract base class declares the standard construction process, and concrete derived classes define the appropriate implementation for each step of the process. In this example, "distributed work packages" have been abstracted to be persistent and platform independent.

This means that the platform-specific mechanism for implementing files, queues, and concurrency pathways is defined in each platform's concrete derived class. A single "reader" object (i.e. parser) retrieves the archived specification for a `DistrWorkPackage` and proceeds to delegate each build step to the builder object that was registered by the client. Upon completion, the client retrieves the end result from the builder.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>

enum PersistenceType
{
  File, Queue, Pathway
};

struct PersistenceAttribute
{
  PersistenceType type;
  char value[30];
};

class DistrWorkPackage
{
  public:
    DistrWorkPackage(char *type)
    {
        sprintf(_desc, "Distributed Work Package for: %s", type);
    }
    void setFile(char *f, char *v)
    {
        sprintf(_temp, "\n  File(%s): %s", f, v);
        strcat(_desc, _temp);
    }
    void setQueue(char *q, char *v)
    {
        sprintf(_temp, "\n  Queue(%s): %s", q, v);
        strcat(_desc, _temp);
    }
    void setPathway(char *p, char *v)
    {
        sprintf(_temp, "\n  Pathway(%s): %s", p, v);
        strcat(_desc, _temp);
    }
    const char *getState()
    {
        return _desc;
    }
  private:
    char _desc[200], _temp[80];
};

class Builder
{
  public:
```

```
    virtual void configureFile(char*) = 0;
    virtual void configureQueue(char*) = 0;
    virtual void configurePathway(char*) = 0;
    DistrWorkPackage *getResult()
    {
        return _result;
    }
  protected:
    DistrWorkPackage *_result;
};

class UnixBuilder: public Builder
{
  public:
    UnixBuilder()
    {
        _result = new DistrWorkPackage("Unix");
    }
    void configureFile(char *name)
    {
        _result->setFile("flatFile", name);
    }
    void configureQueue(char *queue)
    {
        _result->setQueue("FIFO", queue);
    }
    void configurePathway(char *type)
    {
        _result->setPathway("thread", type);
    }
};

class VmsBuilder: public Builder
{
  public:
    VmsBuilder()
    {
        _result = new DistrWorkPackage("Vms");
    }
    void configureFile(char *name)
    {
        _result->setFile("ISAM", name);
    }
    void configureQueue(char *queue)
    {
        _result->setQueue("priority", queue);
    }
    void configurePathway(char *type)
    {
```

```
        _result->setPathway("LWP", type);
    }
};

class Reader
{
  public:
    void setBuilder(Builder *b)
    {
        _builder = b;
    }
    void construct(PersistenceAttribute[], int);
  private:
    Builder *_builder;
};

void Reader::construct(PersistenceAttribute list[], int num)
{
  for (int i = 0; i < num; i++)
    if (list[i].type == File)
      _builder->configureFile(list[i].value);
    else if (list[i].type == Queue)
      _builder->configureQueue(list[i].value);
    else if (list[i].type == Pathway)
      _builder->configurePathway(list[i].value);
}

const int NUM_ENTRIES = 6;
PersistenceAttribute input[NUM_ENTRIES] =
{
  {
    File, "state.dat"
  }
  ,
  {
    File, "config.sys"
  }
  ,
  {
    Queue, "compute"
  }
  ,
  {
    Queue, "log"
  }
  ,
  {
    Pathway, "authentication"
  }
```

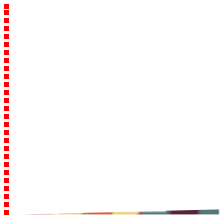
```
  ,
  {
    Pathway, "error processing"
  }
};

int main()
{
  UnixBuilder unixBuilder;
  VmsBuilder vmsBuilder;
  Reader reader;

  reader.setBuilder(&unixBuilder);
  reader.construct(input, NUM_ENTRIES);
  cout << unixBuilder.getResult()->getState() << endl;

  reader.setBuilder(&vmsBuilder);
  reader.construct(input, NUM_ENTRIES);
  cout << vmsBuilder.getResult()->getState() << endl;
}
```

Output

```
Distributed Work Package for: Unix
  File(flatFile): state.dat
  File(flatFile): config.sys
  Queue(FIFO): compute
  Queue(FIFO): log
  Pathway(thread): authentication
  Pathway(thread): error processing
Distributed Work Package for: Vms
  File(ISAM): state.dat
  File(ISAM): config.sys
  Queue(priority): compute
  Queue(priority): log
  Pathway(LWP): authentication
  Pathway(LWP): error processing
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.



All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.

Learn more

Code examples

Java	**Builder in Java: Before and after**	**Builder in Java**
C++	**Builder in C++**	
PHP	**Builder in PHP**	
Delphi	**Builder in Delphi**	