Subject: Creative uses for Void*
Posted by lede on Mon, 09 Feb 2015 17:58:53 GMT
View Forum Message <> Reply to Message

Here is a C++ trick I figured out when creating a custom event class system for on of my game engines. A void pointer is very powerful in C++ because you can pass just about anything as a void pointer. In my events class I used this so that other parts of my game engine could tie into game events and when these happened connect to event to a separate part of the system that may not have been architected at the time of initial design. Not this is an excerpt from my file and is displayed for conceptual discussions.

Prototype header file

```
class CEvent
{
private:
  CEvent(void);

private:
  static CEvent m_instance;
  list<CEventNode*> m_eventList;
  unsigned int NextNodeId;

public:
  ~CEvent(void);
  static CEvent *GetInstance(void);
  bool AddItem(void func(void *, CEventArgs *e), void *obj, EventType type);
  void HandleEvent(EventType type, CEventArgs *e);
}
```

Prototype C++ code file

```
CEvent *CEvent::m_instance = NULL;

CEvent::CEvent(void)
{
  // Initial setup of private class CEvent
}

CEvent::GetInstance(void)
{
  // Check if single pattern object has been created
  if(m_instance == NULL)
  {
    m_instance = new CEvent();
  }
```

```
  return m_instance;
}

CEvent::~CEvent(void)
{
  // Check if we need to clean up our list from memory
  if(!m_eventList.empty())
  {
    while(!m_eventList.empty())
    {
      delete m_eventList.front();
      m_eventList.pop_front();
    }
  }// Endif m_eventList is not NULL
}

int CEvent::Add(void func(void *, CEventArgs *e), void *obj, EventType type)
{
  // Adding an event requires a couple of structures.  One is a structure to hold all the passed
parameters
  // The second is a structure that holds all the events.  In this design I used a ordered link list so
  // that we could use binary searches to speed up finding events when needed.
  CEventNode *node = new CEventNode(func, obj, type);
  node->Id = NextNodeId++;
  // Add new node event to list
  m_eventList.push_back(node);

  return node->Id;
}

public CEvent::HandleEvent(EventType type, CEVentArgs *e)
{
  if(m_eventList->size() > 0)
  {
    list<CEventNode>::iterator i;
 // Find events of type
    for(i=m_eventList.begin(); i != m_eventList.end(); i++)
    {
      if(*i.type == type)
      {
        // Call node function for event
        *i.Node()->func(*i.Node()->obj, e);
      }
    }
  }
}
```

This is just an example and has not been tested to see if it works. The original code we wrote used an internal linked list we developed so the code had to be modified for this demonstration. The idea here to take away is that we can use a void pointer in very creative ways to send a variety of objects in our programs.
