Subject: ConvertInt > templatable Convert<T>
Posted by kohait00 on Fri, 07 Aug 2009 14:17:57 GMT
View Forum Message <> Reply to Message

hi there

I came along to need a Min Max EditValue control, but specificable as template, so to speak.

instead of

EditInt edit, or EditDouble edit

i needed

MyValueEdit<T> edit;

In code speaking, it should be like

```
template<class T>
class MyEditMinMax : public EditMinMax<T, Convert<T>>
{};
```

instead of

```
typedef EditMinMax<T, ConvertInt> EditInt
```

so i needed kind of a Convert<T>
to make this possible.

looking at the code, i found out that the Convert Datatype is used as Base some base levels deeper, but ther then as typesafe.

it would be great to have the Convert<T> implementation. i know it would take much recoding, and the api would change, also the speed penalty for stuff like

```
if(typeid(T)) == typeid(int))
invoke some int specific convert function
```

is there any intelligent way to do that?

known issues to that:
1. Convert uses a typesafe parametrized standard constructor in ConvertInt (cause of some

constants maybe) and the like. this might need to be split, a standard constructor invoking an init function or so..

2. the converters call each another scan founction and implement their own format functions, this might need to form differently.

any comments?

Just keep in mind, what is needed is just a EditMinMax<T> or so, instead of EditMinMax<T, ConvertInt>
the class looks for a fitting converter itself, invoking its functions.

meanwhile, i try to solve it by now, with havnig stuff like that:
there, maybe you'll see and understand better the problem

```
/// an own Edit Control with min max, using Converters in a different way

template <class DataType/*, class Cv*/>
class MyEditValue : public EditField /*, public Cv*/{

protected:
 //as long as we do not have a Convert<T> need to mirror that
 Convert * pcv;

public:
 MyEditValue& operator=(const DataType& t)  { SetData(t); return *this; }
 operator DataType() const                  { return GetData(); }

 MyEditValue()
 {
  pcv = NULL;
  if(  (typeid(DataType) == typeid(int))
    || (typeid(DataType) == typeid(unsigned int))
    || (typeid(DataType) == typeid(short))
    || (typeid(DataType) == typeid(unsigned short))
    || (typeid(DataType) == typeid(char))
    || (typeid(DataType) == typeid(unsigned char))
   )
  {
   pcv = new ConvertInt();
  }
  else if( (typeid(DataType) == typeid(double))
    || (typeid(DataType) == typeid(float))
   )
  {
   pcv = new ConvertDouble();
  }
```

```
 else if( (typeid(DataType) == typeid(long))
   || (typeid(DataType) == typeid(unsigned long))
  )
 {
  pcv = new ConvertInt64();
 }
 else
 {
  D6("MyEditValue","unknown type : " << typeid(DataType).name());
  D6("MyEditValue","not covertible: " << s);
  return;
 }

 if(pcv != NULL)
  SetConvert(*pcv); //instead *this
}

virtual ~MyEditValue()
{
 if(pcv != NULL)
  delete pcv;
 pcv = NULL;
}

DataType GetMax()
{
 if (!pcv) return (DataType)0;

 if(  (typeid(DataType) == typeid(int))
   || (typeid(DataType) == typeid(unsigned int))
   || (typeid(DataType) == typeid(short))
   || (typeid(DataType) == typeid(unsigned short))
   || (typeid(DataType) == typeid(char))
   || (typeid(DataType) == typeid(unsigned char))
  )
 {
  return ((ConvertInt*)pcv)->GetMax();
 }
 else if( (typeid(DataType) == typeid(double))
   || (typeid(DataType) == typeid(float))
  )
 {
  return ((ConvertDouble*)pcv)->GetMax();
 }
 else if( (typeid(DataType) == typeid(long))
   || (typeid(DataType) == typeid(unsigned long))
  )
 {
```

```
  return ((ConvertInt64*)pcv)->GetMax();
 }
 else
 {
  D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
  D6("MyEditMinMax","not covertible: " << s);
  return (DataType)0;
 }
}

DataType GetMin()
{
 if (!pcv) return (DataType)0;

 if(  (typeid(DataType) == typeid(int))
   || (typeid(DataType) == typeid(unsigned int))
   || (typeid(DataType) == typeid(short))
   || (typeid(DataType) == typeid(unsigned short))
   || (typeid(DataType) == typeid(char))
   || (typeid(DataType) == typeid(unsigned char))
  )
 {
  return ((ConvertInt*)pcv)->GetMin();
 }
 else if( (typeid(DataType) == typeid(double))
   || (typeid(DataType) == typeid(float))
  )
 {
  return ((ConvertDouble*)pcv)->GetMin();
 }
 else if( (typeid(DataType) == typeid(long))
   || (typeid(DataType) == typeid(unsigned long))
  )
 {
  return ((ConvertInt64*)pcv)->GetMin();
 }
 else
 {
  D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
  D6("MyEditMinMax","not covertible: " << s);
  return (DataType)0;
 }
}

};

template <class DataType/*, class Cv*/>
class MyEditMinMax : public MyEditValue<DataType/*, Cv*/> {
```

```
public:
 MyEditMinMax& operator=(const DataType& t)          { SetData(t); return *this; }

 MyEditMinMax() {}
 MyEditMinMax(DataType min, DataType max)
 {
  //for the converts a *typesafe* call is needed.
  //as long as we dont have Converter<T>::MinMax(), we need to mirror that
  //Cv::MinMax(min, max);

  if(  (typeid(DataType) == typeid(int))
    || (typeid(DataType) == typeid(unsigned int))
    || (typeid(DataType) == typeid(short))
    || (typeid(DataType) == typeid(unsigned short))
    || (typeid(DataType) == typeid(char))
    || (typeid(DataType) == typeid(unsigned char))
   )
  {
   ((ConvertInt*)MyEditValue<DataType>::pcv)->MinMax(min, max);
  }
  else if( (typeid(DataType) == typeid(double))
    || (typeid(DataType) == typeid(float))
   )
  {
   ((ConvertDouble*)MyEditValue<DataType>::pcv)->MinMax(min, max);
  }
  else if( (typeid(DataType) == typeid(long))
    || (typeid(DataType) == typeid(unsigned long))
   )
  {
   ((ConvertInt64*)MyEditValue<DataType>::pcv)->MinMax(min, max);
  }
  else
  {
   D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
   D6("MyEditMinMax","not covertible: " << s);
  }

 }

 MyEditMinMax& Min(DataType min)
 {
  //Cv::Min(min); return *this;
  if(  (typeid(DataType) == typeid(int))
    || (typeid(DataType) == typeid(unsigned int))
    || (typeid(DataType) == typeid(short))
    || (typeid(DataType) == typeid(unsigned short))
    || (typeid(DataType) == typeid(char))
```

```
   || (typeid(DataType) == typeid(unsigned char))
  )
 {
  ((ConvertInt*)MyEditValue<DataType>::pcv)->Min(min);
 }
 else if( (typeid(DataType) == typeid(double))
   || (typeid(DataType) == typeid(float))
  )
 {
  ((ConvertDouble*)MyEditValue<DataType>::pcv)->Min(min);
 }
 else if( (typeid(DataType) == typeid(long))
   || (typeid(DataType) == typeid(unsigned long))
  )
 {
  ((ConvertInt64*)MyEditValue<DataType>::pcv)->Min(min);
 }
 else
 {
  D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
  D6("MyEditMinMax","not covertible: " << s);
 }
 return *this;
}

MyEditMinMax& Max(DataType max)
{
 //Cv::Max(max); return *this;
 if(  (typeid(DataType) == typeid(int))
   || (typeid(DataType) == typeid(unsigned int))
   || (typeid(DataType) == typeid(short))
   || (typeid(DataType) == typeid(unsigned short))
   || (typeid(DataType) == typeid(char))
   || (typeid(DataType) == typeid(unsigned char))
  )
 {
  ((ConvertInt*)MyEditValue<DataType>::pcv)->Max(max);
 }
 else if( (typeid(DataType) == typeid(double))
   || (typeid(DataType) == typeid(float))
  )
 {
  ((ConvertDouble*)MyEditValue<DataType>::pcv)->Max(max);
 }
 else if( (typeid(DataType) == typeid(long))
   || (typeid(DataType) == typeid(unsigned long))
  )
 {
```

```
   ((ConvertInt64*)MyEditValue<DataType>::pcv)->Max(max);
  }
  else
  {
   D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
   D6("MyEditMinMax","not covertible: " << s);
  }
  return *this;
 }

 MyEditMinMax& NotNull(bool nn = true)
 {
  //Cv::NotNull(nn); return *this;
  if(  (typeid(DataType) == typeid(int))
    || (typeid(DataType) == typeid(unsigned int))
    || (typeid(DataType) == typeid(short))
    || (typeid(DataType) == typeid(unsigned short))
    || (typeid(DataType) == typeid(char))
    || (typeid(DataType) == typeid(unsigned char))
   )
  {
   ((ConvertInt*)MyEditValue<DataType>::pcv)->NotNull(nn);
  }
  else if( (typeid(DataType) == typeid(double))
    || (typeid(DataType) == typeid(float))
   )
  {
   ((ConvertDouble*)MyEditValue<DataType>::pcv)->NotNull(nn);
  }
  else if( (typeid(DataType) == typeid(long))
    || (typeid(DataType) == typeid(unsigned long))
   )
  {
   ((ConvertInt64*)MyEditValue<DataType>::pcv)->NotNull(nn);
  }
  else
  {
   D6("MyEditMinMax","unknown type : " << typeid(DataType).name());
   D6("MyEditMinMax","not covertible: " << s);
   return;
  }
 }
};


///
```