Subject: this one worked for me
Posted by kohait00 on Thu, 20 Aug 2009 09:48:37 GMT
View Forum Message <> Reply to Message

had to change it slightly..code is found at bottom

i havent been able to test it extensivly, but it seems to work quite well..but so far it works for int and float

so incase anyone wants to have a number EditField without specifying Type directly..

so instead of

EditInt edit;

you can use

EditMinMax<DataType, ConvertT<DataType> > edit;

or event better

template <class DataType>
EditNumber : public EditMinMax<DataType, ConvertT<DataType> > {}

EditNumber<float> edit;
EditNumber<T> edit2;

now here is the code that i could compile.
thanks for help

```
template <class DataType>
class ConvertT : public Convert
{
private:
 DataType minval;
 DataType maxval;
 bool notnull;

public:
 //ConvertT() : minval(Null), maxval(Null), notnull(false) {}
 ConvertT()
 {
  if(  (typeid(DataType) == typeid(int))
```

```
   || (typeid(DataType) == typeid(short))
   || (typeid(DataType) == typeid(char))
   || (typeid(DataType) == typeid(int64))
  )
 {
  minval = (DataType) (1<<(8*sizeof(DataType)-1)); //0x8000...
  maxval = (DataType) (minval - 1);//0x7FFF..
  notnull = true;
 }

 else if(  (typeid(DataType) == typeid(unsigned int))
   || (typeid(DataType) == typeid(unsigned short))
   || (typeid(DataType) == typeid(unsigned char))
   || (typeid(DataType) == typeid(uint64))
  )
 {
  minval = 0;
  maxval = (DataType) (-1); //FFFF...
  notnull = true;
 }

 else
 {
  minval = (DataType)0; //Null;
  maxval = (DataType)0; //Null;
  notnull = false;
 }

}

virtual Value Scan(const Value& text) const {
 if (IsNotNull() && IsNull(text)) return NotNullError();
 Value v;
 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))
  )
 {
  v = StdConvertInt().Scan(text);
 }
 else if( (typeid(DataType) == typeid(double))
   || (typeid(DataType) == typeid(float))
  )
 {
  v = StdConvertDouble().Scan(text);
```

```
  }
  else if( (typeid(DataType) == typeid(uint64))
    || (typeid(DataType) == typeid(int64))
   )
  {
   v = ConvertInt64().Scan(text); // No StdConvert64
  }
  else
   v = StdConvert().Scan(text);

  if ( !IsNull(v)
   //&&  !IsNull(minval)
   //&&  !IsNull(maxval)
   && IsNotNull()
   )
  {
   DataType m;

   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))
    )
   {
    m = (DataType)(int)(v);
   }
   else if( (typeid(DataType) == typeid(double))
     || (typeid(DataType) == typeid(float))
    )
   {
    m = (DataType)(double)(v);
   }
   else if( (typeid(DataType) == typeid(uint64))
     || (typeid(DataType) == typeid(int64))
    )
   {
    m = (DataType)(int64)(v);
   }
   else
   {
    return ErrorValue(UPP::Format(t_("Number must be a valid type. %s"),
Format((DataType)(0)))); //gives compile errors if no second format
   }


   if(m >= minval && m <= maxval)
```

```
    return v;
   return ErrorValue(UPP::Format(t_("Number must be between %s and %s."), Format(minval),
Format(maxval)));
  }
  return v;
 }
 virtual Value Format(const Value& q) const
 {
  // You may want to check the type of q here to ensure it matches DataType
  return StdConvert().Format(q);
 }

 ConvertT<DataType>& MinMax(DataType _min, DataType _max) { minval = _min; maxval =
_max; return *this; }
 ConvertT<DataType>& Min(DataType _min)              { minval = _min; return *this; }
 ConvertT<DataType>& Max(DataType _max)              { maxval = _max; return *this; }
 ConvertT<DataType>& NotNull(bool b = true)          { notnull = b; return *this; }
 ConvertT<DataType>& NoNotNull()                     { return NotNull(false); }
 DataType        GetMin() const                  { return minval; }
 DataType        GetMax() const                  { return maxval; }
 bool        IsNotNull() const               { return notnull; }
};
```