Subject: Added partial support for ODBC output parameters, but the solution is ugly
Posted by jjacksonRIAB on Sun, 11 Mar 2012 18:04:47 GMT
View Forum Message <> Reply to Message

The problem with output parameters is that they were being bound to a copy of a Value instead of a reference to that Value.

In ODBC.cpp I added:

```
struct Param {
    int    ctype;
    int    sqltype;
    String data;
    SQLLEN li;
    int    direction; // parameter direction INPUT/OUTPUT/BOTH
    Ref    outAddr;   // address of Value type
};
```

and in SetParam I set all directions to be SQL_PARAM_INPUT by default, so they will not change the way things work now.

```
void ODBCConnection::SetParam(int i, const Value& val)
{
 Param& p = param.At(i);

 Value r;

 if(val.Is<Ref>()) {
  Ref reference = ValueTo<Ref>(val);
  p.outAddr = reference;
  r = reference.GetValue();
 }
 else {
  r = val;
 }

 if(IsNumber(r)) {

  if(r.Is<int64>()) {
   int64 x = r;
   p.ctype = SQL_C_SBIGINT;
   p.sqltype = SQL_BIGINT;
   p.data = String((char *)&x, sizeof(x));
   p.li = sizeof(x);
   p.direction = SQL_PARAM_INPUT;
  }
  else {
```

```
   double x = r;
   p.ctype = SQL_C_DOUBLE;
   p.sqltype = SQL_DOUBLE;
   p.data = String((char *)&x, sizeof(x));
   p.li = sizeof(x);
   p.direction = SQL_PARAM_INPUT;
  }
 }
 if(IsString(r)) {
  p.ctype = SQL_C_CHAR;
  p.sqltype = SQL_LONGVARCHAR;
  p.data = r;
  p.li = p.data.GetLength();
  p.direction = SQL_PARAM_INPUT;
 }
 if(IsDateTime(r)) {
  p.ctype = SQL_C_TYPE_TIMESTAMP;
  p.sqltype = SQL_TYPE_TIMESTAMP;
  Time t = r;
  SQL_TIMESTAMP_STRUCT tm;
  tm.year = t.year;
  tm.month = t.month;
  tm.day = t.day;
  tm.hour = t.hour;
  tm.minute = t.minute;
  tm.second = t.second;
  tm.fraction = 0;
  p.data = String((char *)&tm, sizeof(tm));
  p.li = sizeof(tm);
  p.direction = SQL_PARAM_INPUT;
 }
 if(IsNull(r))
  p.li = SQL_NULL_DATA;

 if(val.Is<Ref>())
 {
  p.direction = SQL_PARAM_INPUT_OUTPUT;
 }
}
```

I added a call to .Is<Ref> so if a reference is passed it will set the direction flag to input/output.

In Execute method I check for bound parameter type:

```
if(p.direction == SQL_PARAM_OUTPUT || p.direction == SQL_PARAM_INPUT_OUTPUT)
{
    if(!IsOk(SQLBindParameter(session->hstmt, i + 1, p.direction, p.ctype, DataType,
            ParameterSize, DecimalDigits, p.outAddr.GetVoidPtr(), p.data.GetLength(),
```

```
      &p.li)))
        return false;
}
else
{
   if(!IsOk(SQLBindParameter(session->hstmt, i + 1, p.direction, p.ctype, DataType,
       ParameterSize, DecimalDigits, (SQLPOINTER)~p.data, p.data.GetLength(),&p.li)))
   return false;
}
```

In sqls.h I added this:

```
#define OUT_PARAM(I) RawToValue(Ref(I))
```

So now when I call the stored procedure:

```
int retVal;

SQL.Execute (
   "{? = call TestProcedure (?,?)}",
   OUT_PARAM(retVal),
   someVal,
   OUT_PARAM(someOtherVal)
   );
```

The value is returned properly. The problem with this approach is I don't like having to use void pointer, or having to use a macro to wrap a Ref that goes into a Value type.

The main problem seems to be that SetParam makes a copy of whatever data is passed to it and operates on the copy. It might work better if the actual data is operated on or the copy is modified by the database driver and copied back to the original.

I say the solution is partial because I have not tested it with stored procedure that returns a table, or anything other than integer return values. I also thought about const vs. non-const to determine input param vs input/output param but I don't know if this can be implemented.

Thoughts?