Working with dataform1detailblock

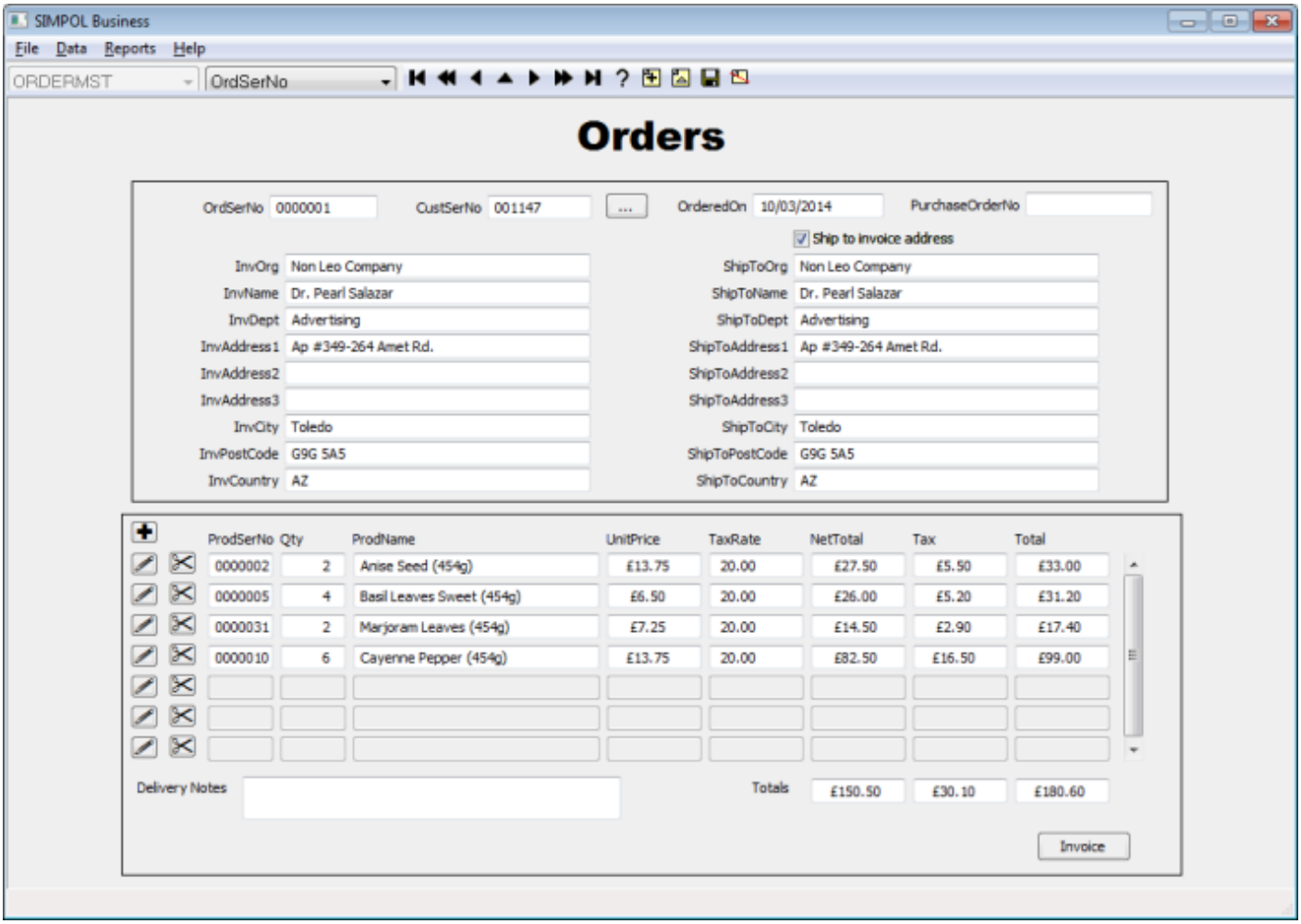
About the Design of Detail Blocks

In our initial design and implementation of detail blocks in SIMPOL we recognized that in the original version in Superbase some things had not gone well. Although it was possible to add records as long as there were less records than visible rows, once the visible rows were filled adding records became difficult. There was also no support for deleting records from the detail block. The ability to nest them up to 8 levels deep was troublesome when trying to use them for reliable data-entry.
As a result many Superbase users have been forced to come up with their own solutions to these problems over the years. The solutions usually were either to use a dialog for adding and editing data, or to add a special set of controls on the form where data was created or modified. Most solutions also added a set of buttons to the left of the rows to edit or delete the row data.

During our initial design and implementation for SIMPOL, we decided to make the detail block read-only to avoid the need to wrestle with these issues since, to start with, we just wanted to get a reliably working read-only implementation. There are numerous issues to resolve with something like this if it is allowed to be read/write. We decided to add the ability to modify the content of the detail block under program control. The early versions had some limited ability to do this, but with the 1.8 release, we decided to commit to a detail block that could be completely managed under program control. We added the necessary methods to add, edit, and delete entries.

Adding New Records to Detail Blocks

For this project we chose to use the dialog method for data-entry. New records are created using the same dialog window and form (which is a normal `dataform1` form). The only difference is the record is a new one rather than an existing one. In this design, when the order record is created, we disable the buttons that allow the creation, editing, or deletion of detail block records. After it has been saved, these buttons are again enabled. This ensures that the detail block records are created based on an existing order record. Otherwise we would have had the problem of ensuring the records are not saved until the order itself is saved. Below is an image of the orders form.



The Orders Form from SIMPOL Business.

The key bit of code for both adding and editing records in the detail block can be found in two of the included functions: `addorderline()` and `editorderline()`. They both call the function `doeditaddorderline()` to actually present the dialog box and handle the interaction with the user. At the end that function returns a `type(db1record)` object and sets the boolean saved variable to `.true` if the user saved the record. Below is the `addorderline()` function.

Example 7.1. The addorderline() function of the SIMPOL Business program

```
function addorderline(dataform1button me, appwindow appw)
  boolean saved
  type(db1record) r
  dataform1record rec
  dataform1table table
  integer ordserno, e
  dataform1detailblock dtb
  dataform1recordset rset
```

```
  saved = .false
  e = 0
  ordserno = me.form.masterrecord.record!OrdSerNo
  r =@ doeditaddorderline(appw, .true, saved=saved, \
                          ordserno=ordserno)
  if saved
    // Here we need to add the row to the detail block and move
    // the current row pointer
    dtb =@ me.form!dtbOrderLines
    if dtb !@= .nul
      rset =@ dataform1recordset.new()
      table =@ me.form.findtable(r.table.tablename)
      rec =@ dataform1record.new(r, table, error=e)
      // Here we are placing the record in the record set as the
      // master record
      rset.records[1] =@ rec
      dtb.addrowdata(rset, error=e)
      calculateordertotals(me.form, dtb, appw)
    end if
  end if
end function
```

The important point of this is the place in the code where the record is added to the detail block.

> Note
> A detail block row is represented by a `dataform1recordset` object. This contains a records property of type array. Each element in the records array is of type `dataform1record`. If the detail block contains multiple linked records that are linked 1:1 (for example a product name that is not stored in the detail record but which is only looked up via the product ID), then for each linked table there will be an additional `dataform1record` object in the records array.

In the example above, we are working with a simple detail block consisting of only the detail table record in each row. To write the new record to the detail block, we create a new record set, create a new `dataform1record` object using the record that was returned, and then assign the `dataform1record` object to the first element of the record set's records array. Once our preparation is complete, we call the `addrowdata()` method of the `dataform1detailblock` object. Also, since we are managing the totals of several of the columns in the ORDERMST record, we also call the `calculateordertotals()` function that calculates the totals to update the values in that record and to show them on the form.

Editing Records in a Detail Block

The code that handles the editing of the detail block is very similar to that which adds a new record:

Example 7.2. The addorderline() function of the SIMPOL Business program

```
function editorderline(dataform1button me, appwindow appw)
  integer row, e, orditemno
  dataform1detailblock dtb
  dataform1recordset rset
  dataform1record rec
  type(db1record) r
  boolean saved
  sbapplication app

  saved = .false
  app =@ appw.app
  e = 0
  dtb =@ me.form!dtbOrderLines
  if dtb !@= .nul
    row = .toval(me.name, nondigits(me.name), 10)
    if row >= 1 and row <= dtb.rows
      rset =@ dtb.getrowdata(row, error=e)
      if rset =@= .nul
        wxmessagedialog(appw.w, "Error no row data available", \
                        sAPPMSGTITLE, "ok", "error")
      else
        rec =@ rset.records[1]
        if rec =@= .nul or rec.record =@= .nul
          wxmessagedialog(appw.w, "Error record not found in the \
                          row data", sAPPMSGTITLE, "ok", "error")
        else
          r =@ rec.record
          orditemno = r!OrdItemNo
          r =@ .nul
          r =@ doeditaddorderline(appw, .false, orditemno, saved)
          if saved
```

```
            // Update the specific row in the detail block
            rec.record =@ r
            dtb.setrowdata(row, rset, error=e)
            calculateordertotals(me.form, dtb, appw)
          end if
        end if
      end if
    end if
  end if
end function
```

In the preceding example the name of the button control contains the row number and that is retrieved by using the `.toval()` function and by declaring all of the non-digit content using the `nondigits()` function. Then we call the `getrowdata()` method of the detail block passing in the row number to retrieve the record set representing that row. We access the `dataform1record` that contains the detail block record from the record set and use that to read our unique record ID, the value of the `OrdItemNo` field.
We then clear the record variable by setting it to `.nul` and call the `doeditaddorderline()` function passing in the `orditemno` variable. If the user has saved the changes to the record, then we need to replace the old version of the record in our `dataform1record` object with the updated version. Then all that is left is to call the `setrowdata()` method of the detail block and as with the new record, we need to call the `calculateordertotals()` function to update the totals in the master record and display them on the screen.

Deleting Records in a Detail Block

All that remains with our detail block is to be able to delete records from it. The program code that does that is very similar to that used for editing:

Example 7.3. The addorderline() function of the SIMPOL Business program

```
function deleteorderline(dataform1button me, appwindow appw)
  integer row, e
  dataform1detailblock dtb
  dataform1recordset rset
  dataform1record rec

  e = 0
  dtb =@ me.form!dtbOrderLines
  if dtb !@= .nul
```

```
    row = .toval(me.name, nondigits(me.name), 10)
    if row >= 1 and row <= dtb.rows
      rset =@ dtb.getrowdata(row, error=e)
      if rset =@= .nul
        wxmessagedialog(appw.w, "Error no row data available", \
                        sAPPMSGTITLE, "ok", "error")
      else
        rec =@ rset.records[1]
        if rec =@= .nul or rec.record =@= .nul
          // wxmessagedialog(appw.w, "Error record not found in \
          //                 the row data", sAPPMSGTITLE, "ok", \
          //                 "error")
        else
          rec.lock(error=e)
          if e != 0
            wxmessagedialog(appw.w, "Error locking the record", \
                            sAPPMSGTITLE, "ok", "error")
          else
            rec.delete(error=e)
            if e != 0
              wxmessagedialog(appw.w, "Error deleting the
record",\
                              sAPPMSGTITLE, "ok", "error")
            else
              dtb.removerowdata(row, error=e)
              calculateordertotals(me.form, dtb, appw)
            end if
          end if
        end if
      end if
    end if
  end if
end function
```

Just as was done in the edit code, first we transform the control name into the row number and then we use that to retrieve the record set representing that row. After extracting the `dataform1record` object from the record set, we lock it and assuming that succeeded, we call the `delete()` method of the `dataform1record` object. If that also succeeds (it should), the `removerowdata()` method of the detail block object is called to remove the actual record set for that row and to adjust the scroll position of the visible rows on the form.

Revision #6
Created 15 November 2021 18:50:42 by Eva Crawford-McKee
Updated 27 December 2021 21:37:08 by Nikolaus Zolnhofer