Currently, we hand-record all the serial numbers embossed onto badges into logbooks, and then use those logbooks to try to prevent future serial numbers from being duplicated. It's a very manual and tedious process, full of opportunities for errors.

So, using C# and a local Access database (though this can be easily changed to a more complex database hosted on a server), I've been working on a Windows program that would replace the logbooks and instead record all serial numbers into the database, using SQL statements to ensure that there will be no duplications.

I've also added some reporting/look-up functions to the program, and I have to say, I think this project has come together great!

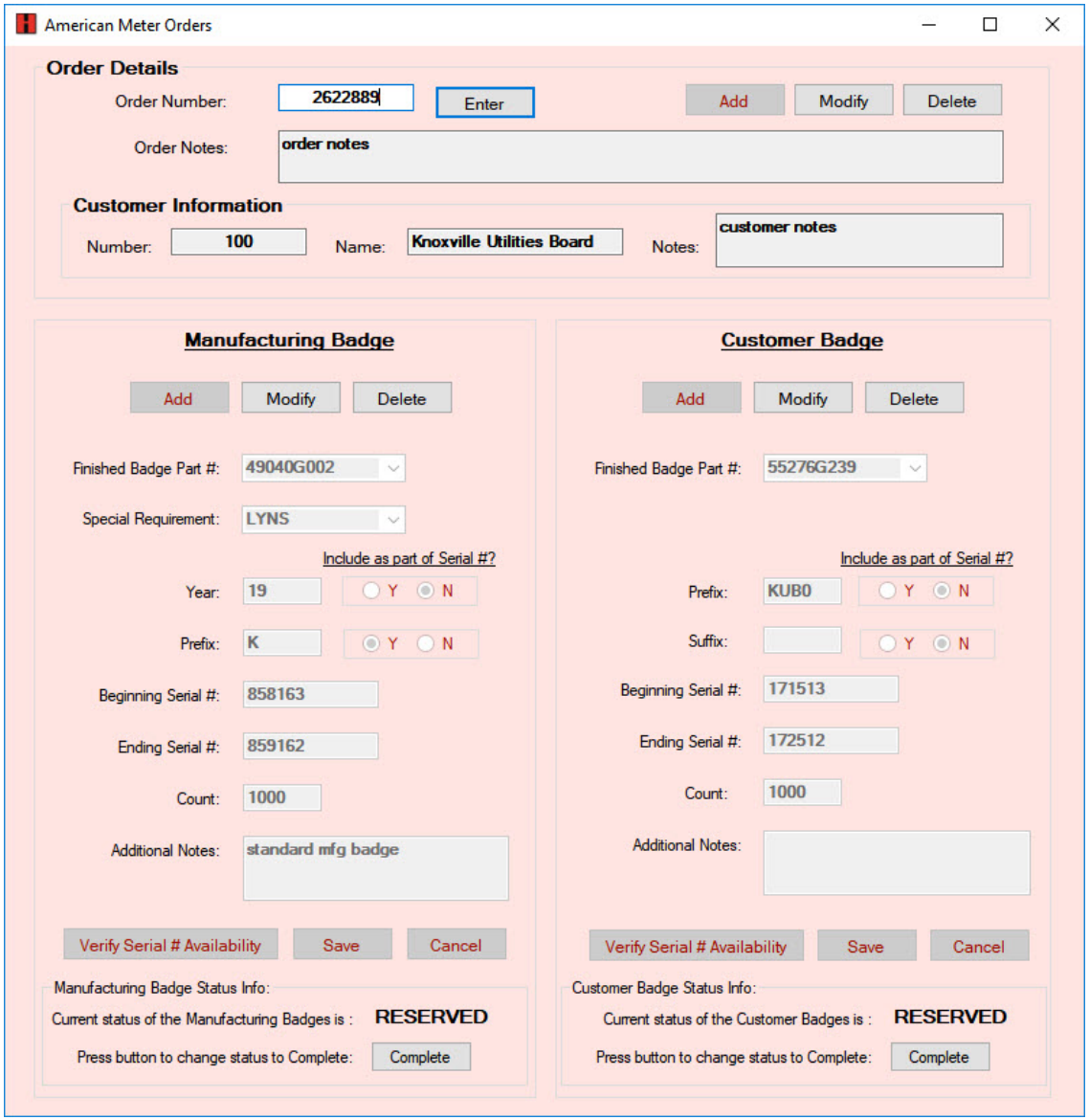
Some screen shots:

This is the Welcome window when the application first starts, self-explanatory. Oh, good time to mention that I'm colorblind. I was aiming for a kind of Honeywell color scheme layout – might be too pinkish…

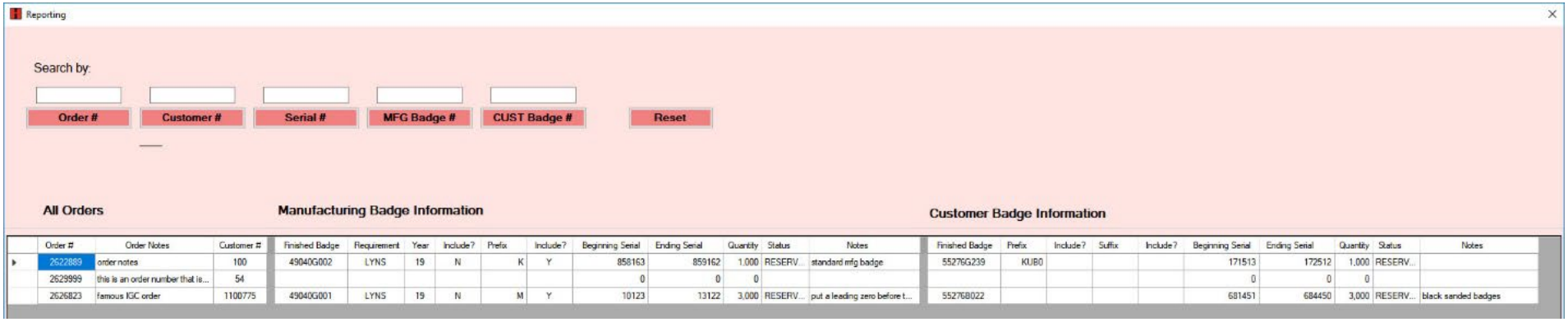
If the user selects the first option, Order / Serial Number Entry, they'll get to the workhorse of the program, which has a lot going on.

The order number in this example has already been added to the database, so the information in the fields is being pulled from a couple different tables. Buttons and fields are made available to use depending on varying criteria. The drop-down fields are linked to database tables so users can only select applicable options, and there are also validation methods to ensure that the information entered for the other fields is correct.

Had this order number not already been in the database, the user would have been prompted to enter the basic order details, then the Manufacturing and Customer badge fields would become available.



Back to the Welcome window - if the user selects the Review / Reporting button, they'd see:

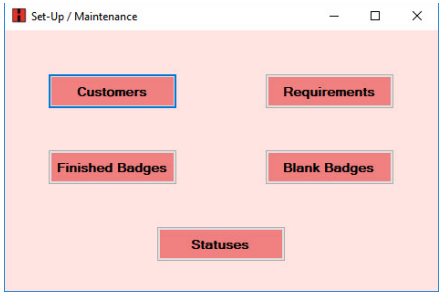


Order #	Order Notes	Customer #	Finished Badge	Requirement	Year	Include?	Prefix	Include?	Beginning Serial	Ending Serial	Quantity	Status	Notes	Finished Badge	Prefix	Include?	Suffix	Include?	Beginning Serial	Ending Serial	Quantity	Status	Notes
2622889	order notes	100	49040G002	LYNS	19	N	K	Y	858163	859162	1,000	RESERV...	standard mfg badge	55276G239	KUB0				171513	172512	1,000	RESERV...	
2629999	this is an order number that is...	54							0	0	0								0	0	0		
2626823	famous IGC order	1100775	49040G001	LYNS	19	N	M	Y	10123	13122	3,000	RESERV...	put a leading zero before t...	552768022					681451	684450	3,000	RESERV...	black sanded badges

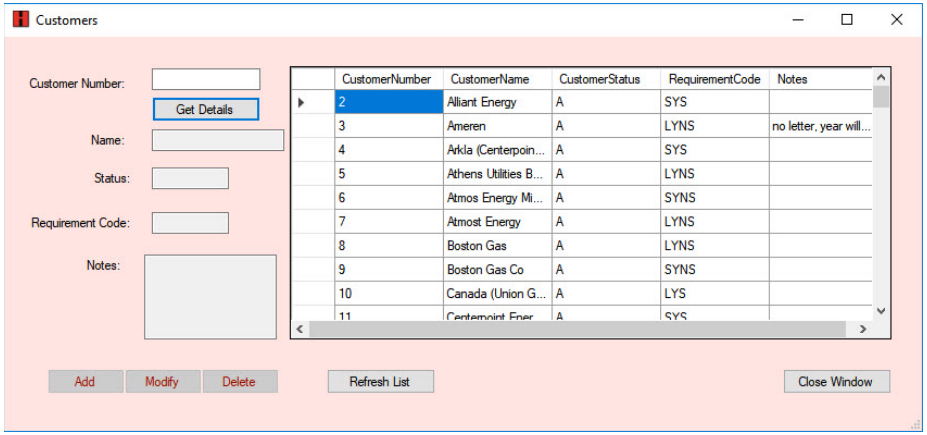
(I have just a few sample orders entered)

This Windows form works great! We (the badge station workers) currently have no way to look up anything by customer number, and to find anything by order number or serial number details thumbing through the pages and pages of different logbooks.

Again, back to the Welcome window - if the user selects Set-Up / Maintenance, they'd see:



And if they selected Customers from that window, they'd see:



This window shows all the current customers and offers the ability to add or modify. The delete key only changes the customer status to inactive, thus saving all history. The other windows from the set-up screens run mostly the same.

Some things that I haven't built in yet are:

- Multiple users with passwords, and security levels for those users
- Date/time stamps for changes/entries to the database
- We have two customers that require a 3rd badge (sticker, actually) with serial numbers, but I believe those could be incorporated into the next point –
- Camcodes. The database could be expanded to track the badges that come from an outside source. This would certainly help our Camcode buyer, who occasionally must come out and dig through our badge drawers to physically see what serial numbers we have stored (the logbooks are confusing).
- Some customer badges don't have serial numbers (name + other info only).
- Some customers require a 4-digit year on their Manufacturing badges, vs. a 2-digit year I've added some example of the code at the end….

I believe I've utilized Object Oriented Programming practices to ensure security for the program and have created the database and its update statements as to prevent SQL Injection.

For customer badges, to check if only verifying the serial numbers are not duplicated:

```
public static Check4Duplicates CustVerifySerialOnly(int beginning, int ending, int orderNumber, string custNumber)
        {
            OleDbConnection connection = SerialsDatabaseDB.GetConnection();

            string checkStatement
                = "SELECT OrderNumber, CustBeginningSerial, CustEndingSerial, CustomerNumber "
                + "FROM Orders "
                + "WHERE ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND (CustBeginningSerial Between @beginning AND @ending)) "
                + "OR ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND (CustEndingSerial Between @beginning AND @ending)) "
                + "OR ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND ((CustBeginningSerial < @beginning) AND (CustEndingSerial > @ending)))";

            OleDbCommand selectCommand = new OleDbCommand(checkStatement, connection);
            selectCommand.Parameters.AddWithValue("@orderNumber", orderNumber);
            selectCommand.Parameters.AddWithValue("@custNumber", custNumber);
            selectCommand.Parameters.AddWithValue("@beginning", beginning);
            selectCommand.Parameters.AddWithValue("@ending", ending);

            try
            {
                connection.Open();
                OleDbDataReader dataReader = selectCommand.ExecuteReader();
                if (dataReader.Read())
                {
                    Check4Duplicates duplicatesYes = new Check4Duplicates
                    {
                        DuplicatesYN = true,
                        OrderNumber = dataReader["OrderNumber"].ToString(),
                        CustNumber = dataReader["CustomerNumber"].ToString(),
                        Beginning = dataReader["CustBeginningSerial"].ToString(),
                        Ending = dataReader["CustEndingSerial"].ToString()
                    };
                    return duplicatesYes;
                }
                else
                {
                    Check4Duplicates duplicatesNo = new Check4Duplicates
                    {
                        DuplicatesYN = false
                    };
                    return duplicatesNo;
                }
            }
            catch (OleDbException ex)
            {
                throw ex;
            }
            finally
            {
                connection.Close();
            }
        }
```

Then, depending if customer badge Prefixes should be included when checking for duplicated serial numbers, the SQL statement is:

```
           string checkStatement
                = "SELECT OrderNumber, CustBeginningSerial, CustEndingSerial, CustPrefix, CustomerNumber "
                + "FROM Orders "
                + "WHERE ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND (CustPrefix = @prefix) AND (CustBeginningSerial Between @beginning AND @ending)) "
                + "OR ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND (CustPrefix = @prefix) AND (CustEndingSerial Between @beginning AND @ending)) "
                + "OR ((OrderNumber <> @orderNumber) AND (CustomerNumber = @custNumber) AND (CustPrefix = @prefix) AND ((CustBeginningSerial < @beginning) AND (CustEndingSerial >
@ending)))";
```

And there are a few more based on what should be included when checking for duplicates.

Lastly, one more example… When the user clicks on the Enter button after inputting an order number on the Order / Serial Number field, several options…

```
private void EnterOrderNumber()
        {
            string orderNumber = txtOrderNumber.Text;

            this.GetOrderDetails(orderNumber);

            if (order == null)
            {
                btnAddOrder.Enabled = true;
            }
            else
            {
                string customerNumber = order.CustomerNumber;
                this.GetCustomerDetails(customerNumber);

                btnModifyOrder.Enabled = true;
                btnDeleteOrder.Enabled = true;
                txtCustNumber.Text = order.CustomerNumber;
                txtCustName.Text = customer.CustomerName;
                txtOrderNotes.Text = order.OrderNotes;
                txtCustNotes.Text = customer.CustomerNotes;

                if (order.MfgQuantity > 0)
                {
                    cboMfgFinished.Text = order.MfgFinishedBadge;
                    cboMfgRequirement.Text = order.MfgRequirementCode;
                    txtMfgYear.Text = order.MfgYear;
                    if (order.MfgYearInclude == "Y")
                    {
                        rbMfgYearYes.Checked = true;
                    }
                    else
                    {
                        rbMfgYearNo.Checked = true;
                    }
                    txtMfgPrefix.Text = order.MfgPrefix;
                    if (order.MfgPrefixInclude == "Y")
                    {
                        rbMfgPrefixYes.Checked = true;
                    }
                    else
                    {
                        rbMfgPrefixNo.Checked = true;
                    }
                    txtMfgBeginning.Text = order.MfgBeginningSerial.ToString();
                    txtMfgEnding.Text = order.MfgEndingSerial.ToString();
                    txtMfgCount.Text = order.MfgQuantity.ToString();
                    txtMfgBadgeNotes.Text = order.MfgNotes;

                    string statusMfg = order.MfgStatus;
                    lblMfgStatusIntro.Visible = true;
                    lblMfgStatus.Visible = true;
                    lblMfgStatus.Text = statusMfg;
                    if (statusMfg == "RESERVED" || statusMfg == "Reserved")
                    {
                        lblMfgStatusChange.Visible = true;
                        btnMfgStatusChange.Visible = true;
                    }
```

```
            btnAddMfgBadges.Enabled = false;
            btnModifyMfgBadges.Enabled = true;
            btnDeleteMfgBadges.Enabled = true;
        }
        else
        {
            this.ClearMfgBadgeData();
            btnAddMfgBadges.Enabled = true;
            btnModifyMfgBadges.Enabled = false;
            btnDeleteMfgBadges.Enabled = false;
        }

        if (order.CustQuantity > 0)
        {
            cboCustFinished.Text = order.CustFinishedBadge;
            txtCustPrefix.Text = order.CustPrefix;
            if (order.CustPrefixInclude == "Y")
            {
                rbCustPrefixYes.Checked = true;
            }
            else
            {
                rbCustPrefixNo.Checked = true;
            }
            txtCustSuffix.Text = order.CustSuffix;
            if (order.CustSuffixInclude == "Y")
            {
                rbCustPrefixYes.Checked = true;
            }
            else
            {
                rbCustSuffixNo.Checked = true;
            }
            txtCustBeginning.Text = order.CustBeginningSerial.ToString();
            txtCustEnding.Text = order.CustEndingSerial.ToString();
            txtCustCount.Text = order.CustQuantity.ToString();
            txtCustBadgeNotes.Text = order.CustNotes;

            string statusCust = order.CustStatus;
            lblCustStatusIntro.Visible = true;
            lblCustStatus.Visible = true;
            lblCustStatus.Text = statusCust;
            if (statusCust == "RESERVED" || statusCust == "Reserved")
            {
                lblCustStatusChange.Visible = true;
                btnCustStatusChange.Visible = true;
            }

            btnAddCustBadges.Enabled = false;
            btnModifyCustBadges.Enabled = true;
            btnDeleteCustBadges.Enabled = true;
        }
        else
        {
            this.ClearCustBadgeData();
            btnAddCustBadges.Enabled = true;
            btnModifyCustBadges.Enabled = false;
            btnDeleteCustBadges.Enabled = false;
        }
    }

}
```