Visual Element Migrator

Release 2.0.0.2

2013-11-18

Features

Release 2.0.0.2 adds the following capabilities to the Migrator and various cleanup / prettification enhancements as noted.

1. Download Inspection notes along with element condition data using Migrator

    - Optional behavior, use the **Notes?** checkbox on the CoRe Elements data page (*bottom left next to Download button*)
    - Notes are downloaded along with the element condition data in a separate Xml element.
      ***The notes are encoded for Xml meaning that special characters are 'escaped' such as the greater-than symbol. When 'read' from Xml using a XmlReader, the notes should get restored to original text characters.*
    - Along with the notes, pointers to the source P4.x ELEMINSP table fields ELMROWIDKEY and DOCREFKEY are also captured.
    - *It is highly recommended that agencies update their DOCREFKEY field contents to a random string such as a GUID and ensure that all rows have unique DOCREFKEY values. The Pontis 4.x application takes care of ELMROWIDKEY so that column's value should* ***not*** *be touched.*

2. From a set of migrated data generated from a set of downloaded data containing notes, the software will generate a SQL update script that can be run against a target 5.1.3 or later database that contains the table PON_ELEM_INSP

    - An **Upload Notes** button is on the AASHTO tab (3rd tab) at the bottom.
    - The target database table must have been quantified from a Migrator file already i.e. the import must have been run so that the downloaded notes can match to their original element row

3. Element data grids now show cell quantities and percentages as individual columns.

    - The percentages are calculated from the incoming data
    - The columns can be sorted and filtered.

4. Capability to run simple validity checks on the source data

    - Optional behavior, use the **Notes?** checkbox on the CoRe Elements data page (*bottom right next to the page size setting*)

- Match Granularity, set the decimal places of precision for checking whether data is 'not equal' from 0 to 6 decimal places to the right of the decimal point
- Rows with errors will have the ERRORS column cell highlighted in Salmon. An example is attached as a PNG.
- The validity checks are on the **incoming quantity condition state distributions**, compared with the **incoming quantity** from the rows in the data source whether file or download.
- **Rounding affects the process**. Setting a low precision will reduce the reporting of spurious/extraneous errors, while a higher precision will generate more suspects.

Based on the source data, the following checks are performed:

Basic Validity Checks

Quantities

- Do state quantities add up to the stated total quantity?
- Is any state quantity less than 0?
- Is the total quantity less than or equal to 0?
- Is any state quantity greater than the total quantity?
- Shows amount of any discrepancies (plus or minus)

Percentages

- Do state percentages, calculated by state quantity over total quantity, add up to 100%?
- Do state percentages, calculated by state quantity over **calculated** (*summated*) quantity, add up to 100%?
- Do the results of the above two checks equal each other?
- Shows amount of any percentage discrepancies(plus or minus)

Display of Error

A typical error string in the grid looks like:

QErrs: SumCSQty <> TotQty - sum: 197.511 qty: 197.51 diff: 0.001

which is interpreted as:
"...the sum of the condition state quantities from the incoming data did not match the incoming total quantity"

In this case, precision was set to 3 decimal places, so a tiny sliver error of .001 was calculated and reported. Why worry?

NB: These error strings may get long if there are multiple problems with a particular row of data. This may require resizing the column to see the entire message.

Limitations

- The checker does not evaluate the situations where data is in an invalid state, such as State 4 for a 3 state element.
- The AASHO output data is not checked other than to the extent the Migrator already reports errors in the processing log
- The use of equality etc. can show tiny sliver errors that are trivial.

Usage

All these errors are concatenated (joined together) into a string as the data is loaded from an Xml file or downloaded, and become a new synthetic data column on the grid titled **ERRORS**. Since only some records will have errors, hopefully very few, it is helpful to **filter the grid** on the ERRORS column. The grid filter tool will show all the errors, which you could pick from to filter, but the easiest thing to do is to set a filter condition using the logical setting **IsGreaterThan** with a value of two double-quotes e.g. "". This will filter the list to any rows that have error text. You can also **sort** on the ERRORS column to get the problem rows into view.

Source Code

Here is the actual method from the application that evaluates incoming data for validity for review (some comments were removed). This method works on an incoming row of data containing a **total quantity** and a **quantity condition state distribution**

```
using AASHTO.ElementMigrationCommon;
using System;
using System.Text;
using AASHTO.ElementMigrationTools;
using AASHTO.ElementMigration.VisualElementMigrator2.Model;
using AASHTO.ElementMigration.VisualElementMigrator2.Properties;

namespace AASHTO.ElementMigration.VisualElementMigrator2.Extensions
{
    /// <summary>
    ///   Calculates and reports any data errors in incoming element data.  Not
all data errors are trapped by this. Primarily to alert user to potential
problems
    /// </summary>
    public static class ElementDataValidationExtensions
    {
        /// <summary>
        ///  Identify all errors in an element data row (CoRe element assumed)
        /// </summary>
        /// <returns></returns>
        public static String EvalElementErrors(this CoReElement sourceElem)
        {
            String errorString = null;
            var qErr = String.Empty;
            var pErr = String.Empty;

            // decimal places for rounding...
            int decPlaces =
Math.Max(Settings.Default.ElementDataErrorCheckingRoundingPlaces, 2);  //
Default is 2 decimal places of precision.

            try
            {
                if (sourceElem != null)
                {

                    qErr = ReportQuantityErrors(sourceElem.QState,
sourceElem.Quantity, decPlaces);

                    if (sourceElem.Quantity > 0)
                    {
                        pErr = ReportPercentageErrors(sourceElem.QState,
sourceElem.Quantity, decPlaces);
                    }
                    else
                    {
                        pErr = "Raw quantity <= 0 - percentages cannot be
calculated.";
                    }
                    if ((!String.IsNullOrEmpty(qErr)) ||
(!String.IsNullOrEmpty(pErr)))
                    {
                        errorString =
                            (!String.IsNullOrEmpty(qErr) ? "QErrs: " + qErr :
String.Empty) +
                            ((!String.IsNullOrEmpty(qErr)) &&
(!String.IsNullOrEmpty(pErr)) ? " | " : String.Empty) +
                            (!String.IsNullOrEmpty(pErr) ? "PErrs: " + pErr :
String.Empty);

                        sourceElem.ElementErrorSummary = errorString; //
popluate the object field

                    }
                }
                else // this should never be!
                    throw new ArgumentNullException("Incoming CoRe element data
row object is null!");

            }
            catch (Exception ex)
            {
                ex.PreserveExceptionDetail();
                throw;
            }
            finally
            {

            }
            return errorString;
        }

        /// <summary>
        /// Extension Method to evaluate incoming data calculate percentages
and determine if sums or state percentages vary from expected or 100% etc.
        /// </summary>
        /// <param name="pState"></param>
        /// <param name="quantity"></param>
        /// <param name="decplcs"></param>
        /// <returns></returns>
        public static String ReportPercentageErrors(Double[] pState, Double
quantity, int decplcs)
        {

            var errSB = new StringBuilder();
            Double sumPCalc = 0;
            Double sumPFromQty = 0;
            Double sumQ = 0;

            Int32 i = 0;

            try
            {
                if (quantity <= 0)
                {
                    throw new ArgumentException(String.Format("Quantity is <= 0
{0} - argument error", quantity));
                }
                else
                {
                    foreach (Double p in pState)
                    {
                        i++;

                        if (p < 0) // 0 is ok.
                            errSB.AppendFormat((errSB.Length > 0 ? " | " : "")
+ "P CS{0} < 0 - {1} ", i, p);

                        sumQ = sumQ + p;
                    }



                    Double pCalc = 0.0;
                    Double pQty = 0.0;
                    if (sumQ > 0.0)
                    {
                        i = 0;

                        foreach (double p in pState)
                        {
                            i++;
                            // sum ercentage based on calculated total quantity
                            pCalc = Math.Round(100.0 * (p / sumQ), decplcs);
                            sumPCalc = sumPCalc + pCalc;

                            pQty = Math.Round(100.0 * (p / quantity), decplcs);
                            // calc sum percentage based on incoming quantity
                            sumPFromQty = sumPFromQty + pQty;

                            if (pCalc > 100.0)
                                errSB.AppendFormat((errSB.Length > 0 ? " | " :
"") + "CS{0} pctQtySum {1} > 100.0 ", i, pCalc);

                            if (pCalc > 100.0)
                                errSB.AppendFormat((errSB.Length > 0 ? " | " :
"") + "CS{0} pctTotQty {1} > 100.0 ", i, pQty);

                            if (pCalc < 0.0)
                                errSB.AppendFormat((errSB.Length > 0 ? " | " :
"") + "CS{0} pctQtySum {1}< 0.0 ", i, pCalc);

                            if (pCalc < 0.0)
                                errSB.AppendFormat((errSB.Length > 0 ? " | " :
"") + "CS{0} pctTotQty {1} <0.0 ", i, pQty);

                            // see if state calculated percentages vary from
each other significantly.
```

```
                            if (!Math.Equals(Math.Round(Math.Abs(pQty - pCalc),
decplcs), 0.0))
                                errSB.AppendFormat((errSB.Length > 0 ? " | " :
"") + "pctQtySum {0} <> pctTotQty {1} - diff: {2}", pCalc, pQty,
Math.Round(pQty - pCalc, decplcs).ToString("N" + (decplcs + 2).ToString()));
                        }
                    }

                    if (!Math.Round(sumPCalc, decplcs).Equals(100.0))
                        errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"StatesSumCalcPct <> 100.0 - sum: {0} - diff: {1}  ", sumPCalc.ToString("N" +
decplcs.ToString()), (sumPCalc - 100.0).ToString("N" + decplcs.ToString()));

                    if (!Math.Round(sumPFromQty, decplcs).Equals(100.0))
                        errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"TotQtyCalcPct <> 100.0 - sum: {0} - diff: {1}  ", sumPFromQty.ToString("N" +
decplcs.ToString()), (sumPFromQty - 100.0).ToString("N" + decplcs.ToString()));

                    if (!Math.Round(sumPCalc,
decplcs).Equals(Math.Round(sumPFromQty, decplcs)))
                        errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"StatesSumCalcPct  sum: {0} <> TotQtyCalcPct {1} - diff: {2}  ",
sumPCalc.ToString("N" + decplcs.ToString()), sumPFromQty.ToString("N" +
decplcs.ToString()), Math.Round(sumPCalc - sumPFromQty, decplcs).ToString("N" +
decplcs.ToString()));

                }
            }
            catch (Exception ex)
            {
                ex.PreserveExceptionDetail();
                throw;
            }
            finally
            {
            }

            return errSB.ToString() ?? String.Empty;
        }

        /// <summary>
        /// Extension method to evaluate incoming element inspection data and
determine if quantity data has any obvious or illogical data.
        /// </summary>
        /// <param name="qState"></param>
        /// <param name="quantity"></param>
        /// <param name="decplcs"></param>
        /// <returns></returns>
        public static String ReportQuantityErrors(Double[] qState, Double
quantity, int decplcs)
        {
            var errSB = new StringBuilder();
            Double sumQ = 0.0;
            Int32 i = 0;

            try
            {
                if (quantity <= 0)
                    errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"Quantity <= 0.0 - {0}", quantity);

                var qty = 0.0;
                foreach (Double q in qState)
                {
                    i++;
                    qty = Math.Round(q, decplcs);

                    if (qty < 0.0) // 0 is ok.
                        errSB.AppendFormat((errSB.Length > 0 ? " | " : "") + "Q
CS{0} < 0 - {1} ", i, q);

                    if (qty > Math.Round(quantity, decplcs))
                        errSB.AppendFormat((errSB.Length > 0 ? " | " : "") + "Q
CS{0} > TotQty  - {1}:{2} ", i, q, quantity);

                    sumQ = sumQ + qty;
                }

                if (sumQ <= 0.0)
                    errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"SumCSQty <= 0.0 - {0} ", sumQ);

                if (!Math.Equals(Math.Round(Math.Abs(sumQ - quantity),
decplcs), 0.0))
                    errSB.AppendFormat((errSB.Length > 0 ? " | " : "") +
"SumCSQty <> TotQty - sum: {0} qty: {1} diff: {2} ", sumQ, quantity,
Math.Round(sumQ - quantity, decplcs));
            }
            catch (Exception ex)
            {

                ex.PreserveExceptionDetail();
                throw;
            }
            finally
            {
            }

            return errSB.ToString() ?? String.Empty;
        }

    }
}
```

5. Any of the Rules, CoRe element, AASHTO data, and transformation parameters confirmation dialog rows can be saved to Excel, Csv, HTML, or text files using a **right-mouse-click context menu**. Click anywhere on the grid and a self-explanatory popup export options menu will be displayed. If rows are selected then only those rows will be exported, otherwise all the grid rows (recognizing any filtering in place) will be exported. By default these will go to an Exports subdirectory in the user's Migrator working directory tree.

- Examples of exports of selected and full Datagrid contents are attached to this issue. Due to limitations of the Telerik export capability, the exports of selected rows generally do not look as nice as exports from the full data grid so filtering is recommended instead of selecting specific rows when possible.
- A warning may be shown when starting Excel to open an xls export. The warning message is attached to the issue as a PNG. This warning is generated because the internal Excel file format used by Telerik controls to export xls is actually **Excel Xml**, not rhe pure **Excel xls/xlsx** native

format, although the file extension is xls.
Just click OK when the error message appears and the data should load into Excel. To eliminate this error, simply save the file to the standard **Excel xls/xlsx** file format

6. If incoming units for data do not match the GUI setting, the incoming file setting will be used and the GUI setting will be updated automatically
7. Full compliance of AASHTO element definitions with the 2013 Manual for Bridge Element Inspection
8. Adjusted rules, basically setting elements 358 and 359 to 7358 and 7359
9. Restored Smart Flag rules to the default rule set (which can be removed)

Miscellaneous

- Right-mouse context menu added as noted above
- Rules manager GUI improved (cleaned up)
- Subterranean plumbing cleaned, tanks relined, leaks removed
- Possible horrid application exception on startup encountered and apparently overcome.
  - This one *may* still occur. If so, **please notify me immediately**. I suspect it is my development environment stepping on the application.
- Help files updated slightly, tooltip added for the Upload Notes button
  - (which is a slight misnomer since it creates a script and does not do a direct uppdate)
- Internal styling made more consistent to improve appearance
- A link has been placed on the Start menu for the program group to navigate to the configured work directory e.g. C:\Work or C:\AASHTO\Visual Element Migrator etc.

Compatibility

Any old Migrator files **should** work with this new version. All changes to the underlying Xml schemas are for optional features.
Please report any issues here.