CUAHSI
universities allied for water research

Text File Export Using the HydroDesktop ImportExport Library: Developer Guide

April 13, 2010

by:

Tim Whiteaker
Center for Research in Water Resources
The University of Texas at Austin

Jingqi Dong
Center for Research in Water Resources
The University of Texas at Austin

Table of Contents

1.0 Introduction

To assist developers in programming applications which export data to delimited text files, HydroDesktop includes a set of export-oriented tools and forms located in the Binaries\ HydroDesktop.Data.dll, with the namespace of "HydroDesktop.ImportExport". (Source is in the Source\Libraries\DataAccess folder.)

There are three export options in this library from which developers may choose when exporting to a text file:

- **DelimitedTextWriter** – Class for writing delimited text files
- **GetExportOptionsDialog** – Dialog for getting common export options for the user
- **ExportDataTableToTextFileDialog** – Dialog for getting the export options and performing the export

These options give you, the developer, flexibility in deciding how you want to export text files. For maximum control, you can write data using the DelimitedTextWriter. Or you can use the GetExportOptionsDialog form to gather and send the user's exporting options to your own program. Or else you can send a DataTable to ExportDataTableToTextFileDialog and let it do all the work.

This document describes the three options for text file export described above, and includes sample code for developers. The intended target audience consists of HydroDesktop developers and programmers.

Data Used in Code Examples

The sample code in this document shows how to export a list of variables that the client builds into a System.Data.DataTable. The code to create the DataTable is shown below.

```
// Create a table of hydrologic variables to export
DataTable tableToExport = new DataTable( "Variables" );

tableToExport.Columns.Add( "Name" );
tableToExport.Columns.Add( "Units" );

DataRow datarow = tableToExport.NewRow();
datarow["Name"] = "Streamflow";
datarow["Units"] = "cubic feet per second";
tableToExport.Rows.Add(datarow);

datarow = tableToExport.NewRow();
datarow["Name"] = "Rainfall";
datarow["Units"] = "inches";
tableToExport.Rows.Add(datarow);
```

Code Snippet 1 Creating a System.Data.DataTable Representing Hydrologic Variable Names and Units

The result is a DataTable object populated with the same information as in Table 1:

Table 1 Example Hydrologic Variable Information

Name	Units
Streamflow	cubic feet per second
Rainfall	inches

2.0 DelimitedTextWriter

This class is used to export data to a file or to a standard output stream, and it gives developers maximum control among the three export alternatives presented in this document. Clients have full control over the choice of output filename, the delimiter, whether to include headers, and whether to append or overwrite output files if they already exist.

This class can be utilized via its static methods to write all of the data in a single step, or instances can be created when it is desirable to write data to the text file one line at a time. The static methods include the ability to pass in a background worker if progress should be reported as the text file is written.

The following sections explain in detail how to use these methods.

2.1 Static Methods

In this situation clients create a System.Data.DataTable including all the data they want to export, and then use DelimitedTextWriter to create and populate the text file in a single method call, passing in the DataTable as an argument to the method. There are two static methods for this purpose:

- DataTableToStream – Writes the data to a standard output stream
- DataTableToDelimitedFile – Writes the data to a text file

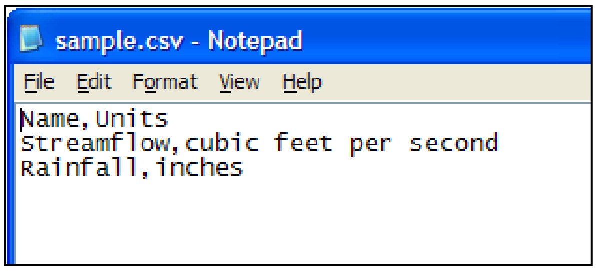
Sample code to export the list of hydrologic variables is shown below. The code uses the tableToExport variable that was populated in the sample code from Section 1.0.

```
// Set arguments and export data
string filename = "c:\\temp\\sample.csv";
string delimiter = ",";
bool includeheaders = true; // true if output should include field names
bool append = true; // true to append; false to overwrite

HydroDesktop.ImportExport.DelimitedTextWriter.DataTableToDelimitedFile
        (tableToExport, filename, delimiter, includeheaders, append);
```

Code Snippet 2 Exporting a Data Table to a Text File with DelimitedTextWriter

The data are written to the .CSV file as shown in the example output below:



```
Name,Units
Streamflow,cubic feet per second
Rainfall,inches
```

Figure 1 Example Text Output from DelimitedTextWriter

2.2 Reporting Progress with a Background Worker

The static methods of DelimitedTextWriter support the reporting of progress via a background worker. If the client should respond to progress (e.g., increment a progress bar) as the data are written to the stream or text file, a background worker and its event arguments can be passed to the static methods described in the previous section. Via the reportingOption argument, clients can specify whether the DelimitedTextWriter should report user state and progress, or just progress, or nothing at all.

```
// Background Worker DoWork Event in the Client
private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{
        object[] parameters = e.Argument as object[];
        BackgroundWorker worker = sender as BackgroundWorker;

        // Read parameters passed to the background worker
        string filename = (string) parameters[0];
        string delimiter = (string) parameters[1];
        bool includeheaders = (bool) parameters[2];
        bool append = (bool) parameters[3];
        DataTable tableToExport = (DataTable) parameters[4];

        // Perform the export with progress reporting turned on
        HydroDesktop.ImportExport.DelimitedTextWriter.DataTableToDelimitedFile(
             tableToExport, filename, delimiter, includeheaders, append, worker, e,
             HydroDesktop.ImportExport.BackgroundWorkerReportingOptions.UserStateAndProgress);
}
```

Code Snippet 3 Using a Background Worker to Report Progress During Export with DelimitedTextWriter

2.3 Instance Methods

Like basic text file writers, instances of DelimitedTextWriter can be created to write data to a file or stream one line at a time, as in the example below. Notice that the WriteLine method takes either a string array or a System.Data.DataRow as input. Each array item or column in the input is separated by the delimiter in the output.

```
// Create an instance of the DelimitedTextWriter to export the data
string filename = "c:\\temp\\sample.csv";
string delimiter = ",";

HydroDesktop.ImportExport.DelimitedTextWriter writer =
        new HydroDesktop.ImportExport.DelimitedTextWriter(filename, delimiter);

// Write the headers
bool checkFormatting = true; // ...in case data includes the delimiter
string[] items = new string[2] { "Name", "Units" };

writer.WriteLine(items, checkFormatting);

// Write each row to the output
foreach (System.Data.DataRow row in tableToExport.Rows)
{
        writer.WriteLine(row, checkFormatting);
}

writer.Close();
```

Code Snippet 4 Writing Data to a Text File One Line at a Time with DelimitedTextWriter

In the example, the “checkFormatting” argument is used to check whether the data contains the delimiter. If it does, then the item is enclosed in quotes so that the output file can be parsed. This option is useful when the data might include the delimiter (e.g., “Colorado River at Austin, TX” as a SiteName value in a comma delimited file), but it also slows down processing. If you are sure that the data does not include the delimiter, then you can set this option to false to speed things up.

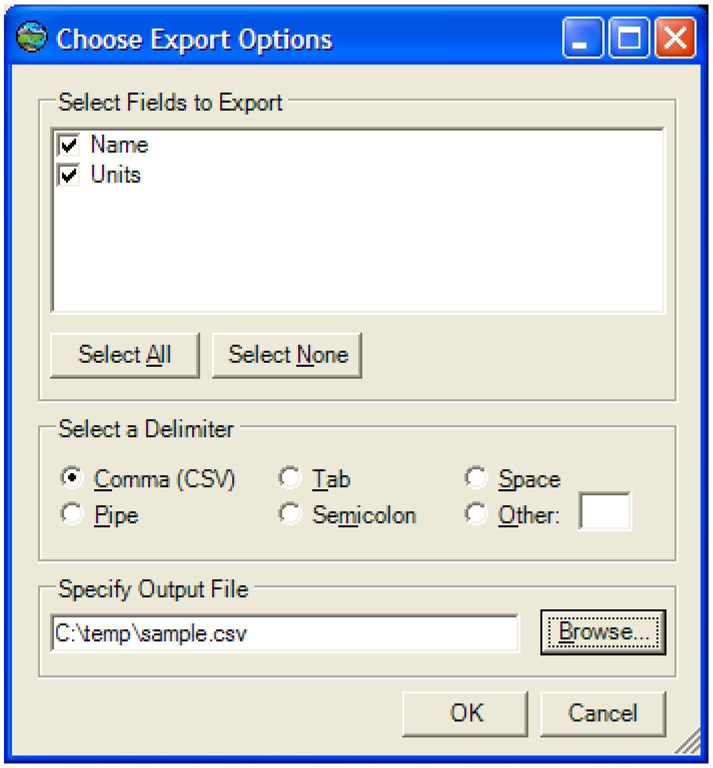
The output result will be the same as in Figure 1.

3.0 GetExportOptionsDialog

Some of the common tasks when preparing to write a delimited text file are:

- Choosing items (e.g., fields in a table) to export
- Choosing a delimiter (e.g., comma, tab)
- Specifying the output text file path and filename

The GetExportOptionsDialog form is designed to prompt the user for these items. After gathering these items from the user, control is returned to the client so that the client can perform the export. This dialog is useful when the client needs export options from the user, and then the client has its own scheme for exporting the data once the options have been gathered.


Choose Export Options

Select Fields to Export

- [✓] Name
- [✓] Units

Select a Delimiter: [✓ Comma (CSV)] Tab Space Pipe Semicolon Other:

Specify Output File: C:\temp\sample.csv

Figure 2 GetExportOptionsDialog Form

Inputs

When creating an instance of GetExportOptionsDialog, a System.Data.DataTable object is passed to the constructor. The DataTable doesn't actually have to have any data in it, because it's only used to provide the list of fields for export. The dialog lets users choose which fields they want to export.

```
// Create an instance of GetExportOptionsDialog to collect export options
HydroDesktop.ImportExport.GetExportOptionsDialog exportForm =
        new HydroDesktop.ImportExport.GetExportOptionsDialog(tableToExport);
```

Code Snippet 5 A DataTable with a Fields to Export Is Passed to the GetExportOptionsDialog Constructor

Outputs

If the DialogResult is OK, the options chosen by the user can be retrieved from the public properties of the form.

```
// Retrieve user's options from the public properties
if (exportForm.ShowDialog() == DialogResult.OK)
{
        List<string> fields = exportForm.FieldsToExport;
        string delimiter = exportForm.Delimiter;
        string filename = exportForm.OutputFilename;
}
```

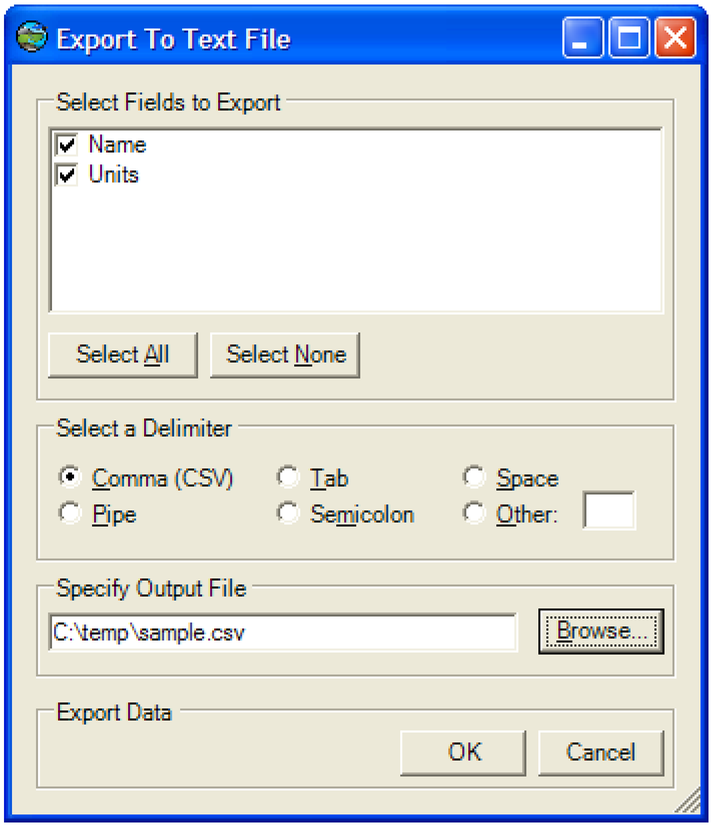
Code Snippet 6 Reading Export Options from GetExportOptionsDialog

Once the options have been retrieved from the form, it's up to the client to use those options to export the data (perhaps by using the DelimitedTextWriter class).

4.0 ExportDataTableToTextFileDialog

The GetExportOptionsDialog is useful for gathering options from the user regarding data export, but it then turns control back over to the client in order to perform the export. If the data export is straightforward, then it may be desirable for a dialog to not only gather user options, but also to perform the export. The ExportDataTableToTextFileDialog form has been created for this purpose.

Given a System.Data.DataTable with all of the data to export, the ExportDataTableToTextFileDialog form gathers export options from the user and then performs the data export, showing progress during the process. It is designed to make life easier for the client developer by handling all aspects of the data export (except for supplying the data to export, which of course should be handled by the client).



Fields: [✓ Name] [✓ Units]

Delimiter: [✓ Comma (CSV)] Tab Space Pipe Semicolon Other:

C:\temp\sample.csv

Figure 3 ExportDataTableToTextFileDialog Form

USAGE

When creating an instance of ExportDataTableToTextFileDialog, a System.Data.DataTable object with the data to export is passed to the constructor. The ShowDialog method is then called, which turns control over to ExportDataTableToTextFileDialog to handle the data export, as in the example below.

```
// Create an instance of ExportDataTableToTextFileDialog to export the DataTable
HydroDesktop.ImportExport.ExportDataTableToTextFileDialog exportForm =
       new HydroDesktop.ImportExport.ExportDataTableToTextFileDialog (tableToExport);

exportForm.ShowDialog();
```

Code Snippet 7 Exporting Data Using ExportDataTableToTextFileDialog