FeneVision® Best Practice

Import/Export Development (BP0109)

Revision	Date	Description of Change	Revised By
A	11/02/2016	Initial Document	TRC
B	12/19/2016	Test Shell uses DLLs installed by Core.	LMW
C	11/17/2020	Advanced Development	ALB
D	08/10/2023	Test Shell Setup	SAB
E	10/04/2023	Message Logging	GEC
F	01/26/2024	Added FV Agent Order Import Jobs section	TRC
G	02/19/2024	Added additional to Solution Troubleshooting	CLT

Introduction

This document outlines the steps required to develop FeneVision Import/Export scripts. This includes the following import and export document types.

- Import
  - Order
  - Receipt
  - Cycle Counts
  - Bin Transfer
- Export
  - PO
  - Invoice
  - Order (Acknowledged, Release, Shipped, Closed)
  - Route (Shipped)

Order imports are used to ease the entry of orders from large customers that order standard items with regularity. A file with a hundred items will take a user much more time to enter than to import using a pre-defined script and file format.

Exports are used in a comparable manner to imports, but in most cases a customer or vendor is asking for a file to ease the entry of data on their end. Both parties benefit because of the consistency of the file and the speed at which information can be transferred.

FeneVision has a base XML import/export for each type. If this base XML file can be used, no custom development is necessary. This base XML file should be reviewed first and used if possible. This document will focus on the scenario where the base FeneVision XML will not work.

Programming (coding) experience is required to create a custom import/export script. Scripts can be written using Visual Basic (VB) or C#, so experience with either language is also preferred.

FeneVision Core needs to be installed on the computer with the Test Shell solution, since the Test Shell uses DLL files installed by Core.

Visual Studio Installation

Scripts can be developed with the use of a plain text editor or Visual Studio (VS). However, a text editor will not provide intellisense for those not familiar with FeneVision objects, so this document will focus on using VS. Follow the steps below to install VS.

Download Microsoft Visual Studio Express using this link. Scroll down to bottom and select “Express for Desktop.” Once installed, start the application. If prompted for environment setup, choose VB/C# development. The PC with VS installed also needs FeneVision CORE installed.

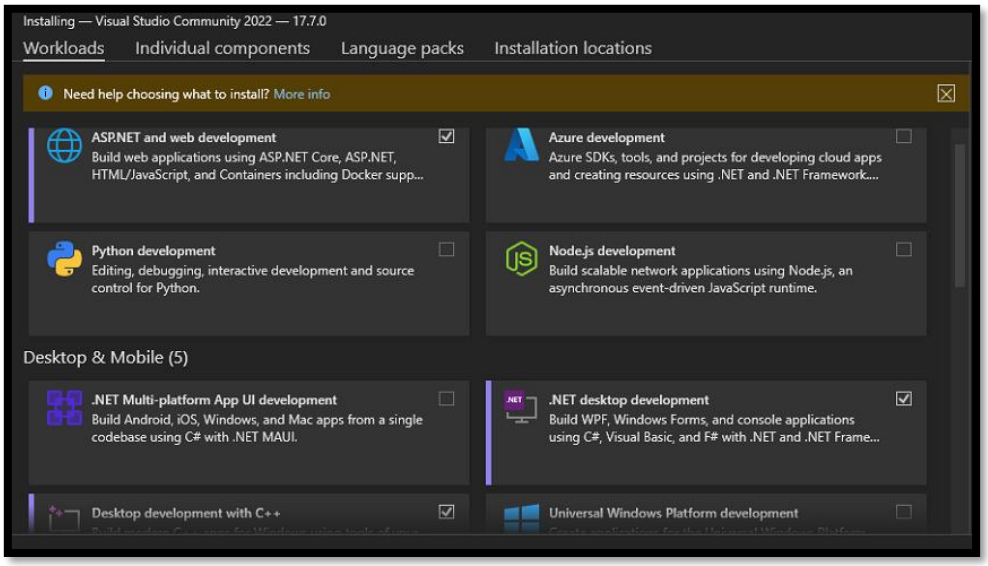
Note: The Community version of Visual Studio is also an option but requires a Microsoft login to unlock it for free. Also, a full version of VS can be used if the proper licensing is available.

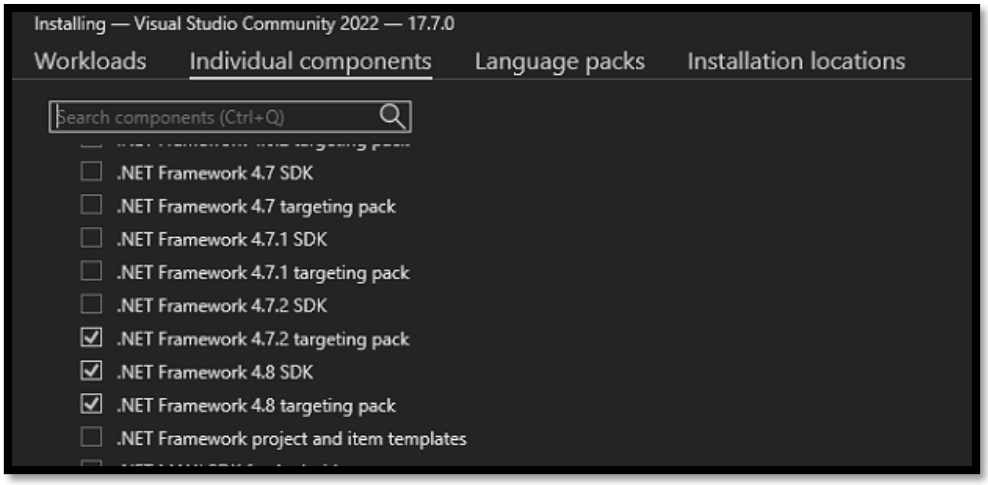
Import/Export Test Shell

FeneTech has developed a Test Shell to allow for easier development of import/export scripts. This shell is a VS solution that references pre-compiled DLLs and allows the user to evaluate their code with intellisense, break points and real-time error reporting.

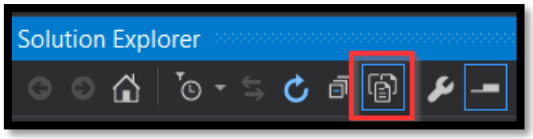
Contact FeneTech for a copy of this solution (Utilities folder with each build).

After getting a copy of the solution, you’ll need to open it in the latest version of Visual Studio. When installing visual studio check off the following Workloads: Asp.Net and web development, .Net Desktop development, and Desktop Development with C++. Also, make sure you check the necessary .Net Framework required for Core. At the time of this edit, the Framework is 4.8.





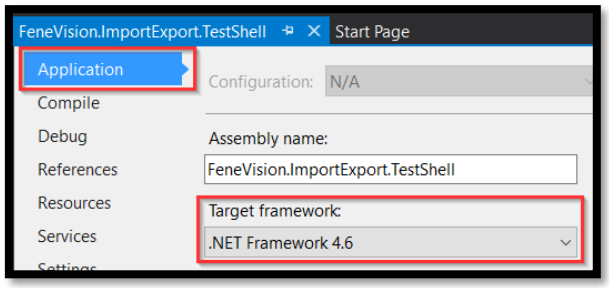
Open the solution in VS and Start the application. Turn on “Show All Files” to show files not included in the project. If you have errors, see the Solution Troubleshooting below for troubleshooting tips.



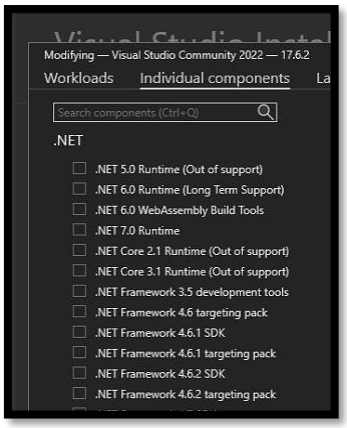
Solution Troubleshooting

Below are potential issues that may occur when attempting to build the project.

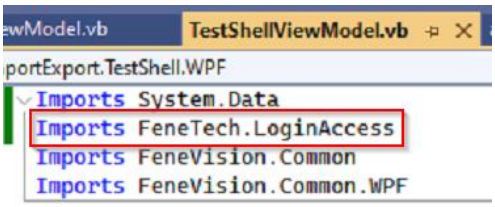
- **Issue:** Incorrect “Target framework”
  - **Solution:** Open “My Project”, choose the “Application” page, change the “Target framework” setting



- **Note:** The .NET Framework version must match the version used by FeneVision which is listed in the FeneVision Hardware and Software Requirements document. If the Test Shell project does not show the version needed in the above dialog, then install or download the .NET Framework Developer Package for the version so it is available to the Test Shell.
  - Newer versions of Visual Studio Installer allow the targeting packs to be installed individually. Click the Modify button and then navigate to the Individual Components tab. The first section is .NET.



- **Issue:** ‘OptionsWizardBase’ is not declared
  - BC30451 'OptionsWizardBase' is not declared. It may be inaccessible due to its protection level.
  - **Solution:** Add FeneVision.Options.dll as a reference.
- **Issue:** Type ‘BasicNoUIObject’ is not defined
  - BC30002 Type 'BasicNoUIObj' is not defined.
  - **Solution:** Add Interopt.ww10_com.dll as a reference.
- **Issue:** Type ‘Shape’ is not defined
  - **Solution:** Add FeneVision.ShapeLibMgr.dll as a reference.
- **Issue:** Error with the Connection Registry not being Defined in TestShellViewModel.vb.
  - **Solution:**
    - Add FeneTech.LoginAccess.dll from the Core folder to the references.
    - Import FeneTech.LoginAccess.
    - May also need to add “Common.” To the DatabaseSelector line.



- **Issue:** Argument not specified for parameter 'sublocationKey' in ImportControlViewModel.vb.
  - BC30455 Argument not specified for parameter 'sublocationKey' of 'Sub Init(filePath As String, connectionString As String, locationID As String, importFileFormatID As Integer, postTransaction As Boolean, employeeID As Integer, sublocationKey As Integer)'.
  - **Solution:**
    - Run the following SQL to get the 'sublocationKey', changing the LocationID if needed.
    - SELECT SublocationKey FROM Sublocations WHERE LocationID = 'MAIN'
    - Add 'SublocationKey' to the end of 'inventoryImportHandler.Init' on line 426. You may also want to update location from Main in function.

Sample Files

The Test Shell solution comes complete with sample files for each of the import/export types. One sample is required to exist to build the project.

- SampleExport.vb – Sample PO Export script
- SampleImport.vb – Sample Order import script
- SampleImportEDI.vb – Sample Order import script with more detailed logic provided
- SampleInvoiceExport.vb – Sample Invoice Export script
- SampleOrderExport.vb – Sample Order Export script

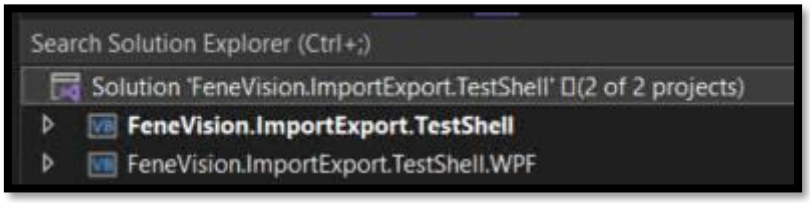
These samples are bare bones. Contact FeneTech for a sample file that may contain a better foundation.

Development

Once the VS solution compiles without error, development can begin. The Advanced Development section below will outline advanced resources and techniques that can be used.

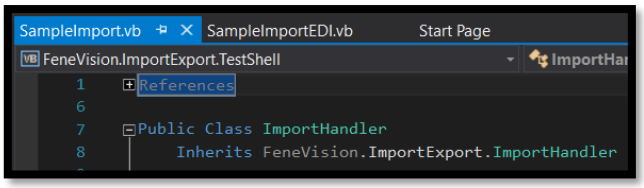
The latest versions of the Test Shell solution contain two separate projects. The first is designed to contain all the custom import/export files while the second contains the shell UI. Users should not need to ever edit the second project.

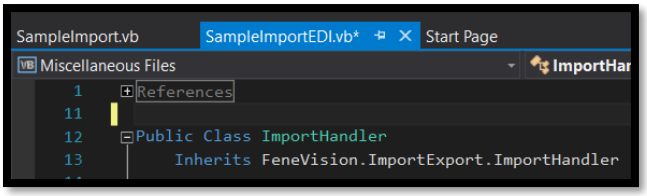
- FeneVision.ImportExport.TestShell – Contains custom files/scripts.
- FeneVision.ImportExport.TestShell.WPF – Contains the WPF UI for the test shell. **This should not be edited except for fixes above!**



File Inclusion

Only one class of each import/export type can exist in the solution at any one time. For example, the SampleImport and SampleImportEDI files cannot both be included in the project at the same time because they both define the "ImportHandler" class. Files should be included when being worked on but excluded when working on another script.

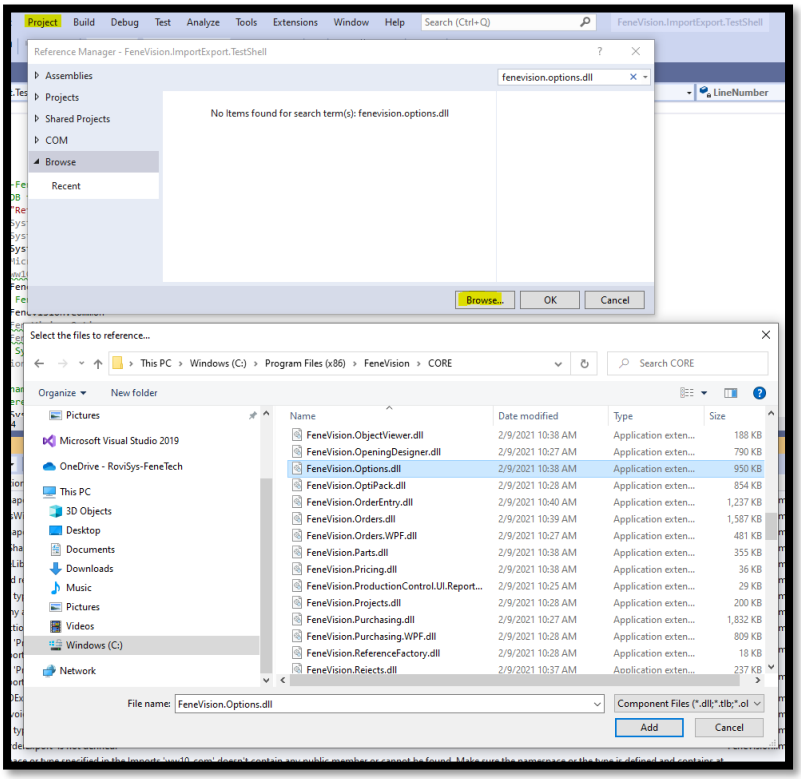




Files that are not currently included in the project can still be seen in the Solution Explorer with “Show All Files” turned on.

Adding a Reference

To add a reference to the project, such as FeneVision.Options.dll, use the Project Menu and select Add Reference. Select Browse on the left side of the dialog and then the Browse button, navigate to the folder, select the DLL, and press Add.



Advanced Development

Below are advanced topics for import/export development and should only be attempted with experience.

Structure

Below is the best practice for structuring the code. This will improve readability for developers that may need to look at the code for troubleshooting.

- Public Class ImportHandler Inherits FeneVision.ImportExport.ImportHandler
    - Public Overrides Function ImportToFtOrders() As EResult
        - Get file
            - Dim filepath As String = MyBase.InputFilePath
        - Create instance of CustomImportClass
            - Dim import As New CustomImportClass(MyBase.ConnectionString, Me)
                - Me is the instance of ImportHandler
        - Call entry method of CustomImportClass
        - Either returns EResult.eOKAY on success or EResult.eFAIL_FULLSTOP on failure
    - Add any other Main Import Helper Methods here
    - CustomImportClass
        - Define Constructor

```
Class ImportRoom

    Private _connectionString As String
    Private _import As ImportHandler

    Public Sub New(connectionString As String, import As ImportHandler)

        _connectionString = connectionString
        _import = import

    End Sub
```

        - Entry method called from ImportToFtOrders

        - Calls Parse method which should parse file into a data structure with easily accessible variables
        - Should then take parsed data structure and pass it as a parameter to a ImportOrder function
    - Create method to parse file
    - Create methods to create Order and insert Order data
    - _import.AddOrderToCollection(FVOrder)
        - Adds the newly created order to ImportHandler.Orders Collection
        - This should be the last call after all line items and Options have been added
    - Create any other helper or common methods that are needed

Custom Dialog Boxes and Forms

Sometimes, the Order Import may require functionality to require the user answer a yes/no question, select a different part, a different option, etc. Being able to create custom dialog boxes and forms will help give customers more control over the import process versus being overwhelmed by error messages.

Custom Form

Start by creating a new project from scratch, selecting Windows Forms App as the project template. After the project is created, there should be a blank project with a Form. Select View->Toolbox to select from various tools like buttons, combo boxes, etc.

After selecting the Controls needed for the form and the general layout is defined, double-clicking any of the Controls will generate an empty method to define the functionality of the Control.

Next, define the method that will “Load” the form with data. See Appendix A for an example load method.

Define a constructor for the form.

```
Public Class Form1
    Private _connectionString As String
    Public Sub New(conn As String, item As LineItem)
        'Call to MyBase.New must be the very first in a constructor.
        MyBase.New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()
        'Set ConnectionString
        _connectionString = conn
```

Evaluate the form to ensure the data needed is loaded correctly and the controls function correctly. Once the form is evaluated, copy the Class from Form1.vb and the partial class from Form1.Designer.vb and nest them in the ImportHandler class. Then, write the code to open the new form when it is needed.

```
Using selectPart As New Form1(_connectionString, item)
    selectPart.StartPosition = FormStartPosition.CenterScreen
    If selectPart.ShowDialog() = DialogResult.OK Then
        result = selectPart.cmbParts.SelectedItem
        partNo = result.Row.Item(1).ToString
        partNoSuffix = result.Row.Item(2).ToString
        item.PartNo = partNo
        item.PartNoSuffix = partNoSuffix
    Else
        'Cancel Import (Rolls back current file)
        Throw New Exception("PartNo: " + item.PartNo + " Location: " + item.PartNoSuffix + " does not exist." + vbCrLf + "Import cancelled by User!")
    End If
End Using
```

Setup the form to appear in the middle of the screen.

Expects an OK click or Cancel click corresponding to DialogResult.OK or DialogResult.Cancel.

Generic Dialog Box

In certain situations, a custom form is not necessary and all we need is a yes/no answer from the user. A generic dialog result MsgBox works best in these scenarios. There is no designer needed for the MsgBox to work properly.
See Appendix A for an example function.

Querying FVMaster

It is possible to query FVMaster to get information necessary to import/export. Below are examples of this in practice. Knowledge of SQL and the FeneVision schema is required.

Scalar Value

```
Dim SQL As String = "SELECT [Description] FROM MasterPartList WHERE MasterPartNo=@PartNo AND
PartNoSuffix=@PartNoSuffix"
Dim SQLParameters As New SqlParameters
SQLParameters.Add(New SqlClient.SqlParameter("@PartNo", partNo))
SQLParameters.Add(New SqlClient.SqlParameter("@PartNoSuffix", partNoSuffix))

Dim Result As Object = FeneVision.Common.DatabaseMethods.GetScalar(ConnectionString, SQL,
CommandType.Text, SQLParameters.ToArray)
```

DataTable

```
Dim SQL As String = "SELECT mpo.Code, mpo.OptionType & 1 AS [Default] " &
    "FROM MasterPartList mpl " &
    "JOIN MasterPartOptions mpo ON mpo.MasterPartNo=CASE WHEN mpl.SubPartNo='' THEN
mpl.MasterPartNo ELSE mpl.SubPartNo END " &
    "AND mpo.PartNoSuffix=CASE WHEN mpl.SubPartNoSuffix='' THEN mpl.PartNoSuffix ELSE
mpl.SubPartNoSuffix END " &
    "WHERE mpl.MasterPartNo=@PartNo AND mpl.PartNoSuffix=@PartNoSuffix AND mpo.[Group]=@Group"
Dim SQLParameters As New SqlParameters
SQLParameters.Add(New SqlClient.SqlParameter("@PartNo", partNo))
SQLParameters.Add(New SqlClient.SqlParameter("@PartNoSuffix", partNoSuffix))
SQLParameters.Add(New SqlClient.SqlParameter("@Group", group))

Dim Result As DataTable = FeneVision.Common.DatabaseMethods.GetDataTable(ConnectionString, SQL,
CommandType.Text, SQLParameters.ToArray)
```

Table Lookups

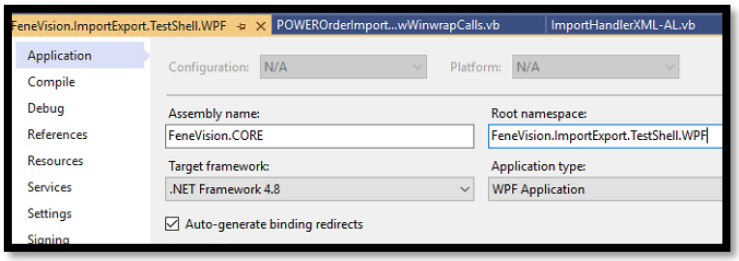
The base import/export functionality allows you to call “TableLookup” directly to get lookup tables values from import/export lookup tables. In certain cases, it may be necessary to call into lookup tables not defined for a specific import/export, but instead defined in Setup > Products > Lookup Tables. To do this, it is necessary to instantiate an object of the FeneVision BOM script engine. See below for an example. Note that the “Init” call is done inside of the overridden function for each import/export type (Order import would be the “ImportToFtOrders” function).

```
Private ScriptEngine As New FeneVision.BOM.ScriptExt
…
ScriptEngine.Init(ConnectionString, ModuleType.Order)
…
ScriptEngine.TableLookup("MyLookupTable", "X", "Y")
```

Expression Evaluation

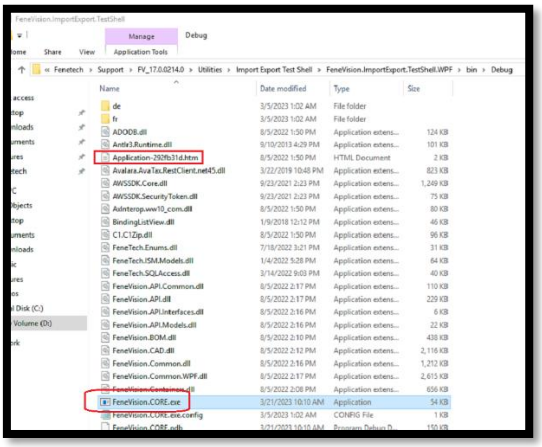
It is possible to evaluate expressions in an Order import script. This includes Visibility, Default and Default Value Expressions. However, doing so requires running WinWrap. WinWrap will not automatically be able to work in Visual studio due to licensing issues but can be made to function during debugging if you complete the following steps.

1. Right click properties of the WPF project, select the Application tab in the top left, and change the Assembly name to Fenevision.CORE.

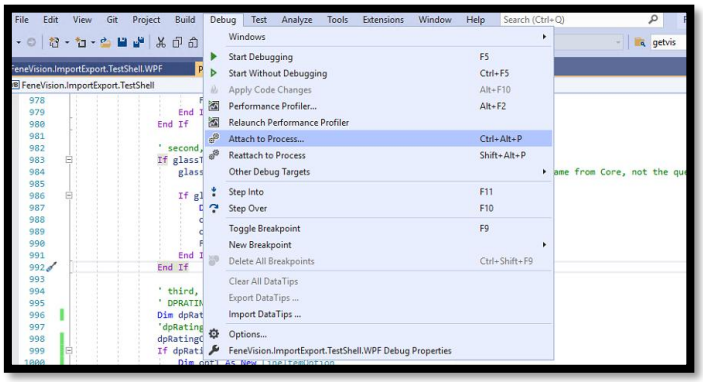


2. Build the application to create the exe.

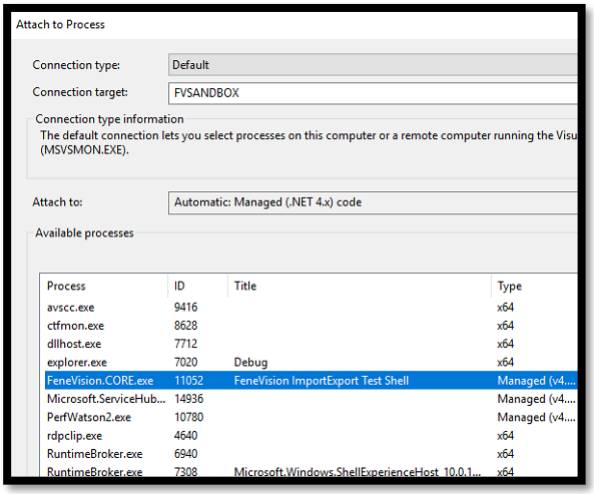
3. Copy the Application-292fb31d.htm file out of the FeneVision CORE folder in Program Files (C:\Program Files (x86)\FeneVision\CORE) and place it in the directory where the newly built FeneVision.CORE executable exists. This should be something like Import Export Test Shell\FeneVision.ImportExport.TestShell.WPF\bin\Debug.



4. Before running the project in Visual Studio, the FeneVision.CORE.exe needs to be run first. This will bring up the ImportExport Test Shell application.
5. Attach the code in VS to that process by selecting the Debug file menu and selecting Attach to Process.



6. Attach the FeneVision ImportExport Test Shell. Note that the process name will be FeneVision.CORE.exe.

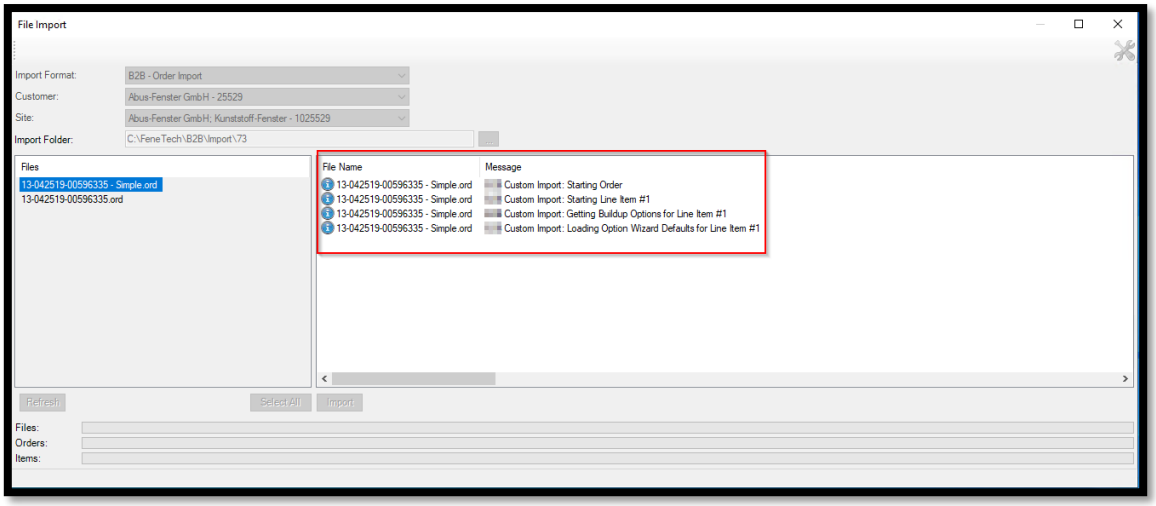


7. The Test Shell can now be used as normal with breakpoints to debug the code in Visual Studio.

Message Logging

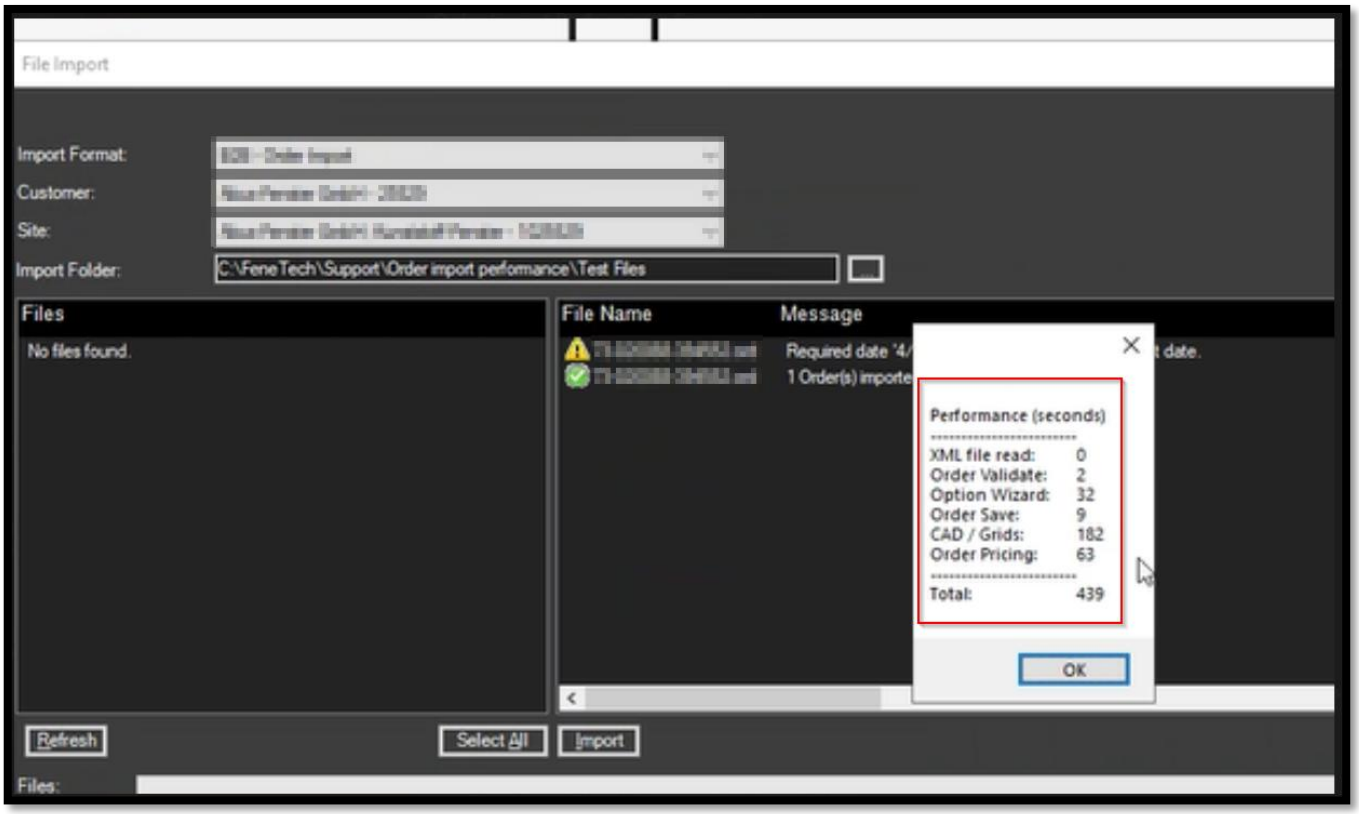
Custom file import scripts run within the File Import screen before the base import methods take over. Due to this, on larger orders, the import process may appear to not do anything until the base import takes over and starts updating the status within the UI. It is possible for the custom import scripts to raise messages to that UI directly to help the end user understand the progress.

```
MyBase.OnLogStatusDetailed(EImportStatusTypes.eISTAT_Informational, InputFileName, "YOUR MESSAGE HERE")
```



Performance

After an order Import runs in Core, press Ctrl+Shift+P to see a summary of the performance at each step of the import.



FeneVision Agent Order Import Jobs

Once developed, order imports can be set up to automatically run every X minutes. For FeneTech resources, the original write up is located here.

An agent job can be set up that will automatically import orders, removing the need for someone to manually do this. Files that are successfully imported will be moved to the 'backup' folder. If a file fails to import, a notification email will be sent to a configurable email address, and the file will be moved to a 'Failed' folder so that it's easy to find which files failed.

Note that one job is required for each import definition.

Depending on FeneVision version, the AgentJobs table will exist in either FVPerformance (latest) or FVMaster. A record will need to be inserted into this table with the appropriate parameters.

```
INSRET INTO dbo.AgentJobs (JobName, ProgID, Params, StartDateTime, RunsEveryXMinutes,
NextRunDateTime, Enabled, ThreadPriority)

VALUES (Order Import – Format 1', 'FeneVision.ImportExport.OrderImportJob',
'Database=FVMaster;FormatID=1;FilePath=C:\temp\FV\OrderImport1;LocationID=MAIN;ImportAll=1;Ext
ension=*.csv', '1/1/2024', 1440, '1/1/2024', 1, 2)
```

Job params

- Database (Required) (Default: FVMaster)
- FilePath (Required) (Default: C:\)
- LocationID (Optional) (Default: MAIN)
- FormatID (Optional) (Default: -1) For custom definitions, this is the ImpExpFormatID from ImpExpFormats.
- ImportAll (Optional) (Default: 0) ImportAll = 0 means it's going to import the FILE that's specified as part of your 'FilePath' param. 1 means you import any file in the folder.
- Extension (Optional) (Default: *.xml for default import definition, Required with no default for custom definitions)

Appendix A

MsgBox

```
Private Function YesNoDialogOrderExists(okey As Integer, CustomerID As String, SiteID As
String, PONumber As String) As Boolean

	Dim answer As MsgBoxResult = MsgBox("Order already exists for CustomerID: " & CustomerID
	& " SiteID: " & SiteID & " PONumber: " & PONumber & vbCrLf & vbCrLf & "Would you like to
	override the existing order?", MsgBoxStyle.Exclamation + MsgBoxStyle.YesNo +
	MsgBoxStyle.DefaultButton2, "Order Import")

	If answer = MsgBoxResult.Yes Then
		DeleteOrder(okey)
		AddMessage("Order was overridden for CustomerID: " & CustomerID & " SiteID: " &
		SiteID & " PONumber: " & PONumber & "!", EImportStatusTypes.eISTAT_Informational)
	Else
		AddMessage("Skipping Import for CustomerID: " & CustomerID & " SiteID: " & SiteID
		& " PONumber: " & PONumber & "!", EImportStatusTypes.eISTAT_Informational)
		Return False
	End If

	Return True
End Function
```

Custom Form Load Method

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
	Dim connectionString = _connectionString
	Dim Sql = "SELECT MasterPartNo + '(' + PartNoSuffix + ') - ' + Description AS Part,
	MasterPartNo, PartNoSuffix "
	Sql = Sql + "FROM MasterPartList "
	Sql = Sql + "WHERE Ordered = 1"

	Dim connection = New SqlConnection(connectionString)
	Dim adapter As New SqlDataAdapter()
	Dim ds As New DataSet()

	Try
		connection.Open()
		Dim Command = New SqlCommand(Sql, connection)
		adapter.SelectCommand = Command
		adapter.Fill(ds)
		adapter.Dispose()
		Command.Dispose()
		connection.Close()
		cmbParts.DataSource = ds.Tables(0)
		cmbParts.ValueMember = "Part"
		cmbParts.DisplayMember = "Part"
	Catch ex As Exception
		MessageBox.Show("Can not open connection ! ")
	End Try
End Sub
```