Design Document

McMaster-Carr Add-In for Autodesk Inventor
Version 1.0.4 • December 07 2019

Nicholas Quandt
nicholas.quandt@marquette.edu

Marquette University Masters Student of
Computational Mathematical and Statistical Sciences

1 Overview

This project aims to implement an embedded web-browser into Autodesk Inventor 2019 [1]. The purpose being to allow a user to browse the McMaster-Carr online catalog of products, in order to eventually include a product within their Inventor files [2]. Nearly every product in the catalog has a 3D model available for download, along with properties pertaining to the part. The McMaster Add-In for Autodesk Inventor (MAFI) will reduce time needed to implement these 3D models into an Inventor project.

2 Context

This docuemnt should give a user or other developer ample knowledge of the current state of MAFI, including software packages implemented, code repositories, and languages used, along with goals for the future, challenges faced, etc.

2.1 Document Conventions and Definitions

The following describes all naming conventions used throughout the document:

- MAFI: The McMaster-Carr Add-In for (Autodesk) Inventor software.
- Inventor: The computer application "Autodesk Inventor 2019" developed by Autodesk, Inc. [1].
- McMaster: The online catalog available at www.mcmaster.com [2].
- API: Application programming interface.
- 3D model: A three dimensional representation of a physical object, as shown within a computer program such as Inventor.
- Add-in: A software that relies on a parent application in order to execute, typically embedded within the parent application, i.e. non-standalone.
- IProperties: A set of attributes embedded within each Inventor file such as part number, description and physical material.
- Button: A user interface element that executes a process of an application from a mouse click.
- .NET: A software framework developed by Microsoft that includes a large class library and provides language interoperability across several programming languages [3].
- CefSharp: A fast, fully embedded web browser for .NET applications [4].

3 Scope

MAFI is an application add-in built for Inventor as a method for more efficiently connecting to and retrieving models from, the McMaster database of 3D models that often are used by mechanical designers. It will allow for a, direct, in-program access to the online catalog. Along with, "fast" conversion of the database file-type (.stp) into something usable by an Inventor user (.ipt). This is a software that should speed up design projects for engineers who use 3D models available from McMaster, in their Inventor projects.

4 Overview of Requirements

These requirements are a summary of the user stories gathered from the developer and potential users of the add-in. See Appendix A for a collection of user stories.

1. Runs in Windows 64-bit for Inventor 2019 software.
2. Use Inventor API to add Button for interface.
3. Use McMaster online catalog that has links to 3d models of the majority of their items. Primarily hardware, fasteners, etc.
4. Best to keep online interface from McMaster, and somehow open the "form" in inventor to access.
5. Replace "add to cart" button in HTML with "add to assembly" or "open as part" buttons.

5 Languages and Packages

- C#
- XAML
- HTML
- Autodesk Inventor Interop 2020
- CEFSharp v69.0 [4]
- .NET 4.7.2 [3]

6 User Interface

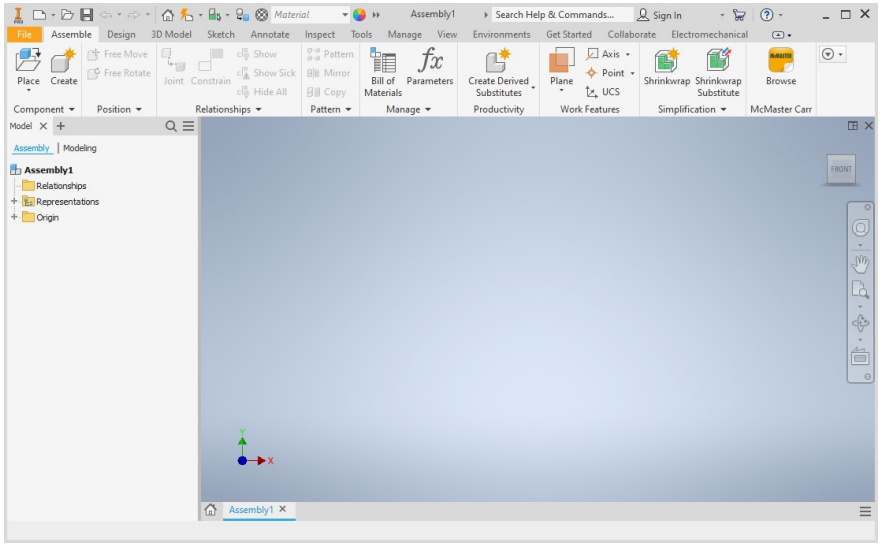

Figure 1: The button(far right) addition to the ribbon interface within Autodesk Inventor.

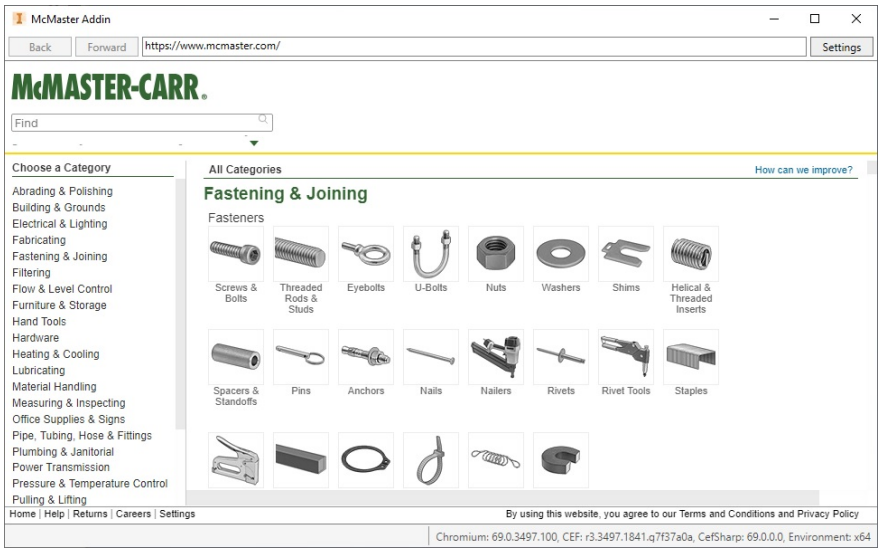

Figure 2: The browser instance following the button execute event.

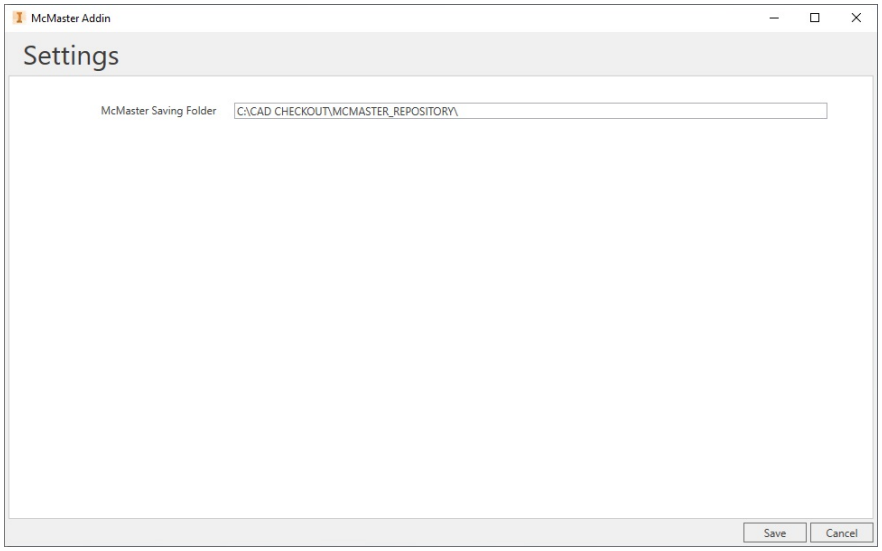

Figure 3: The settings page for the Add-In.

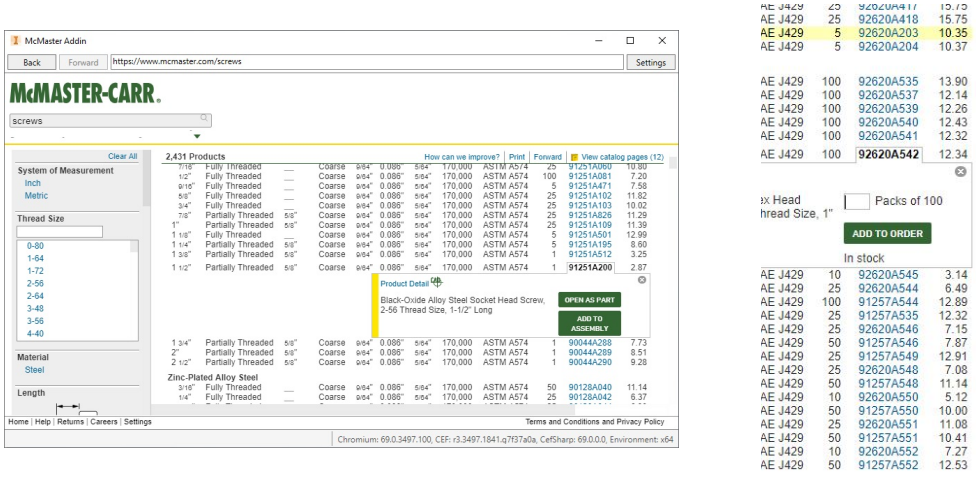

Figure 4: The alteration of the McMaster webpage for new model saving interactions(new on left, old on right)

7 Classes

7.1 StandardAddInServer

This class is responsible for all Add-In initialization, such as hooking into the primary application and calling for a new instance of the McMasterButton to be added into the ribbon interface within Inventor. Currently it contains the method GetSource() as well which stores webpage html to memory in order to parse for a file location, of which can be returned to an instance of the McMasterImporter class for conversion from a .stp file to .ipt, something Inventor can work with. In order to obtain the html from a dynamically created javascript enabled webpage, an instance of HeadlessWebBrowser is used to render the webpage silently.

7.2 HeadlessWebBrowser

This class is a background process version of the CefSharp utility. It allows for a silent operation of visiting webpages. This enables the software to obtain the html of the "part" page, which contains the url of the 3D model location.

7.3 McMasterButton

This class is a template for the button that StandardAddInServer adds to the Inventor interface. Upon clicking the button, an embedded web-browser will be open, MainWindow, to allow for a user to browse the McMaster catalog for a part they wish to obtain the 3D model of.

7.4 MainWindow

The MainWindow class is essentially a form that acts as a holder for MainView class and SettingsView class.

7.5 MainView

This class is the primary interface a user will see while using MAFI. The McMasterButton will create an instance of this web-browser form. Javascript is injected into the web-browser that dynamically adds "Open as Part" and "Add to Assembly" buttons within the webpage, See Appendix B/addButtonScripts.js for code that is ran via the chromium embedded browser. Clicking these buttons will execute methods to begin the importing process, which is handled by the McMasterImporter, owned by the StandardAddInServer. This class also contains a button for opening the SettingsView class UI.

7.6 SettingsView

The SettingsView class allows for the user to edit the add-in config file via a set of text boxes. It includes a save and cancel button, both of which return to the MainView.

7.7 McMasterImporter

This class locates the Inventor translator feature that allows for .stp to .ipt conversion. Then when called will silently convert a file that was obtained from the McMaster catalog, from MainWindow.

7.8 JavaScriptInteractionObj

This class is a binding object that is sent from the MainView class to the CefSharp chromium webbrowser as a new javascript object. It allows for callbacks to the .NET underlying application. Essentially this class allows for the new buttons on the McMaster webpage to call to the add-in.

8 Extra Notes

- CEFSharp v69.0 specifically. v69.0 is built into Inventor 2020, though .WPF package is not, so that may need to be included in release of .dll's, unless it can be embedded.
- Possible revisions after all other features:
  - Take product information from online catalog and automatically import into Inventor part file.
  - Implement a bill of materials export to send all McMaster items onto clipboard with appropriate QTY per assembly.
  - Incorporate a "no model" warning for items that do not have corresponding 3d models.
  - Option for choosing folder to save models per project, or per item.
  - Always on top windowing, currently can end up behind, which leads to difficulties.
  - Comments throughout code for future revising, if ever McMaster online changes.
  - If final release available, contact McMaster to check usability, possible copyright or anything legal.
  - Possibly develop for use with Solidworks, a similar program by Dassault Systèmes.
- Current Issues:
  - Web browser seems to lag on initial startup.
  - During testing, there were cases of crashing while importing. Possibly a threading error.

9 References

[1] Autodesk, Inc., "Inventor | Mechanical Design And 3D CAD Software | Autodesk," https://www.autodesk.com/products/inventor/overview.

[2] McMaster-Carr Supply Company, "McMaster-Carr," https://www.mcmaster.com.

[3] Microsoft Corporation, ".NET Documentation | Microsoft Docs," https://docs.microsoft.com/en-us/dotnet/.

[4] A. Maitland, "CefSharp," https://cefsharp.github.io, v69.0.

[5] N. Quandt, "MAFI," https://github.com/nquandt/MAFI, v1.0.4.

10 Appendix A: Story Cards

101 As a user I want to be able to use Windows operating system.
Acceptance Criteria: Runs on Windows 64-bit without errors.

102 As a user I want to use Autodesk Inventor 2020.
Acceptance Criteria: Uses Autodesk Inventor 2020 without errors.

103 As a user I want to have access to McMaster-Carr catalog from within Inventor.
Acceptance Criteria: No outside browser needed to access catalog, embed chromium into Inventor.

104 As a user I don't want the interface to change much.
Acceptance Criteria: Uses original McMaster webpage layout, and original Inventor UI, except for new button.

105 As a user I should be able to add parts from McMaster right into my project.
Acceptance Criteria: No outside program or steps needed to import file, all done via Inventor, or windows, in background.

106 As a user I want to be able to choose whether to add a part to my current assembly or as an individual part.
Acceptance Criteria: Two individual buttons for "Add to Assembly" and "Open as Part".

107 As a user I would like properties like the material added to my file for me.
Acceptance Criteria: Properties are auto-populated on file addition.

108 As a user I would like to have control over the saving location of files imported from McMaster.
Acceptance Criteria: Incorporate a settings page/config file, for the addin that allows for options such as saving location, file naming conventions.

109 As a developer I would like to use C#
Acceptance Criteria: All code written with C# on .NET 4.7.2 or newer

110 As a developer I shouldn't need to install too many files or packages on the final computer.
Acceptance Criteria: Keep total file count below 3.

111 As a developer I should well document code to allow for future updates.
Acceptance Criteria: An outside individual can proof read code and follow along without difficulty.

11 Appendix B: Code

All code can be found on the MAFI github repository[5].

StandardAddInServer.cs

```
using System;
using System.IO;
using reflect = System.Reflection;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Drawing;
using Inv = Inventor;//two using statements to allow for
   disambiguation
//between Inventor and System.IO namespace at times
using Inventor;
using CefSharp.Wpf;
using CefSharp;

#pragma warning disable IDE1006 //naming rules
namespace McMasterAddin
{
  /// <summary>
  /// This is the primary AddIn Server class that implements the
  /// ApplicationAddInServer interface that all Inventor AddIns
  /// are required to implement. The communication between
     Inventor
  /// and the AddIn is via the methods on this interface.
  /// </summary>
  [GuidAttribute("4989fc73-4710-47df-9034-98d770e68fbb")]
  public class StandardAddInServer : ApplicationAddInServer
  {
    #region ObjectInitialization
    private static readonly GuidAttribute m_ClientID =
      (GuidAttribute)System.Attribute.GetCustomAttribute(
        typeof(StandardAddInServer), typeof(GuidAttribute));
    public static readonly string m_ClientIDstr =
      "{" + m_ClientID.Value + "}";

    private bool wasOff = false;

    // Inventor application object.
    public Inv.Application m_invApp;
    private HeadlessWebBrowser _headlessBrowser;

    public static readonly string urlBase =
       "https://www.mcmaster.com/";
    //public System.Collections.Generic.List<string> fileList =
       new System.Collections.Generic.List<string>();
    public System.Collections.Generic.Dictionary<string, string>
       fileList = new
```

```
      System.Collections.Generic.Dictionary<string, string>();

  //single button for McMaster Catalog
  private McMasterButton m_Button;
  private McMasterImporter m_Importer;
  //user interface event
  private UserInterfaceEvents m_UIEvents;

  private
     UserInterfaceEventsSink_OnResetRibbonInterfaceEventHandler
      UIESink_OnResetRibbonInterfaceEventDelegate;

  #endregion

  public StandardAddInServer()
  {
    //Keep empty
  }

  #region ApplicationAddInServer Members

  public void Activate(ApplicationAddInSite addInSiteObject,
     bool firstTime)
  {
    try
    {
      //the Activate method is called by Inventor when it
         loads the addin
      //the AddInSiteObject provides access to the Inventor
         Application
      //object the FirstTime flag indicates if the addin is
         loaded for the
      //first time

      //initialize AddIn members
      m_invApp = addInSiteObject.Application;
      m_Importer = new McMasterImporter(this);

      if (Properties.Settings.Default.projectFolder == "")
      {
        Properties.Settings.Default.projectFolder =
          m_invApp.DesignProjectManager.ActiveDesignProject
          .WorkspacePath + "\\MCMASTER_REPOSITORY\\";
        Properties.Settings.Default.Save();
      }
      //initialize event delegates
      m_UIEvents =
         m_invApp.UserInterfaceManager.UserInterfaceEvents;
```

```
        UIESink_OnResetRibbonInterfaceEventDelegate = new
    UserInterfaceEventsSink_OnResetRibbonInterfaceEventHandler(
            UIE_OnResetRibbonInterface);
        m_UIEvents.OnResetRibbonInterface +=
          UIESink_OnResetRibbonInterfaceEventDelegate;

        m_Button = new McMasterButton(this);

        if (firstTime == true)
        {
          //access user interface manager
          UserInterfaceManager UIManager =
             m_invApp.UserInterfaceManager;

          //create the UI for classic interface
          if (UIManager.InterfaceStyle ==
             InterfaceStyleEnum.kClassicInterface)
          {
            //For first iterations assume RibbonInterface
          }
          //create the UI for ribbon interface
          else if (UIManager.InterfaceStyle ==
            InterfaceStyleEnum.kRibbonInterface)
          {
            CreateOrUpdateRibbonUserInterface();
          }
        }

        InitializeCEF();
      }
      catch (Exception e)
      {
        MessageBox.Show(e.ToString());
      }
    }

    public void Deactivate()
    {
      //Need to call on main thread
      if (wasOff){
        Cef.Shutdown();
      }
    }

    public void ExecuteCommand(int commandID)
    {
      // Note:this method is now obsolete, you should use the
      // ControlDefinition functionality for implementing
         commands.
```

```
    }

    public object Automation
    {
      // This property is provided to allow the AddIn to expose
         an API
      // of its own to other programs. Typically, this  would be
         done by
      // implementing the AddIn's API interface in a class and
         returning
      // that class object through this property.

      get
      {
        // TODO: Add ApplicationAddInServer.Automation getter
           implementation
        return null;
      }
    }

    #endregion

    #region Event Handlers

    private void CreateOrUpdateRibbonUserInterface()
    {
      m_Button.AddToUI();
    }

    private void UIE_OnResetRibbonInterface(NameValueMap context)
    {
      CreateOrUpdateRibbonUserInterface();
    }

    #endregion

    public void PreLoadStepFile(string pNumber, int open)
    {
      if (!fileList.ContainsKey(pNumber))
      {
        fileList.Add(pNumber, "initialized");

        string savingDirectory =
           Properties.Settings.Default.projectFolder;
        foreach (string s in
           System.IO.Directory.GetFiles(savingDirectory))
        {
          if (s.Substring(savingDirectory.Length)
            .Contains(pNumber))
```

```
        {
          if (s.Contains(".ipt"))
          {
            fileList[pNumber] = "exists:" + s;
            return;
          }
        }
      }

      //Load Offscreen browser to partNumber webpage, to
         extract file locations
      int tries = 0;
      string fileName = "";
    retry:
      if (tries < 3)
      {
        string url = urlBase + pNumber;
        _headlessBrowser.OpenUrl(url);
        var tS =
           _headlessBrowser.Page.EvaluateScriptAsync(@"var a =
           'empty';for (let i of
           document.getElementsByClassName('li--cad')){if
           (i.dataset.mcmCadOption.includes('STEP')){a =
           i.dataset.mcmCadOption;}} a;");
        tS.Wait();
        JavascriptResponse response = tS.Result;
        var result = response.Success ? (response.Result ??
           "null") : response.Message;
        fileName = (string)result;
        if (!fileName.Contains(pNumber))
        {
          tries++;
          System.Threading.Thread.Sleep(500);
          goto retry;
        }
        else
        {
          fileList[pNumber] = urlBase + fileName.Substring(1);
          System.Diagnostics.Debug.WriteLine("url: " +
             fileName);
          fileName = ReverseString(fileName);
          fileName = ReverseString(fileName.Substring(0,
             fileName.IndexOf('/')));
          System.Diagnostics.Debug.WriteLine("good to go: " +
             fileName);
          if (Directory.Exists(savingDirectory))
          {
            using (var client = new System.Net.WebClient())
            {
```

```
                string filePath =
                   System.IO.Path.Combine(savingDirectory,
                   fileName);//Saving Directory for .step temp
                   file
                System.Diagnostics.Debug.WriteLine(filePath);
                System.Net.ServicePointManager.SecurityProtocol =
                  System.Net.SecurityProtocolType.Tls |
                  System.Net.SecurityProtocolType.Tls11 |
                  System.Net.SecurityProtocolType.Tls12;
                fileList[pNumber] = "beginDownload:" +
                   fileList[pNumber];
                client.DownloadFile(new
                   System.Uri(fileList[pNumber]
                  .Substring("beginDownload:".Length)),
                     filePath);
                fileList[pNumber] = "isDownloaded:" +
                   fileName.Length.ToString("X4") + filePath;
                fileList[pNumber] = "exists:" +
                   m_Importer.Translate(fileList[pNumber]
                  .Substring("isDownloaded:".Length));
                if (open != 0)
                {
                  bool isAssembly = true;
                  if (open == 2)
                  {
                    isAssembly = false;
                  }
                  m_Importer.Open(fileList[pNumber]
                    .Substring("exists:".Length), isAssembly);
                }
              }
            }
          }
          else
          {
            fileList.Remove(pNumber);
            MessageBox.Show("Couldn't retrieve " + pNumber);
          }
        }
      }

      public void GetSource(string pNumber, bool isAssembly)
      {
        if (fileList.ContainsKey(pNumber))
        {
          int tries = 0;
        retry:
          if (tries < 20)
```

```
        {
          if (fileList[pNumber].Contains("exists"))
          {
            m_Importer.Open(fileList[pNumber]
              .Substring("exists:".Length), isAssembly);
          }

        }
        else {
          if (fileList[pNumber].Contains("isDownloaded"))
          {
            tries = 0;
          }
              tries++;
              System.Threading.Thread.Sleep(500);
              goto retry;
        }
      }
      else
      {
        int open = 2;
        if (isAssembly)
        {
          open = 1;
        }
        PreLoadStepFile(pNumber, open);
      }
    }

    /// <summary>
    /// A method to reverse a string type by
    /// converting to charArray and using char[].Reverse method
    /// </summary>
    /// <param name="s">The string that you want to reverse the
       order of</param>
    /// <returns>A string with reversed character order</returns>
    public static string ReverseString(string s)
    {
      char[] arr = s.ToCharArray();
      Array.Reverse(arr);
      return new string(arr);
    }

    private void InitializeCEF()
    {
      //Keep CEF on until INVENTOR exits, not just the WPF form.
      if (!Cef.IsInitialized) {
        CefSharpSettings.ShutdownOnExit = false;
```

```
        var settings = new CefSettings { RemoteDebuggingPort =
            8088 };
    //    Example of setting a command line argument
  //      Enables WebRTC
        settings.CefCommandLineArgs.Add("enable-media-stream",
            "1");
        //Must call once on main thread, and shutdown on main
            thread.
        Cef.Initialize(settings);
        wasOff = true;
      }
      CefSharpSettings.LegacyJavascriptBindingEnabled = true;
      _headlessBrowser = new HeadlessWebBrowser();
    }
    public void CleanupTempFiles()
    {
      try
      {
        foreach (System.Collections.Generic.KeyValuePair<string,
            string> s in fileList)
        {
          if (s.Value.Contains("isDownloaded"))
          {
            if (System.IO.File.Exists(
              s.Value.Substring("isDownloaded:XXXX".Length)))
            {
              System.IO.File.Delete(
                s.Value.Substring("isDownloaded:XXXX".Length));
            }
          }
        }
      }
      catch { }
    }
  }

  public sealed class PictureDispConverter

  {
    [DllImport("OleAut32.dll",
        EntryPoint = "OleCreatePictureIndirect",
        ExactSpelling = true,
        PreserveSig = false)]

    private static extern stdole.IPictureDisp
        OleCreatePictureIndirect(
            [MarshalAs(UnmanagedType.AsAny)] object picdesc,
            ref Guid iid,
            [MarshalAs(UnmanagedType.Bool)] bool fOwn);
```

```
    static Guid iPictureDispGuid =
       typeof(stdole.IPictureDisp).GUID;

    private static class PICTDESC
    {
      //Picture Types
      public const short PICTYPE_UNINITIALIZED = -1;
      public const short PICTYPE_NONE = 0;
      public const short PICTYPE_BITMAP = 1;
      public const short PICTYPE_METAFILE = 2;
      public const short PICTYPE_ICON = 3;
      public const short PICTYPE_ENHMETAFILE = 4;

      [StructLayout(LayoutKind.Sequential)]
      public class Icon
      {
        internal int cbSizeOfStruct =
            Marshal.SizeOf(typeof(PICTDESC.Icon));
        internal int picType = PICTDESC.PICTYPE_ICON;
        internal IntPtr hicon = IntPtr.Zero;
        internal int unused1;
        internal int unused2;

        internal Icon(System.Drawing.Icon icon)
        {
          this.hicon = icon.ToBitmap().GetHicon();
        }
      }

      [StructLayout(LayoutKind.Sequential)]
      public class Bitmap
      {
        internal int cbSizeOfStruct =
            Marshal.SizeOf(typeof(PICTDESC.Bitmap));

        internal int picType = PICTDESC.PICTYPE_BITMAP;
        internal IntPtr hbitmap = IntPtr.Zero;
        internal IntPtr hpal = IntPtr.Zero;
        internal int unused;

        internal Bitmap(System.Drawing.Bitmap bitmap)
        {
          this.hbitmap = bitmap.GetHbitmap();
        }
      }
    }

    public static stdole.IPictureDisp ToIPictureDisp(
```

```
        System.Drawing.Icon icon)
    {
      PICTDESC.Icon pictIcon = new PICTDESC.Icon(icon);

      return OleCreatePictureIndirect(
          pictIcon, ref iPictureDispGuid, true);
    }

    public static stdole.IPictureDisp ToIPictureDisp(
        System.Drawing.Bitmap bmp)
    {

      PICTDESC.Bitmap pictBmp = new PICTDESC.Bitmap(bmp);

      return OleCreatePictureIndirect(pictBmp, ref
         iPictureDispGuid, true);
    }
  }
}
```

HeadlessWebBrowser.cs

```
using System.Threading;
using System;
using CefSharp;
using CefSharp.OffScreen;

namespace McMasterAddin
{
  public class HeadlessWebBrowser
  {

    /// <summary>
    /// The browser page
    /// </summary>
    public ChromiumWebBrowser Page { get; private set; }
    /// <summary>
    /// The request context
    /// </summary>
    public RequestContext RequestContext { get; private set; }

    // chromium does not manage timeouts, so we'll implement one
    private ManualResetEvent manualResetEvent = new
       ManualResetEvent(false);

    public HeadlessWebBrowser()
    {
      RequestContext = new RequestContext();
      Page = new ChromiumWebBrowser("", null, RequestContext);
      PageInitialize();
    }

    /// <summary>
    /// Open the given url
    /// </summary>
    /// <param name="url">the url</param>
    /// <returns></returns>
    public void OpenUrl(string url)
    {
      try
      {
        Page.LoadingStateChanged += PageLoadingStateChanged;
        if (Page.IsBrowserInitialized)
        {
          Page.Load(url);

          //create a 60 sec timeout
          bool isSignalled =
             manualResetEvent.WaitOne(TimeSpan.FromSeconds(60));
          manualResetEvent.Reset();
```

```
            //As the request may actually get an answer, we'll
               force stop when the timeout is passed
            if (!isSignalled)
            {
              Page.Stop();
            }
          }
        }
        catch (ObjectDisposedException)
        {
          //happens on the manualResetEvent.Reset(); when a
             cancelation token has disposed the context
        }
        Page.LoadingStateChanged -= PageLoadingStateChanged;
      }

      /// <summary>
      /// Manage the IsLoading parameter
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void PageLoadingStateChanged(object sender,
         LoadingStateChangedEventArgs e)
      {
        // Check to see if loading is complete - this event is
           called twice, one when loading starts
        // second time when it's finished
        if (!e.IsLoading)
        {
          manualResetEvent.Set();
        }
      }

      /// <summary>
      /// Wait until page initialization
      /// </summary>
      private void PageInitialize()
      {
        SpinWait.SpinUntil(() => Page.IsBrowserInitialized);
      }
    }
}
```

McMasterButton.cs

```
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Inventor;
using System.Drawing;
using System;

namespace McMasterAddin
{
  class McMasterButton
  {
    private StandardAddInServer _stAddIn;

    public ButtonDefinition m_buttonDefinition;

    private MainWindowViewModel mv;

    private ButtonDefinitionSink_OnExecuteEventHandler
      m_button_Definition_OnExecute_Delegate;

    public McMasterButton(StandardAddInServer s)
    {
      _stAddIn = s;
       mv = new MainWindowViewModel();
      Stream myStream = System.Reflection.Assembly.
        GetExecutingAssembly().GetManifestResourceStream(
        "McMasterAddin.Resources.mcmaster.ico");

      stdole.IPictureDisp largeImage =
        PictureDispConverter.ToIPictureDisp(new Icon(myStream));

      //Button definition
      m_buttonDefinition = _stAddIn.m_invApp.CommandManager.
        ControlDefinitions.AddButtonDefinition("Browse",
        "BrowseButton",
        CommandTypesEnum.kQueryOnlyCmdType,
           StandardAddInServer.m_ClientIDstr,
        "Browse McMaster-Carr Inventory", "Use this to find " +
        "hardware and other products available on McMaster.com",
        largeImage, largeImage,
           ButtonDisplayEnum.kAlwaysDisplayText);

      m_button_Definition_OnExecute_Delegate = new
        ButtonDefinitionSink_OnExecuteEventHandler(
        m_button_OnExecute);
      m_buttonDefinition.OnExecute +=
```

```
        m_button_Definition_OnExecute_Delegate;
      m_buttonDefinition.Enabled = true;
    }

    public void AddToUI()
    {

      UserInterfaceManager UIManager =
        _stAddIn.m_invApp.UserInterfaceManager;

      Ribbon assemblyRibbon = UIManager.Ribbons["Assembly"];
      RibbonTab assembleTab =
         assemblyRibbon.RibbonTabs["id_TabAssemble"];
      bool exists = false;
      foreach (RibbonPanel r in assembleTab.RibbonPanels)
      {
        if (r.InternalName == "McMasterPanelAssembly")
        {
          exists = true;
        }
      }
      if (!exists)
      {
        RibbonPanel mcMasterPanel =
          assembleTab.RibbonPanels.Add("McMaster-Carr",
            "McMasterPanelAssembly",
              StandardAddInServer.m_ClientIDstr);

        mcMasterPanel.CommandControls
          .AddButton(m_buttonDefinition, true);
      }
      Ribbon partRibbon = UIManager.Ribbons["Part"];

      RibbonTab partTab = partRibbon.RibbonTabs["id_TabManage"];
      exists = false;
      foreach (RibbonPanel r in partTab.RibbonPanels)
      {
        if (r.InternalName == "McMasterPanelPart")
        {
          exists = true;
        }
      }
      if (!exists)
      {
        RibbonPanel mcMasterPanel =
          partTab.RibbonPanels.Add("McMaster-Carr",
            "McMasterPanelPart",
              StandardAddInServer.m_ClientIDstr);
```

```
        mcMasterPanel.CommandControls
          .AddButton(m_buttonDefinition, true);
      }

    }

    /// <summary>
    /// This is the mcMaster-addin button execution
    /// </summary>
    /// <param name="Context"></param>
    private void m_button_OnExecute(NameValueMap Context)
    {
      //Refer to MainWindow.xaml for code of the browser
         extension
      var wpfWindow = new McMasterAddin.MainWindow(_stAddIn);
      //This allows for a WPF control to be displayed without
      //the need of a fullfledge WPF Application.

      var helper = new System.Windows
        .Interop.WindowInteropHelper(wpfWindow)
      {
        Owner = new IntPtr(_stAddIn.m_invApp.MainFrameHWND)
      };
      wpfWindow.DataContext = mv;
      wpfWindow.ShowDialog();
    }
  }
}
```

MainWindow.xaml.cs

```
using System.Threading;
using System.Windows;
using System.Windows.Input;
using System;
using CefSharp;
using System.Text.RegularExpressions;
using System.ComponentModel;
using System.Diagnostics;
using System.Collections.Generic;

namespace McMasterAddin
{
  /// <summary>
  /// Interaction logic for MainWindow.xaml
  /// </summary>
  ///
  public partial class MainWindow : Window
  {
    private StandardAddInServer _stAddIn;

    public StandardAddInServer StandardAddInServer
    {
      get { return _stAddIn; }
      set { _stAddIn = value; }
    }

    public MainWindow(StandardAddInServer a)
    {
      InitializeComponent();
      _stAddIn = a;
    }

    private void Window_Closing(object sender,
      System.ComponentModel.CancelEventArgs e)
    {
      Properties.Settings.Default.heightClosed =
         this.ActualHeight;
      Properties.Settings.Default.widthClosed = this.ActualWidth;
      Properties.Settings.Default.Save();
      _stAddIn.CleanupTempFiles();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
      this.Height = Properties.Settings.Default.heightClosed;
      this.Width = Properties.Settings.Default.widthClosed;
    }
  }
```

```
public class RelayCommand<T> : ICommand
{
  private readonly Predicate<T> _canExecute;
  private readonly Action<T> _execute;

  public RelayCommand(Action<T> execute)
     : this(execute, null)
  {
    _execute = execute;
  }

  public RelayCommand(Action<T> execute, Predicate<T>
     canExecute)
  {
    if (execute == null)
    {
      throw new ArgumentNullException("execute");
    }
    _execute = execute;
    _canExecute = canExecute;
  }

  public bool CanExecute(object parameter)
  {
    return _canExecute == null || _canExecute((T)parameter);
  }

  public void Execute(object parameter)
  {
    _execute((T)parameter);
  }

  public event EventHandler CanExecuteChanged
  {
    add { CommandManager.RequerySuggested += value; }
    remove { CommandManager.RequerySuggested -= value; }
  }
}

public class RelayCommand : ICommand
{
  private readonly Predicate<object> _canExecute;
  private readonly Action<object> _execute;

  public RelayCommand(Action<object> execute)
     : this(execute, null)
  {
    _execute = execute;
```

```
    }

    public RelayCommand(Action<object> execute,
       Predicate<object> canExecute)
    {
      if (execute == null)
      {
        throw new ArgumentNullException("execute");
      }
      _execute = execute;
      _canExecute = canExecute;
    }

    public bool CanExecute(object parameter)
    {
      return _canExecute == null || _canExecute(parameter);
    }

    public void Execute(object parameter)
    {
      _execute(parameter);
    }

    // Ensures WPF commanding infrastructure asks all
       RelayCommand objects whether their
    // associated views should be enabled whenever a command is
       invoked
    public event EventHandler CanExecuteChanged
    {
      add
      {
        CommandManager.RequerySuggested += value;
        CanExecuteChangedInternal += value;
      }
      remove
      {
        CommandManager.RequerySuggested -= value;
        CanExecuteChangedInternal -= value;
      }
    }

    private event EventHandler CanExecuteChangedInternal;

    public void RaiseCanExecuteChanged()
    {
      CanExecuteChangedInternal.Raise(this);
    }
  }
```

```
public static class EventRaiser
{
  public static void Raise(this EventHandler handler, object
     sender)
  {
    handler?.Invoke(sender, EventArgs.Empty);
  }

  public static void Raise<T>(this EventHandler<EventArgs<T>>
     handler, object sender, T value)
  {
    handler?.Invoke(sender, new EventArgs<T>(value));
  }

  public static void Raise<T>(this EventHandler<T> handler,
     object sender, T value) where T : EventArgs
  {
    handler?.Invoke(sender, value);
  }

  public static void Raise<T>(this EventHandler<EventArgs<T>>
     handler, object sender, EventArgs<T> value)
  {
    handler?.Invoke(sender, value);
  }
}
public class EventArgs<T> : EventArgs
{
  public EventArgs(T value)
  {
    Value = value;
  }

  public T Value { get; private set; }
}

public abstract class BaseViewModel : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;

  protected void OnPropertyChanged(string propertyName)
  {
    VerifyPropertyName(propertyName);
    PropertyChanged?.Invoke(this, new
       PropertyChangedEventArgs(propertyName));
  }

  [Conditional("DEBUG")]
  private void VerifyPropertyName(string propertyName)
```

```
    {
      if (TypeDescriptor.GetProperties(this)[propertyName] ==
         null)
        throw new ArgumentNullException(GetType().Name + " does
           not contain property: " + propertyName);
    }
  }
  public static class Mediator
  {
    private static IDictionary<string, List<Action<object>>>
       pl_dict =
       new Dictionary<string, List<Action<object>>>();

    public static void Subscribe(string token, Action<object>
       callback)
    {
      if (!pl_dict.ContainsKey(token))
      {
        var list = new List<Action<object>>();
        list.Add(callback);
        pl_dict.Add(token, list);
      }
      else
      {
        bool found = false;
        foreach (var item in pl_dict[token])
          if (item.Method.ToString() ==
             callback.Method.ToString())
            found = true;
        if (!found)
          pl_dict[token].Add(callback);
      }
    }

    public static void Unsubscribe(string token, Action<object>
       callback)
    {
      if (pl_dict.ContainsKey(token))
        pl_dict[token].Remove(callback);
    }

    public static void Notify(string token, object args = null)
    {
      if (pl_dict.ContainsKey(token))
        foreach (var callback in pl_dict[token])
          callback(args);
    }
  }
}
```

McMasterImporter.cs

```
using Inventor;

namespace McMasterAddin
{
  public class McMasterImporter
  {
    private StandardAddInServer _stAddIn;
    private string translatorID = "";

    public McMasterImporter(StandardAddInServer s)
    {
      _stAddIn = s;
      GetTranslatorAddInID("Translator: STEP");
    }

    public string Translate(string substitutePathVal)
    {
      string strFilePath = substitutePathVal.Substring(4);
      string strFileName =
         strFilePath.Substring(strFilePath.Length -
         int.Parse(substitutePathVal.Substring(0, 4),
         System.Globalization.NumberStyles.HexNumber));
      strFileName = strFileName.Substring(0, strFileName.Length
         - 5);
      System.Diagnostics.Debug.WriteLine(strFileName + "///" +
         strFilePath);
      ApplicationAddIns oAddIns =
         _stAddIn.m_invApp.ApplicationAddIns;
      TranslatorAddIn oTransAddIn =
         (TranslatorAddIn)oAddIns.ItemById[translatorID];
      oTransAddIn.Activate();

      TransientObjects transObj =
         _stAddIn.m_invApp.TransientObjects;

      DataMedium file = transObj.CreateDataMedium();
      file.FileName = strFilePath;

      TranslationContext context =
         transObj.CreateTranslationContext();
      context.Type = IOMechanismEnum.kFileBrowseIOMechanism;

      NameValueMap options = transObj.CreateNameValueMap();

      bool oHasOpt = oTransAddIn.HasOpenOptions[file, context,
         options];

      oTransAddIn.Open(file, context, options, out object oDoc);
```

```
    Document doc = (Document)oDoc;
    _stAddIn.m_invApp.SilentOperation = true;
    string savingDirectory =
       Properties.Settings.Default.projectFolder;
    if (savingDirectory == "")
    {
      savingDirectory =
        _stAddIn.m_invApp.DesignProjectManager
        .ActiveDesignProject.WorkspacePath
        + "\\MCMASTER_REPOSITORY\\";
    }
    doc.SaveAs(savingDirectory + strFileName + ".ipt", false);
    if (System.IO.File.Exists(strFilePath))
    {
      System.IO.File.Delete(strFilePath);
    }
    _stAddIn.m_invApp.SilentOperation = false;
    doc.Close();
    return savingDirectory + strFileName + ".ipt";
  }

  public void Open(string filePath, bool isAssembly)
  {
    //Add the converted .ipt file into my active assembly
    if (isAssembly)
    {
      //Create an operation matrix that contains information
      //about starting position of my part.
      Matrix oMatrix = _stAddIn.m_invApp
        .TransientGeometry.CreateMatrix();
      oMatrix.SetTranslation(_stAddIn.m_invApp
        .TransientGeometry.CreateVector(), true);
      ((AssemblyDocument)_stAddIn.m_invApp.ActiveDocument)
        .ComponentDefinition.Occurrences.Add(filePath,
          oMatrix);
    }
    else
    {
      Document doc =
         _stAddIn.m_invApp.Documents.Open(filePath);
    }
  }

  private void GetTranslatorAddInID(string translatorName)
  {
    ApplicationAddIns oAddIns =
      _stAddIn.m_invApp.ApplicationAddIns;
    TranslatorAddIn oTransAddIn;
```

```
      foreach (ApplicationAddIn a in oAddIns)
      {
        if (a.AddInType ==
          ApplicationAddInTypeEnum.kTranslationApplicationAddIn)
        {
          oTransAddIn = (TranslatorAddIn)a;
          if (oTransAddIn.DisplayName == translatorName)
          {
            translatorID = oTransAddIn.ClassIdString;
          }
        }
      }
    }
  }
}
```

MainView.xaml.cs

```
using System.Threading;
using System.Windows;
using CefSharp;
using System.Text.RegularExpressions;
using System.Windows.Controls;
using McMasterAddin;
using System.Windows.Input;

namespace McMasterAddin
{
  /// <summary>
  /// Interaction logic for MainView.xaml
  /// </summary>
  public partial class MainView : UserControl
  {
    private MainWindow _myWindow;
    // chromium does not manage timeouts, so we'll implement one
    //private ManualResetEvent manualResetEvent = new
       ManualResetEvent(false);

    private string currentURL = "";
    //private string currentSource = "";
    private string s = "";
    JavaScriptInteractionObj jsObj;
    public MainView()
    {
      InitializeComponent();
      jsObj = new JavaScriptInteractionObj();
      Browser.RegisterAsyncJsObject("mainWindowOBJ",jsObj);
      currentURL = (string)Browser.Address.Clone();
      using (System.IO.Stream myStream = System.Reflection
        .Assembly.GetExecutingAssembly()
         .GetManifestResourceStream(
          "McMasterAddin.Resources.addButtonScripts.js"))
      {
        using (System.IO.StreamReader sRdr =
          new System.IO.StreamReader(myStream))
        {
          s = sRdr.ReadToEnd();
        }
      }
    }

    /// <summary>
    /// Manage the IsLoading parameter
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
```

```
    private void PageLoadingStateChanged(object sender,
      LoadingStateChangedEventArgs e)
    {
      if (!e.IsLoading)
      {
        Dispatcher.Invoke(new System.Action(() =>
        {
          if (currentURL != Browser.Address)
          {
            Browser.ExecuteScriptAsync(s);
            currentURL = (string)Browser.Address.Clone();
          }
        }));
      }
    }

    private void UserControl_Loaded(object sender,
       RoutedEventArgs e)
    {
      if (IsInitialized)
      {
        Browser.LoadingStateChanged += PageLoadingStateChanged;
        _myWindow = (MainWindow)((Grid)((ContentControl)
          ((ContentPresenter)this.TemplatedParent)
          .TemplatedParent).Parent).Parent;
        jsObj._mW = _myWindow;
      }
    }
  }

  public class UserControl1ViewModel : BaseViewModel,
     IPageViewModel
  {
    private ICommand _goTo2;

    public ICommand GoTo2
    {
      get
      {
        return _goTo2 ?? (_goTo2 = new RelayCommand(x =>
        {
          Mediator.Notify("GoTo2Screen", "");
        }));
      }
    }
  }
  public class JavaScriptInteractionObj
  {
    public MainWindow _mW;
```

```
    public JavaScriptInteractionObj()
    {

    }
    public void PreLoadStep(string pNumber)
    {
      _mW.StandardAddInServer.PreLoadStepFile(
        pNumber.Replace("partNumber", ""),0);
    }
    public void AddToAssembly(string pNumber)
    {
      //MessageBox.Show(pNumber);
      _mW.StandardAddInServer.GetSource(
        pNumber.Replace("partNumber", ""), true);
    }

    public void OpenAsPart(string pNumber)
    {
      //MessageBox.Show(pNumber);
      _mW.StandardAddInServer.GetSource(
        pNumber.Replace("partNumber", ""), false);
      _mW.Dispatcher.BeginInvoke(new System.Action(() =>
      {
        _mW.Close();
      }));
    }
  }

}
```

SettingsView.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace McMasterAddin
{
  /// <summary>
  /// Interaction logic for SettingsControl.xaml
  /// </summary>
  public partial class SettingsView : UserControl
  {

    public SettingsView()
    {
      InitializeComponent();
      projectFolderTextBlock.Text =
         Properties.Settings.Default.projectFolder;
    }

    private void saveButton_Click(object sender, RoutedEventArgs
       e)
    {
      string s = projectFolderTextBlock.Text;
        if (s.Substring(s.Length-1,1) != "\\")
      {
        s += "\\";
      }
      Properties.Settings.Default.projectFolder = s;
      Properties.Settings.Default.Save();
      if (cancelButton.Command.CanExecute(
        cancelButton.CommandParameter)){
        cancelButton.Command.Execute(
          cancelButton.CommandParameter);
      }
    }

    private void UserControl_Loaded(object sender,
```

```
        RoutedEventArgs e)
    {
      if (IsInitialized)
      {
      }
    }
  }

  public class UserControl2ViewModel : BaseViewModel,
     IPageViewModel
  {
    private ICommand _goTo1;

    public ICommand GoTo1
    {
      get
      {
        return _goTo1 ?? (_goTo1 = new RelayCommand(x =>
        {
          Mediator.Notify("GoTo1Screen", "");
        }));
      }
    }
  }
}
```

addButtonScripts.cs

```
var scriptAddButton = document.createElement('script');
scriptAddButton.type = 'text/javascript';
scriptAddButton.text = 'function callAddToAssembly(pNumber){
    mainWindowOBJ.addToAssembly(pNumber.parentElement.id);
}
function callOpenAsPart(pNumber){
    mainWindowOBJ.openAsPart(pNumber.parentElement.id);
}
function callPreLoadStep(p){
    console.log("hello");
    mainWindowOBJ.preLoadStep(p);
}
var mutationObserver = new MutationObserver(function (mutations)
{
    mutations.forEach(function (mutation)
    {
        var nodeIDZ = null;
        mutation.addedNodes.forEach(function (nodeX)
        {
            if (nodeX.hasChildNodes())
            {
                nodeX.childNodes.forEach(function (nodeY)
                {
                  try {
                     if (nodeY.id.includes("InLnOrd_ItmBxRw_1_"))
                      { nodeIDZ = nodeY; }
                  } catch(e) { }
                });
                try {
                    if (nodeX.id.includes("InLnOrd_ItmBxRw_1_"))
                    { nodeIDZ = nodeX; }
                } catch(e) { }
            }
        });
        if (nodeIDZ != null) {
        var partNumber =
           "partNumber".concat(nodeIDZ.id.substr(18));
        callPreLoadStep(partNumber);
        var nodeDiv = document.createElement("div");
        nodeDiv.id = partNumber;
var nodeSave =
   nodeIDZ.getElementsByClassName("InLnOrdWebPart_TransInfo")[0];
            var nodeParent = nodeSave.parentElement;
            nodeParent.replaceChild(nodeDiv, nodeSave);
            var node = document.getElementById(partNumber);
            var newAssembleButton =
               document.createElement("button");
            newAssembleButton.className =
```

```
                "button-add-to-order-inline add-to-order
                customButton";
            newAssembleButton.style = "font-size:11px;";
            var newSpan = document.createElement("span");
            newSpan.className = "button-reset--IE";
            newSpan.innerText = "add to assembly";
            newAssembleButton.appendChild(newSpan);
            newAssembleButton.onclick = function () {
                callAddToAssembly(this); };
            var newPartButton = document.createElement("button");
            newPartButton.className =
                "button-add-to-order-inline add-to-order
                customButton";
            newPartButton.style = "font-size:11px;";
            var newPSpan = document.createElement("span");
            newPSpan.className = "button-reset--IE";
            newPSpan.innerText = "open as part";
            newPartButton.appendChild(newPSpan);
            newPartButton.onclick = function () {
                callOpenAsPart(this); };
            node.appendChild(newPartButton);
            node.appendChild(newAssembleButton);
        }
    });
});
mutationObserver.observe(document.documentElement,
    {
        characterData: true,
        childList: true,
        subtree: true,
        characterDataOldValue: true
    });';
document.getElementsByTagName('head')[0].appendChild(scriptAddButton);
```