USER GUIDE

MADCAP FLARE 2024 r2

Plug-In API

MadCap Software
1660 17th Street, Suite 201
Denver, Colorado 80202
858-320-0387
www.madcapsoftware.com

THIS PDF WAS CREATED USING MADCAP FLARE.

CONTENTS

CHAPTER 1

Introduction 5
- What You Need 6
- How to Create a Plug-In 7
- How to Integrate a Plug-In Into Flare 9
- How to Enable the Plug-In Within Flare 10
- Interfaces and Enumerations 11
- Examples 12
- Best Practices/Guidelines 13

CHAPTER 2

Interfaces and Enumerations 14
- IPlugin Interface 15
- IHost Interface 16

CHAPTER 3

Examples 31
- Context Menu Example 32
- Controlled Language Example 33
- Ribbon Example 37
- Search and Change Text Style Example 40
- Toolbar Example 41

APPENDIX

PDFs 43

Tutorials 43

Cheat Sheets 44

User Guides 45

CHAPTER 1

Introduction

The MadCap Software plug-in API lets you integrate Flare with DLLs that you produce. For example, you might want to add customized ribbons, menus, or toolbar buttons to Flare.

Most of the information provided regarding the plug-in API is intended for developers or those who are quite familiar with APIs and DLLs.

This chapter discusses the following:

What You Need 6
How to Create a Plug-In 7
How to Integrate a Plug-In Into Flare 9
How to Enable the Plug-In Within Flare 10
Interfaces and Enumerations 11
Examples 12
Best Practices/Guidelines 13

What You Need

Here is what you need before you begin:

- Latest build of MadCap Flare
- B3.PluginAPIKit.dll
- Visual Studio 2010 or later

How to Create a Plug-In

1. In Visual Studio, create a new Class Library project.
2. In the new project dialog, select .Net Framework 4.0.
3. Add a library reference to the B3.PluginAPIKit.dll assembly.
4. Implement the IPlugin interface. See "IPlugin Interface" on page 15.

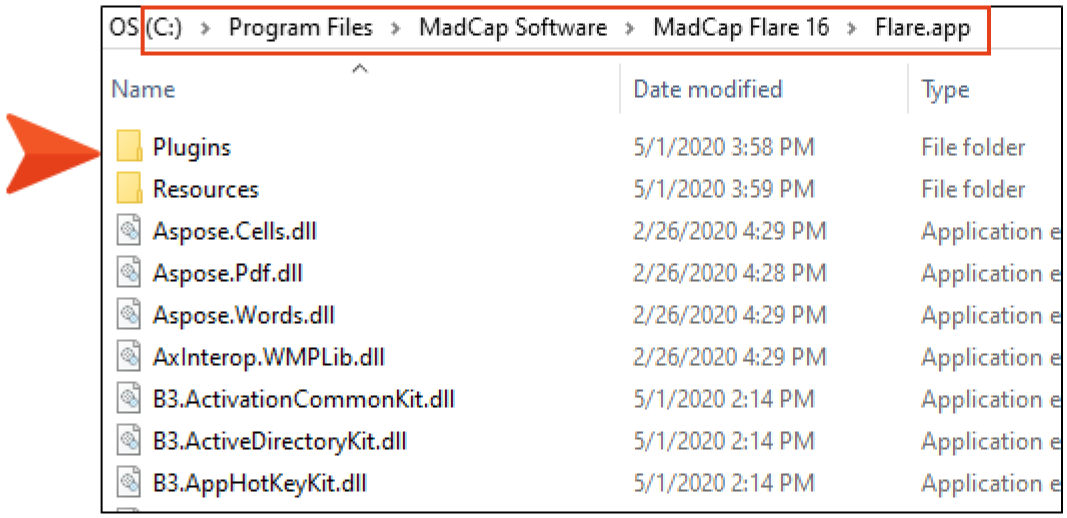
   Following is a basic example where the function of the plug-in is to simply show a message box whenever the plug-in is activated or deactivated.

```
using System;
using System.Windows.Forms;
using B3.PluginAPIKit;
namespace DemoPlugin
{
    public class DemoPlugin:IPlugin
    {
        private IHost mHost;
        private bool mActivated;
        public bool IsActivated
        {
            get { return mActivated; }
        }
        public string GetVersion()
        {
            return "1.0";
        }
        public string GetAuthor()
        {
             return "Bob Smith";
          }
          public string GetDescription()
          {
             return "Displays a message box.";
          }
          public string GetName()
          {
             Return "DemoPlugin";
          }
          public void Initialize(IHost host)
          {
             mHost = host;
          }
          public void Execute()
          {
             mActivated = true;
            MessageBox.Show(GetName() + " activated!");
        }
        public void Stop()
        {
            MessageBox.Show(GetName() + " deactivated!");
            mHost.Dispose();
            mActivated = false;
        }
    }
}
```

How to Integrate a Plug-In Into Flare

Flare monitors the Plugins folder under its root application directory. Any valid plug-in detected inside the Plugins folder will be listed on the Plugins tab of the Options dialog in Flare.

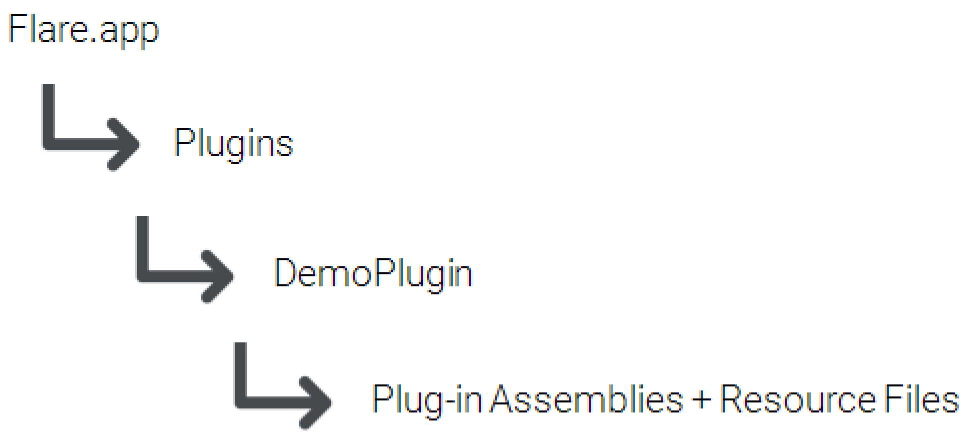
1. Make sure Flare is not open.
2. In Windows navigate to the Flare.app\Plugins folder where you have installed Flare (e.g., C:\Program Files\MadCap Software\MadCap Flare20\Flare.app\Plugins).



OS (C:) › Program Files › MadCap Software › MadCap Flare 16 › Flare.app

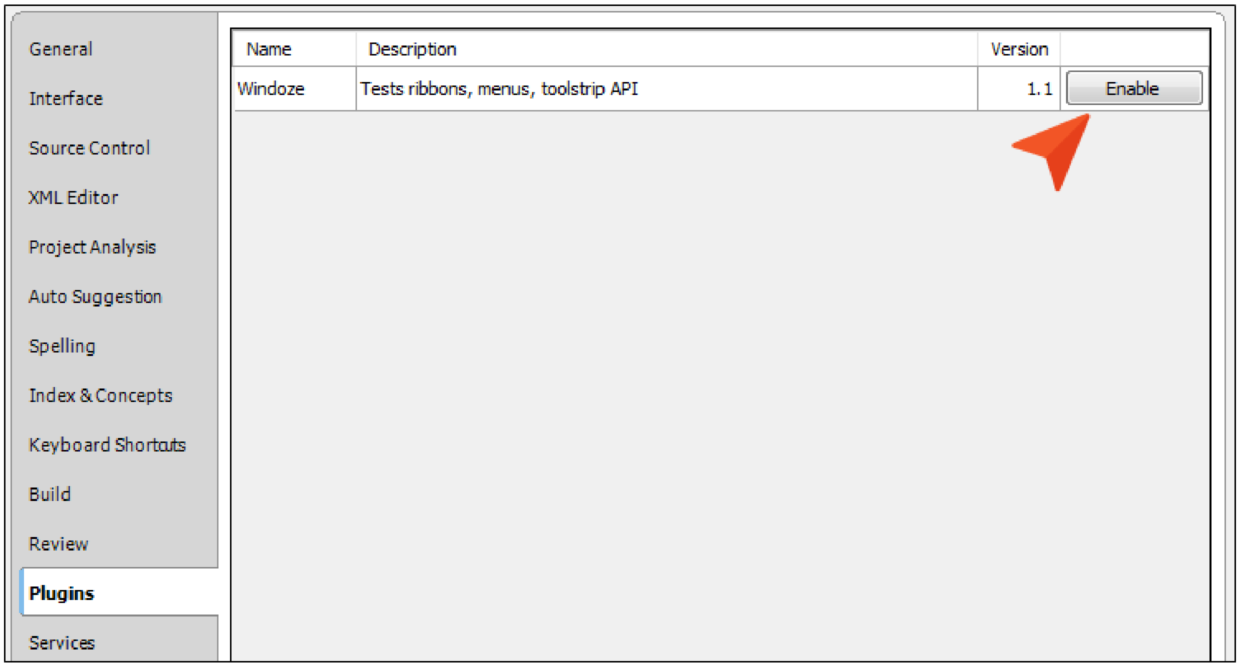
Name	Date modified	Type
Plugins	5/1/2020 3:58 PM	File folder
Resources	5/1/2020 3:59 PM	File folder
Aspose.Cells.dll	2/26/2020 4:29 PM	Application e
Aspose.Pdf.dll	2/26/2020 4:28 PM	Application e
Aspose.Words.dll	2/26/2020 4:29 PM	Application e
AxInterop.WMPLib.dll	2/26/2020 4:29 PM	Application e
B3.ActivationCommonKit.dll	5/1/2020 2:14 PM	Application e
B3.ActiveDirectoryKit.dll	5/1/2020 2:14 PM	Application e
B3.AppHotKeyKit.dll	5/1/2020 2:14 PM	Application e

3. Within the **Plugins** directory, create a new folder named after your plugin. For the example above, the plugin directory is named “DemoPlugin.”
4. Add your built plug-in assemblies and resource files into that directory. The directory hierarchy should look as follows:



How to Enable the Plug-In Within Flare

1. Launch Flare.
2. Select **File > Options**. The Options dialog opens.
3. Select the **Plugins** tab. You should see a row that represents your DLL.
4. Click **Enable**.



Name	Description	Version
Windoze	Tests ribbons, menus, toolstrip API	1.1

NOTE You can also use the Plugins tab in the Options dialog (**File > Options**) to disable a plugin.

5. Restart Flare.

Interfaces and Enumerations

For more details on the interfaces and enumerations with the plug-in API, see the following:

- "IPlugin Interface" on page 15
- "IHost Interface" on page 16
    - "IEditorContext Interface" on page 17
        - "IDocument Interface" on page 18
        - "ISelection Interface" on page 21
        - "EditorView Enumeration" on page 22
    - "INavContext Interface" on page 23
        - "ICustomToolBar Interface" on page 24
        - "IToolStripMenuItem Interface" on page 25
        - "IRibbon Interface" on page 25
        - "IRibbonControlData Interface" on page 26
        - "IRibbonMenuData Interface" on page 27
        - "IRibbonTab Interface" on page 27
        - "IRibbonGroup Interface" on page 28
        - "IRibbonComboBox Interface" on page 29
        - "IRibbonMenu Interface" on page 29
        - "RibbonIconSize Enumeration" on page 30

Examples

For examples of creating and working with the plug-in API, see the following:

- "Context Menu Example" on page 32
- "Controlled Language Example" on page 33
- "Ribbon Example" on page 37
- "Search and Change Text Style Example" on page 40
- "Toolbar Example" on page 41

Best Practices/Guidelines

Following are some best practices an guidelines to keep in mind as you work with the plug-in API.

Target Framework

- Currently, the Flare API framework only supports libraries targeted to .NET Framework 4.0.

Initialization

- IPlugin.Initialize(IHost) should only be used to set the instance of IHost passed in to a class variable.
- Calls to obtain the editor or navigation context should be done in IPlugin.Execute().

Cleaning Up

- Events should be properly detached.
- Changes made to the UI should be reverted back to its original state (e.g., ribbon, context menus, tool strip).
- IHost.Dispose() should be called in IPlugin.Stop(). This will dispose of most of the menu bar items and ribbon items added.
- The IPlugin.IsActivated property should be set to false.

References

- The working directory is the base application directory. If you are passing in a URL to any of the interface methods, relative paths are relative to the “Plugins” directory. For example, “DemoPlugin/Icons/Filter.png” maps to:

  C:/[Flare Install Path]/Flare.app/Plugins/DemoPlugin/Icons/Filter.png.

CHAPTER 2

Interfaces and Enumerations

There are several interfaces and enumerations involved with the plug-in API.

This chapter discusses the following:

IPlugin Interface 15
IHost Interface 16

IPlugin Interface

The IPlugin interface represents the plug-in. For the plug-in assembly to be considered valid, it must implement the IPlugin interface.

Properties

- *bool* **IsActivated** Gets a value indicating whether the plug-in is activated or not.

Methods

- *void* **Execute()** Activates the plug-in. This method is called when users click **Enable** on the Plugins tab of the Options dialog (**File > Options**).
- *string* **GetAuthor()** Returns the author of the plug-in.
- *string* **GetDescription()** Returns the description of the plug-in.
- *string* **GetName()** Returns the name of the plug-in.
- *string* **GetVersion()** Returns the version of the plug-in.
- *void* **Initialize(IHost)** Initializes the plug-in. An instance of IHost is passed in as a parameter, which gives users access to Flare components. See "IHost Interface" on the next page.
- *void* **Stop()** Deactivates the plug-in. This method is called when users click **Disable** on the Plugins tab of the Options dialog (**File > Options**).

IHost Interface

The IHost interface represents the host application. An instance of IHost is be passed in as a parameter to the Initialize() method in the IPlugin interface. See "IPlugin Interface" on the previous page.

Methods

- *string* GetCompany() Returns the name of the company.
- *IEditorContext* GetEditorContext() Returns the active IEditorContext object. See "IEditorContext Interface" on the next page.
- *string* GetName() Returns the name of the active application.
- *INavContext* GetNavContext() Returns the active INavContext object. See "INavContext Interface" on page 23.
- *void* Dispose() Cleans up the instance of IHost after usage. This must be called whenever the plug-in is disabled.

IEditorContext Interface

An instance of IEditorContext is returned using the GetEditorContext() method in the IHost interface (see "IHost Interface" on the previous page). IEditorContext gives access to opened Flare documents in the editor.

Events

- **DocumentSwitched** Occurs when the active document changes.

Methods

- *IDocument* **GetActiveDocument()** Returns the currently active IDocument. See "IDocument Interface" on the next page.
- *IEnumerable<IDocument>* **GetDocuments()** Returns a System.Collections.IEnumerable containing all the IDocument(s) currently open. See "IDocument Interface" on the next page.
- *XmlSchema* **GetMadCapSchema()** Returns the System.Xml.Shema.XmlSchema of available MadCap element names.
- *IDocument* **OpenDocument(string)** Opens the given file path and returns the associated IDocument. See "IDocument Interface" on the next page.
- *IDocument* **OpenDocument(string, EditorView)** Opens the given file path in the EditorView passed in and returns the associated IDocument. See "EditorView Enumeration" on page 22 and "IDocument Interface" on the next page.

IDocument Interface

The IDocument interface provides access to components associated with an editor document.

Properties

- *EditorView* CurrentEditorView Gets the current EditorView in focus. See "EditorView Enumeration" on page 22.
- *bool* EnableLocks Gets and sets whether locks are enabled in the current document .
- *ISelection* Selection Gets the ISelection of the currently selected text. See "ISelection Interface" on page 21.
- *bool* ShowChanges Gets and sets whether changes are shown in the current document.
- *bool* ShowLocks Gets and sets whether locks are shown in the current document.

Events

- CheckingIn Occurs when the editor document checks in to source control.
- Closing Occurs when the editor document is closing.
- CurrentEditorViewChanged Occurs when the editor view switches focus.
- EnableLocksChanged Occurs when the editor document EnableLocks property changes.
- KeyDown Occurs when a key is pressed down while the editor document has focus.
- KeyUp Occurs when a key is released while the editor document has focus.
- MouseClick Occurs when the editor document is clicked by the mouse.
- MouseDoubleClick Occurs when the editor document is double-clicked by the mouse.
- MouseHover Occurs when the editor document is hovered over by the mouse.
- MouseMove Occurs when the mouse moves over the editor document.
- OnContentChanged Occurs when content is changed in the editor document.
- PreviewKeyDown Occurs before the KeyDown event when a key is pressed while the document has focus.
- Printing Occurs when the Print or Print Preview command is invoked before the document prints.

- Saved Occurs when the editor document saves.
- SavedAs Occurs when the editor document runs the "Save As" command..
- Saving Occurs before the editor document saves.
- SavingsAs Occurs before the editor document runs the "Save As" command .
- ShowChangesChanged Occurs when the editor document ShowChanges property changes.
- ShowLocksChanged Occurs when the editor document ShowLocks property changes.

Methods

- *bool* ApplyStylesheet(string) Adds the referenced CSS stylesheet to the working list of styles. Paths may be absolute or relative.
- *void* Close() Closes the document.
- *void* EndOperation() Ends the current operation.
- *int* GetCursorPosition() Returns the linear position of the insertion point. The intial position starts from 0.
- *string* GetDocumentText() Returns the document text.
- *string* GetDocumentXml() Returns the xml of the document as text.
- *string* GetSourceUrl() Returns the url of the editor document.
- *List<ToolStripItem>* GetUserPendingContextMenuItems() Returns the list of System.Windows.Forms.ToolStripItem(s) that are pending to be added to the context menu.
- *XmlDocument* GetXmlDocument() Returns the System.Xml.XmlDocument of the document.
- *void* InsertDocumentNode(XmlNode, XmlNode, int, bool) Inserts a new System.Xml.XmlNode as a child under the referenced parent node in the DOM. The last boolean parameter determines whether this operation will be reflected in the Flare Track Change system.
- *void* LookupVariableValue(string, string) Returns the value of the given variable set and name.
- *void* RemoveDocumentNode(XmlNode, bool) Removes an existing System.Xml.XmlNode from the DOM. The last boolean parameter determines whether this operation will be reflected in the Flare Track Change system.
- *bool* RemoveStylesheet(string) Removes the referenced CSS stylesheet to the working list of styles. Paths may be absolute or relative.

- *void* ReplaceDocumentNode(XmlNode, XmlNode, bool) Replaces an existing System.Xml.XmlNode with a new node in the DOM. The last boolean parameter determines whether this operation will be reflected in the Flare Track Change system.
- *bool* Save() Saves the current document. Returns bool indicating success of the operation.
- *void* Select(string) Selects the first instance of text matching the search string after the current position of the insertion point.
- *void* Select(XmlNode) Selects the matching System.Xml.XmlNode.
- *void* SetCursorPosition(int) Sets the position of the insertion point to the specified value.
- *void* StartOperation(string) Starts a new operation. All changes within a StartOperation and EndOperation call will be bundled into one undoable operation in Flare's undo/redo stack.
- *void* UpdateView() Refreshes the document.

ISelection Interface

The ISelection interface provides functionality to edit the document selection.

Properties

- *bool* IsBold Gets the value indicating whether the selection is bold.
- *bool* IsItalic Gets the value indicating whether the selection is italic.
- *bool* IsUnderlined Gets the value indicating whether the selection is underlined.
- *bool* TrackChange Gets and sets the value indicating if ISelection operations are tracked.

Methods

- *bool* Bold() Bolds the selection.
- *bool* ChangeBackColor(Color) Changes the background of the selection to the specified System.Drawing.Color.
- *bool* ChangeFontFamily(string) Changes the font family of the selection to the specified value.
- *bool* ChangeFontSize(float) Changes the font size of the selection to the specified value.
- *bool* ChangeForeColor(Color) Changes the foreground color of the selection to the specified System.Drawing.Color.
- *bool* FormatStyle(string) Changes the span style of the selection to the specified CSS class and/or ID.
- *Color* GetBackColor() Returns the System.Drawing.Color of the background.
- *string* GetFontFamily() Returns the font family of the selection.
- floatGetFontSize() Returns the font size of the selection.
- *Color* GetForeColor() Returns the System.Drawing.Color of the foreground.
- *string* GetStyle() Returns the CSS class and/or ID of the selection.
- *string* GetText() Returns the selected text.
- *IList<XmlNode>* GetXmlNodeList() Returns a list of System.Xml.XmlNode(s) that the selection is a part of.
- *bool* Italicize() Italicizes the selection.

- *bool* RemoveFormatStyle(string) Removes the specified span style/class of the selection. If the specified argument is null, all inline formatting is removed from the selection.
- *bool* ReplaceText(string) Replaces the selected text with the specified text.
- *void* SetSelectionLength(int) Sets the length of the selection to the specified value.
- *bool* Underline() Underlines the selection.

EditorView Enumeration

The EditorView enum defines constants, which indicate the type of editor views.

Members

- Text Text Editor view.
- Xml Xml Editor view.

INavContext Interface

An instance of INavContext is returned using the GetNavContext() method in the IHost interface (see "IHost Interface" on page 16). INavContext gives access to the navigational user interface components of Flare.

Methods

- *Form* **GetParentForm()** Returns the parent System.Windows.Form. To be used as a reference control for child windows.
- *ICustomToolBar* **CreateCustomToolBar(string)** Creates an instance of ICustomToolBar. See "ICustomToolBar Interface" on the next page.
- *IToolStripMenuItem* **CreateToolStripMenu(string, string)** Creates an instance of IToolStripMenuItem to be added to the Tool Strip menu. See "IToolStripMenuItem Interface" on page 25.
- *IRibbon* **GetRibbon()** Returns the current IRibbon object. See "IRibbon Interface" on page 25.

ICustomToolBar Interface

The ICustomToolBar interface lets you build a custom toolbar to be added to the Tool Strip user interface in Flare.

Methods

- *Button* AddButton(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a button to the toolbar. Returns an instance of System.Windows.Controls.Button.
- *Button* AddButton(IRibbonControlData) Adds a button to the toolbar. Returns an instance of System.Windows.Controls.Button.
- *IRibbonComboBox* AddCombobox(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a combo box to the toolbar. Returns an instance of IRibbonComboBox. See "IRibbonComboBox Interface" on page 29.
- *IRibbonComboBox* AddCombobox(IRibbonMenuData) Adds a combo box to the toolbar. Returns an instance of IRibbonComboBox. See "IRibbonComboBox Interface" on page 29.
- *IRibbonMenu* AddMenuButton(string, string, RibbonIconSize, string, string, string) Adds a menu button to the toolbar. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on page 29.
- *IRibbonMenu* AddMenuButton(IRibbonMenuData) Adds a menu button to the toolbar. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on page 29.
- *void*AddSeparator() Adds a separator to the toolbar.
- *IRibbonMenu* AddSplitMenuButton(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a split menu button to the toolbar. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on page 29.
- *IRibbonMenu* AddSplitMenuButton(IRibbonMenuData) Adds a split menu button to the toolbar. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on page 29.
- *ToggleButton* AddToggleButton(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a split menu button to the toolbar. Returns an instance of System.Windows.Controls.Primitives.ToggleButton.
- *ToggleButton* AddToggleButton(IRibbonControlData) Adds a split menu button to the toolbar. Returns an instance of System.Windows.Controls.Primitives.ToggleButton.

NOTE The custom toolbar is only visible when the Flare interface is in “Tool Strip” mode.

IToolStripMenuItem Interface

The IToolStripMenuItem interface lets you build a custom Tool Strip menu to be added to Flare’s Tool Strip user interface.

Methods

- *IToolStripMenuItem* AddMenuItem(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a menu item to the menu. Returns an instance of IToolStripMenuItem.
- *IToolStripMenuItem* AddMenuItem(IRibbonMenuData) Adds a menu item to the menu. Returns an instance of IToolStripMenuItem.
- *void* AddSeparator() Adds a separator to the menu.

IRibbon Interface

The IRibbon interface provides the starting point to add custom elements to the application ribbon user interface.

Properties

- *bool* IsCollapsed Gets the value indicating whether the ribbon is collapsed.

Methods

- *IRibbonTab* AddNewRibbonTab(string, string) Adds a new tab to the ribbon. Returns an instance of IRibbonTab. See "IRibbonTab Interface" on page 27.

IRibbonControlData Interface

The IRibbonControlData interface gives users an object template for data binding to ribbon controls.

Properties

- *bool* CanAddToQuickAccessToolBarDirectly Gets and sets a value that indicates whether this control can be added directly to the Quick Access toolbar.
- *ICommand* Command Gets and sets the command to execute.
- *string* Font Gets and sets the font.
- *bool* IsChecked Gets and sets a value that indicates whether this control is checked.
- *string* KeyTip Gets and sets the key tip value.
- *string* Label Gets and sets the label.
- *string* MenuLabel Gets and sets the menu label.
- *ImageSource* LargeImage Gets and sets the large image (32x32).
- *bool* ShouldExecuteCommand Gets and sets the value indicating whether the command should be executed.
- *ImageSource* SmallImage Gets and sets the small image (16x16).
- *string* ToolTipDescription Gets and sets the tool tip description.
- *string* ToolTipFooterDescription Gets and sets the tool tip footer description.
- *Uri* ToolTripFooterImage Gets and sets the tool tip footer image.
- *string* ToolTipFooterTitle Gets and sets the tool tip footer title.
- *Uri* ToolTipImage Gets and sets the tool tip image.
- *string* ToolTipTitle Gets and sets the tool tip title.
- *object* Value Gets and sets the value.

IRibbonMenuData Interface

The IRibbonMenuData interface gives you an object template for data binding to menu-type Ribbon controls. It inherits from IRibbonControlData. See "IRibbonControlData Interface" on the previous page.

Properties

- *ObservableCollection<IRibbonControlData>* ControlDataCollection Gets the collection of IRibbonControlData used to generate the content of the control. See "IRibbonControlData Interface" on the previous page.

IRibbonTab Interface

The IRibbonTab interface represents a tab in IRibbon. See "IRibbon Interface" on page 25.

Methods

- *IRibbonGroup* AddNewRibbonGroup(string) Adds a new ribbon group to the ribbon tab. Returns an instance of IRibbonGroup. See "IRibbonGroup Interface" on the next page.

IRibbonGroup Interface

The IRibbonGroup interface represents a group of controls that appear in IRibbonTab. See "IRibbonTab Interface" on the previous page.

Methods

- *Button* AddRibbonButton(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a ribbon button to the ribbon group. Returns an instance of System.Windows.Controls.Button.
- *Button* AddRibbonButton(IRibbonControlData) Adds a ribbon button to the ribbon group. Returns an instance of System.Windows.Controls.Button.
- *CheckBox* AddRibbonCheckBox(string, ICommand, object, string, string, string) Adds a ribbon check box to the ribbon group. Returns an instance of System.Windows.Controls.CheckBox.
- *CheckBox* AddRibbonCheckBox(IRibbonControlData) Adds a ribbon check box to the ribbon group. Returns an instance of System.Windows.Controls.CheckBox.
- *IRibbonComboBox* AddRibbonCombobox(string, ICommand, object, string, RibbonIconSize, string, string, string) Adds a ribbon combo box to the ribbon group. Returns an instance of IRibbonComboBox. See "IRibbonComboBox Interface" on the next page.
- *IRibbonComboBox* AddRibbonCombobox(IRibbonMenuData) Adds a ribbon combo box to the ribbon group. Returns an instance of IRibbonComboBox. See "IRibbonComboBox Interface" on the next page.
- *IRibbonMenu* AddRibbonMenuButton(string, string, RibbonIconSize, string, string, string) Adds a ribbon menu button to the ribbon group. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on the next page.
- *IRibbonMenu* AddRibbonMenuButton(IRibbonMenuData) Adds a ribbon menu button to the ribbon group. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on the next page.
- *IRibbonMenu* AddRibbonSplitMenuButton(string, ICommand, object,string, RibbonIconSize, string, string, string) Adds a ribbon split menu button to the ribbon group. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on the next page.
- *IRibbonMenu* AddRibbonSplitMenuButton(IRibbonMenuData) Adds a ribbon split menu button to the ribbon group. Returns an instance of IRibbonMenu. See "IRibbonMenu Interface" on the next page.

- *ToggleButton* AddRibbonToggleButton(string, ICommand, object,string, RibbonIconSize, string, string, string) Adds a ribbon split menu button to the ribbon group. Returns an instance of System.Windows.Controls.Primitives.ToggleButton.
- *ToggleButton* AddRibbonToggleButton(IRibbonControlData) Adds a ribbon split menu button to the ribbon group. Returns an instance of System.Windows.Controls.Primitives.ToggleButton.

IRibbonComboBox Interface

The IRibbonComboBox represents a combo box on IRibbonTab. See "IRibbonTab Interface" on page 27.

Methods

- *void* AddComboboxItem(string) Adds a combo box item with the specified value to the combo box control.
- *string* GetComboboxValue() Gets the current value of the combo box control.
- *void* SetComboboxValue(string) Sets the current value of the combo box control to the specified string.

IRibbonMenu Interface

The IRibbonMenu interface represents a menu on IRibbonTab. See "IRibbonTab Interface" on page 27.

Methods

- *void* AddMenuItem(string, ICommand, object, string) Adds a menu item for ribbon menu-type controls.
- *void* AddSeparator() Adds a separator to the ribbon menu-type control.

RibbonIconSize Enumeration

The RibbonIconSize enum defines constants, which indicate the size of a ribbon icon image.

Members

- **Collapsed** The image is not visible.
- **Small** The image size is 16x16 pixels at 96 DPI.
- **Large** The image size is 32x32 pixels at 96 DPI.

CHAPTER 3

Examples

When creating and working with the plug-in API, reviewing examples might be of use to you.

This chapter discusses the following:

Context Menu Example 32
Controlled Language Example 33
Ribbon Example 37
Search and Change Text Style Example 40
Toolbar Example 41

Context Menu Example

Following is an example of how to add and properly remove a context menu item for a given instance of IDocument. See "IDocument Interface" on page 18.

```
public void AddContextMenuItem(IDocument doc)
{
    if (doc != null)
    {
        List<System.Windows.Forms.ToolStripItem> cms =
doc.GetUserPendingContextMenuItems();
        cms.Add(new System.Windows.Forms.ToolStripMenuItem("Editor Plugin Item"));
    }
}
public void RemoveContextMenuItem(IDocument doc)
{
    if (doc != null)
    {
        List<System.Windows.Forms.ToolStripItem> cms =
mCurrentDocument.GetUserPendingContextMenuItems();
        cms.Clear();
    }
}
```

NOTE If you have multiple documents with custom context menu items, you need to manually clear their context “toolstripitem” lists.

Controlled Language Example

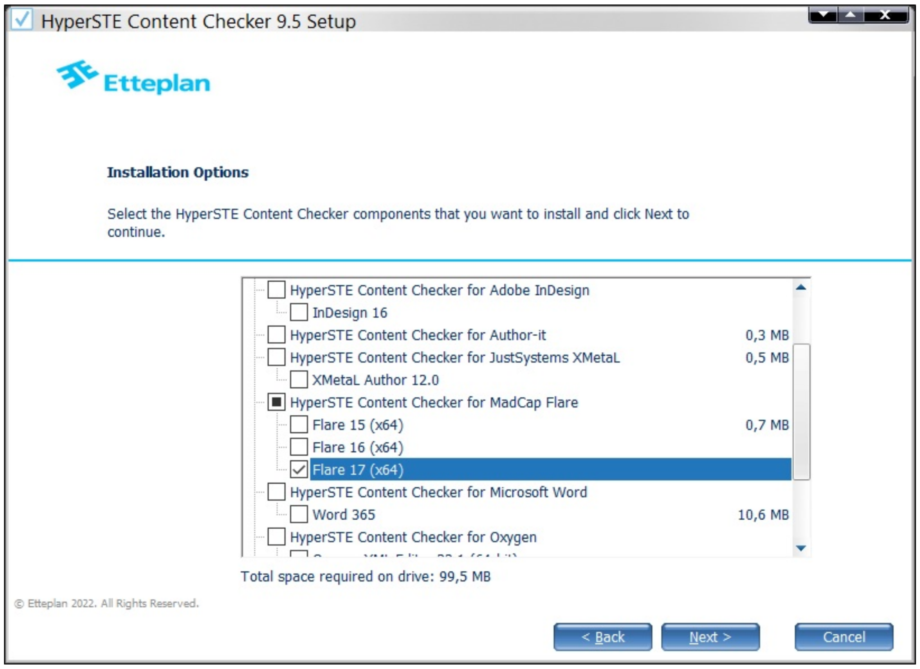
One of the most common uses for the Flare plug-in API is to implement controlled language with Simplified Technical English (STE). This lets you validate the language and terminology used in your Flare content, ensuring the use of standardized vocabulary and style, while improving consistency, eliminating ambiguity, and reducing complexity. This also means you can ensure compliance with corporate terminology and style guide rules.

Two STE solutions that can be integrated with Flare come from Acrolinx (acrolinx.com) and Etteplan (etteplan.com).

EXAMPLE You use Etteplan HyperSTE, which is one of the leading software solutions for STE. You want to integrate HyperSTE with your Flare project so that you can take advantage of controlled language when you write topics and snippets. You might do the following to set it all up and use it.

REGISTER HYPERSTE AND ENABLE THE PLUG-IN

1. Install HyperSTE and launch it.
   a. Select **HyperSTE for MadCap Flare**.



HyperSTE Content Checker 9.5 Setup

Etteplan

Installation Options

Select the HyperSTE Content Checker components that you want to install and click Next to continue.

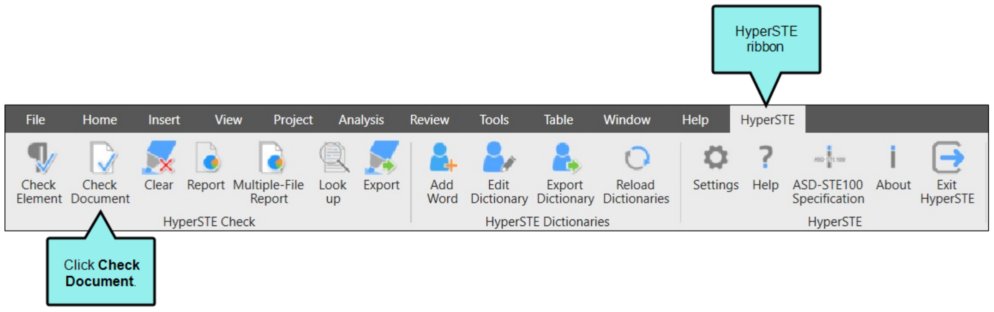
- [ ] HyperSTE Content Checker for Adobe InDesign
  - [ ] InDesign 16
- [ ] HyperSTE Content Checker for Author-it 0,3 MB
- [ ] HyperSTE Content Checker for JustSystems XMetaL 0,5 MB
  - [ ] XMetaL Author 12.0
- ■ HyperSTE Content Checker for MadCap Flare
  - [ ] Flare 15 (x64) 0,7 MB
  - [ ] Flare 16 (x64)
  - [✓] Flare 17 (x64)
- [ ] HyperSTE Content Checker for Microsoft Word
  - [ ] Word 365 10,6 MB
- [ ] HyperSTE Content Checker for Oxygen

Total space required on drive: 99,5 MB

© Etteplan 2022. All Rights Reserved.

   b. Don't launch the license server.
   c. Click **Finish**.
2. Open a Flare project and select **File > Options**. The Options dialog opens.
3. Select the **Plugins** tab.
4. Click **Enable**.

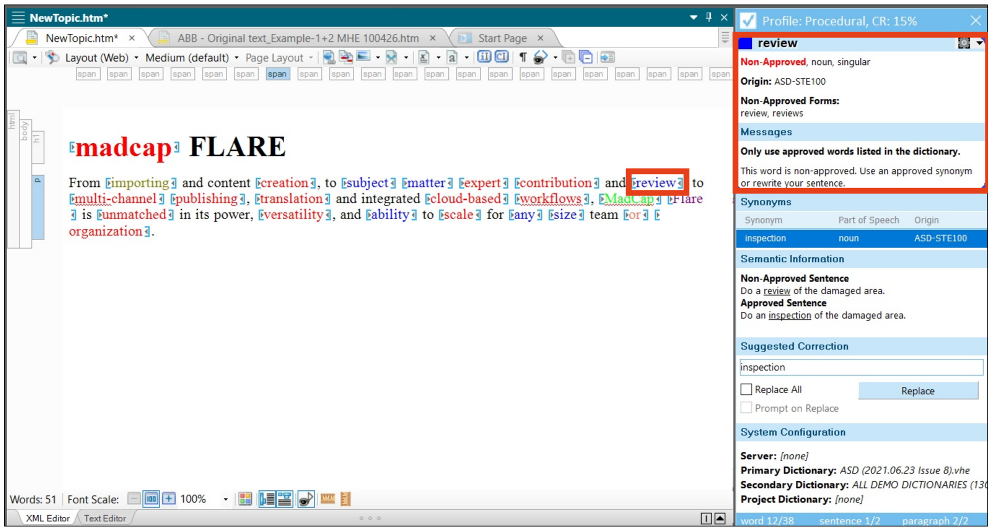
5. Select the **HyperSTE** ribbon and click **Check Document**.



6. When you check a document for the first time, the HyperSTE plugin looks for your associated license. If you don't have a license already, you'll need to request one from Etteplan. To request a trial license of HyperSTE, contact ste-support@etteplan.com.
7. When a license file is returned, save it on your machine.
8. From the **HyperSTE** ribbon, click **Check Document**.
9. When the licensing UI launches, select **I have a license**. This launches the HyperSTE configuration wizard.
10. Click **Next** and set up HyperSTE, filling out the information as needed.
11. Browse for the **V2C** file and click **Next**.
12. Select a dictionary file (acquired from Etteplan).

USE HYPERSTE TO CHECK A CONTENT FILE IN FLARE

1. Open a project in Flare.
2. Open the topic you want to validate.
3. From the **HyperSTE** ribbon, click **Check Document**.
4. Click on highlighted words throughout the topic. Notice the HyperSTE window explaining the terms and usage.



madcap FLARE

From importing and content creation, to subject matter expert contribution and review to multi-channel publishing, translation and integrated cloud-based workflows, MadCap Flare is unmatched in its power, versatility, and ability to scale for any size team or organization.

Profile: Procedural, CR: 15%

review

Non-Approved, noun, singular

Origin: ASD-STE100

Non-Approved Forms:
review, reviews

Messages

Only use approved words listed in the dictionary.

This word is non-approved. Use an approved synonym or rewrite your sentence.

Synonyms

Synonym	Part of Speech	Origin
inspection	noun	ASD-STE100

Semantic Information

Non-Approved Sentence
Do a review of the damaged area.
Approved Sentence
Do an inspection of the damaged area.

Suggested Correction

inspection

Replace All
Replace
Prompt on Replace

System Configuration

Server: *[none]*
Primary Dictionary: *ASD (2021.06.23 Issue 8).vhe*
Secondary Dictionary: *ALL DEMO DICTIONARIES (13*
Project Dictionary: *[none]*

word 12/38 sentence 1/2 paragraph 2/2

You can also use the other features shown in the HyperSTE ribbon in Flare.

Ribbon Example

To add a ribbon tab for your plug-in, you need to use the INavContext interface. See "INavContext Interface" on page 23.

From it, you may call GetRibbon() to retrieve an IRibbon instance. With the IRibbon instance, you may add a new ribbon tab.

Here is an example using data binding to the IRibbonControlData/IRibbonMenuData interfaces. See "IRibbonControlData Interface" on page 26 and "IRibbonMenuData Interface" on page 27.

```
private void CreatePluginRibbon()
{
    IRibbon ribbon = mNavigationContext.GetRibbon();
    IRibbonTab tab = ribbon.AddNewRibbonTab("PluginTab", "P");
    IRibbonGroup group = tab.AddNewRibbonGroup("PluginGroup");
    Button searchAndHighlightButton = group.AddRibbonButton
(PluginViewModel.SearchHighlight);
    IRibbonComboBox comboMenu = group.AddRibbonCombobox(PluginViewModel.Fonts);
}
```

Here is sample code for the PluginViewModel.

```
public static classPluginViewModel
{
    private staticRibbonControlData _searchHighlight;
    private staticRibbonMenuData _fonts;
    public staticRibbonControlData SearchHighlight
    {
        get
        {
            if (_searchHighlight == null)
            {
                _searchHighlight = new RibbonControlData()
                {
                    Label = "Search and Highlight",
                    Command = new ButtonCommand(),
                    KeyTip = "S"
                };
            }
            return _searchHighlight;
        }
    }
    public staticRibbonMenuData Fonts
    {
        get
        {
            if (_fonts == null)
            {
                BitmapImage image = new BitmapImage(new Uri(
"C:/TFS/Trunk/ObjectApplications/Debug/Flare.app/Plugins/NavigationPlugin/Icons/EditDoc
ument.png"));
                _fonts = new RibbonMenuData()
                {
                    Label = "My Fonts",
                    LargeImage = image
                };
                _fonts.ControlDataCollection.Add(new RibbonControlData()
                {
                    Label = "Arial"
                });
                _fonts.ControlDataCollection.Add(new RibbonControlData()
                {
                    Label = "Helvetica"
                });
                _fonts.ControlDataCollection.Add(new RibbonControlData()
                {
                    Label = "Times New Roman"
                });
            }
            return _fonts;
        }
        }
}
```

NOTE The ribbon tab is only visible when the Flare interface is in “Ribbon” mode.

Search and Change Text Style Example

Following is a basic example where IEditorContext is used to search for a input string and change the style of it throughout the document. See "IEditorContext Interface" on page 17.

In this example, assume your class instance of IEditorContext is named “mEditorContext.”

```
private void SearchAndChangeStyle(string searchString)
{
    IDocument currentDocument = mEditorContext.GetActiveDocument();
    if (currentDocument != null)
    {
        string text = currentDocument.GetDocumentText();
        int occurences = Regex.Matches(text, searchString).Count;
        for (int i = 0; i < occurences; i++)
        {
            currentDocument.Select(searchString);
            currentDocument.Selection.ChangeBackColor(Color.Yellow);
            currentDocument.Selection.ChangeForeColor(Color.Red);
            currentDocument.Selection.ChangeFontFamily("Comic Sans MS");
        }
    }
}
```

Toolbar Example

To add a custom toolbar, you need to use the INavContext interface. See "INavContext Interface" on page 23.

Here is an example assuming your class instance of INavContext is named “mNav.”

```
private void CreateCustombar()
{
    ICustomToolBar toolBar = mNav.CreateCustomToolBar("My ToolBar");
    toolBar.AddButton("My Button",newButtonCommand());
    toolBar.AddSeparator();
    toolBar.AddMenuButton(MyViewModel.MenuData);
}
```

ButtonCommand and MyViewModel are defined as follows.

```
public classButtonCommand :ICommand
{
    public bool CanExecute(object parameter)
    {
        return true;
    }
    public voidExecute(object parameter)
    {
        MessageBox.Show("I got pressed!");
    }
}
public static classRibbonViewModel
{
    private static RibbonMenuData _menuData;
    public staticRibbonMenuData MenuData
    {
        get
        {
            if (_menuData ==null)
            {
                BitmapImage image =new BitmapImage(new Uri("<ICON PATH>"));
                _menuData =new RibbonMenuData()
                {
                    Label ="My Menu",
                    SmallImage = image,
                    KeyTip ="D"
                };
                _menuData.ControlDataCollection.Add(new RibbonControlData()
                {
                    MenuLabel = "item1"
                });
                _menuData.ControlDataCollection.Add(new RibbonControlData()
                {
                    MenuLabel = "item2"
                });
            }
            return _menuData;
        }
    }
}
```

NOTE The custom toolbar is only visible when the Flare interface is in “Tool Strip” mode.

PDFs

The following PDFs are available for download from the online Help.

Tutorials

Autonumbers Tutorial

Back-to-Top Button Tutorial

Context-Sensitive Help Tutorial

Custom Toolbar Tutorial

eLearning Tutorial—Basic

eLearning Tutorial—Advanced

Getting Started Tutorial

Image Tooltips Tutorial

Lists Tutorial

Meta Tags Tutorial

Micro Content Tutorial—Basic

Micro Content Tutorial—Advanced

Responsive Output Tutorial

Single-Sourcing Tutorial

Snippet Conditions Tutorial

Styles Tutorials

Tables Tutorial

Word Import Tutorial

Cheat Sheets

Context-Sensitive Help Cheat Sheet

Folders and Files Cheat Sheet

Learning & Development Cheat Sheet

Lists Cheat Sheet

Micro Content Cheat Sheet

Print-Based Output Cheat Sheet

Search Cheat Sheet

Shortcuts Cheat Sheet

Structure Bars Cheat Sheet

Styles Cheat Sheet

User Guides

Accessibility Guide

Analysis and Reports Guide

Architecture Guide

Autonumbers Guide

Branding Guide

Condition Tags Guide

Context-Sensitive Help Guide

Eclipse Help Guide

eLearning Guide

Getting Started Guide

Global Project Linking Guide

HTML5 Guide

Images Guide

Import Guide

Indexing Guide

Key Features Guide

Lists Guide

MadCap Central Integration Guide

Meta Tags Guide

Micro Content Guide

Navigation Links Guide

Plug-In API Guide

Print-Based Output Guide

Project Creation Guide

QR Codes Guide

Reviews & Contributions With Contributor Guide

Scripting Guide

Search Guide

SharePoint Guide

Skins Guide

Snippets Guide

Source Control Guide: Git

Source Control Guide: Perforce Helix Core

Source Control Guide: Subversion

Source Control Guide: Team Foundation Server

Styles Guide

Tables Guide

Tables of Contents Guide

Targets Guide

Template Pages Guide

Templates Guide

Topics Guide

Touring the Workspace Guide

Transition From FrameMaker Guide

Translation and Localization Guide

Variables Guide

Videos Guide

What's New Guide