Alma Style Fix

# Premessa:

Questa estensione di Visual Studio si basa un programma Open Source pubblicata su Codeplex. <http://stylefix.codeplex.com/>

Il codice originale è stato corretto e trasformato in una estensione di Visual Studio. Durante il processo di trasformazione, è stata ridotta la funzionalità di base, in modo tale da consentire di poter modificare solo il file corrente. L’unico linguaggio supportato è il C#.

Affinché il programma funzioni correttamente, il documento DEVE far parte di un progetto.

# Panoramica sulle librerie di base:

Le librerie a cui il programma si appoggia sono 2:

1. StileFix: Questa libreria si basa su StyleCop, che viene usato per individuare gli errori nei file da correggere. Una volta individuati la libreria interviene sulle singole linee di codice da modificare.

Poiché questa libreria è stata oggetto di pesanti modifiche, è inclusa all’interno della soluzione sotto forma di progetto.

1. Narrange : è una libreria open soure in grado di supportare sia C# che VB. Il codice è strutturato in modo da forzare il programmatore ad agire sui file a partire dal path. Coregge alcuni errori di stile non supportati da StyleFix, in particolare cambia l’orine delle funzioni membro delle classi in modo conforme alle regole di stile. Anche in questo caso è stato necessario modificare i sorgenti, per adattarli al funzionamento come estensione di Visual Studio.

# Struttura di AlmaStyleFix:

La soluzione è composta da due progetti.

* Un progetto di tipo Package che si occupa soprattutto dell’aspetto dell’estensione e dell’interfacciamento all’ambiente di VS. In questa parte della soluzione è presente anche del codce che si occupa del fix di quegli errori che conviene correggere utilizzando i comandi messi a disposizione da VS.
* Un progetto di tipo libreria che contiene StyleFix. Questa parte della soluzione è il “motore” dell’estensione, e si occupa di esplorare il testo del documento alla ricerca delle violazioni e della loro correzione.

# Funzionamento di AlmaStyleFix:

L’estensione pubblica due comandi:

1. Correggi il documento corrente ( Menù di Contesto)
2. Visualizza le opzioni disponibili. (Menù Tools)

Come già accennato, la presenza di un file di progetto è fondamentale per il buon funzionamento del programma, in quanto richiesta dalla libreria StyleCop sottostante. Per questo motivo, prima di poter lanciare l’algoritmo di correzione dello stile occorre salvare il documento (operazione eseguibile automaticamente da parte del comando stesso). Una volta che l’algoritmo ha terminato, il testo corretto diventa il contenuto dell’ACTIVE DOCUMENT. L’operazione è reversibile tramite la combinazione di tasti “CTRL+Z”.

Durante l’esecuzione dell’algoritmo di correzione dello stile, vengono utilizzati alcuni comandi di NATIVI di Visual Studio, che arricchiscono le funzionalità offerte dall’algoritmo stesso.

L’altro comando visualizza semplicemente le opzioni di Visual Studio, tra le quali compare AlmaStyleFix -> General. In questa schermata l’utente può decidere se:

1. Abilitare l’autosalvataggio del documento prima di avviare l’algoritmo.
2. Abilitare il controllo della versione dell’assembly generato.
3. Inserire il nome dell’Azienda ed il proprio nome, che verranno cpoiati nel commento HEADER in testa ad ogni file di codice.
4. Abilitare l’evidenziatore degli errori di stile, per una più rapida individuazione e correzione degli stessi.

## Il controllo delle Versioni:

Questa funzionalità verifica che la versione dell’assembly generato corrisponda alla versione del branch. In caso di discrepanze l’utente può scegliere se aggiornare la versione dell’assembly. Mentre la versione dell’assembly è ottenibile con precisione dalle API di Visual Studio, non è possibile ottenere in modo analogo la versione del Branch. L’algoritmo esamina quindi il PATH del progetto, a partire dalla foglia verso la radice, alla ricerca di una stringa che contenga quattro cifre numeriche separate da tre punti (Standard per il naming dei branch). Nel caso in cui tale combinazione di caratteri sia presente nel path, questa viene interpretata come versione del branch. Ovviamente il controllo delle versioni non può essere applicato quando si lancia AlmaStyleFix su un progetto che appartiene alla MainLine, che non è versionata.

## L’evidenziatore:

Le estensioni di vVisual Studio consentono non solo la modifica dei sorgenti aperti, ma anche di interventire sull’aspetto grafico dell’editor. AlmaStyleFix sfrutta questa funzionalità evidenziando eventuali errori di stile. Le righe che violano almeno una regola vengono visualizzate con lo sfondo colorato. Nella schermata delle opzioni è possibile cambiare il colore a seconda delle preferenze dello sviluppatore. Le righe presentano inoltre un ToolTip che specifica il testo della prima delle regole violate, in modo da facilitarne la correzione. Questo accorgimento semplifica notevolmente il lavoro di correzione del codice, ma richiede un maggiori risorse di processore, per cui se ne sconsiglia l’uso su macchine poco potenti.

# Le regole di FIX:

Il progetto iniziale è in grado di risolvere una dozzina circa di violazioni a StyleCop. L’algoritmo base segue i seguenti passi:

1. Per ogni rigo del documento crea un elemento di una lista.
2. Esegue StyleCop e segna le violazioni riscontrate.
3. Ad ogni riga della lista viene aggiunta a sua volta una lista delle violazioni riscontrate per quella riga.
4. Per ogni regola implementata:
   1. Si scorre la lista delle righe alla ricerca di quelle che violano la regola (foreach).
   2. In caso di match si prendono le opportune contromisure

Il codice che implementa le varie regole è suddiviso per categorie.

## Categorie di regole:

Le regole di StyleCop sono suddivise in categorie. Ogni regola è identificata dalla sigla SA seguita da un numero a 4 cifre. Solo alcune regole possono essere risolte automaticamente, mentre le altre richiedono comunque l’intervento dell’utente.

Le prime due cifre identificano la categoria, le altre la regola particolare.

Le categorie più importanti sono:

* SA10\*\* : regole di spaziatura! Sono tutte risolte dal comando “Edit.FormatDocument” di VS.
* SA11\*\* : regole di leggibilità.
* SA12\*\* : regole di ordinamento delle righe. Sono risolte dalla libreria Narrange.
* SA13\*\* : regole di nome. Vengono risolte dal comando VS “Rename”.
* SA15\*\* : regole su linee bianche e graffe.
* SA16\*\* : regole sulla documentazione.

## Il renaming:

Il renaming è una regola molto particolare. Per poterla implementare in maniera efficiente è stato usato il comando Visual Studio Refactor -> Rename. Durante l’esecuzione di StyleFix viene creato un dizionario le cui entrate rappresentano

1. Il vecchio nome
2. Il nuovo nome

Una volta completata l’esecuzione di StyleFix parte un codice che, per ogni entrata del dizionario, ricerca l’elemento nel DOM (Struttura ad albero) e quando lo trova invoca il comando RENAME di VS, che ha effetto anche sugli altri file del progetto.

# Dedicato agli Sviluppatori:

## Creazione di una nuova Estensione di VS:

Le estensioni sono usate, oltre che per inserire nuovi comandi, anche per esportare template, sia di progetto che di oggetti.

In questo caso è stato esportato il solo comando. Per creare una estensione occorre creare un nuovo progetto di Visual Studio di tipo extendibility -> package. Il progetto crea in output file con estensione VSIX, che è a tutti gli effetti un file di installazione per Visual Studio, e che può essere installato semplicemente facendoci sopra doppio click.

Per poter usufruire di questo tipo di progetto, occorre scaricarare il Visual Studio SDK, scaricabile al seguente LINK : http://www.microsoft.com/download/en/details.aspx?id=2680

Il nuovo comando viene inserito di default nel menu “TOOL” di Visual Studio, ma è possibile aggiungere un nuovo menù oppure aggiungere il programma al menù di CONTESTO, che viene visualizzato quando si usa il tasto destro sul documento attivo.

## File di un progetto di tipo Extension:

In genere si lavora su 3 dei numerosi file che vengono creati nel progetto:

1. <Nome>Package.cs : Contiene il codice del pacchetto, incluso il corpo dei comandi che l’estensione mette a disposizione.
2. <Nome>.vstc : è un file in formato XML il cui contenuto rappresenta l’aspetto grafico dell’Estensione. Tra le altre informazioni, il contenuto di questo file specifica:
   1. Il testo e le icone dei comandi
   2. Il menù di appartenenza dei comandi
3. source.extension.vsixmanifest : Anche se il suo contenuto è un XML, conviene visualizzare questo file aprendolo con un doppio click. Al suo interno sono specificate le informazioni dell’estensione, quali:
   1. Versione
   2. Icona
   3. Link alla documentazione
   4. Descrizione
   5. Versioni di Visual Studio supportate

## Dentro Visual Studio: Comandi e struttura dati

La documentazione Microsoft è lacunosa, e le librerie degli add-in non dispongono di summary, per cui può essere complicato sviluppare un progetto di add-in. Qui di seguito sono riportati alcuni suggerimenti che possono rivelarsi molto utili.

La variabile di ambiente che contiene tutte le informazioni sullo stato di Visual Studio è DTE.

var dte = Package.GetGlobalService(typeof(SDTE)) as DTE;

### Ottenere la lista dei documenti aperti:

var docs = dte.Documents;

### Ottenere il documento corrente:

var theDoc = dte.ActiveDocument;

### Ottenere il progetto corrente corrente:

Project activeProject = null;

Array activeSolutionProjects = dte.ActiveSolutionProjects as Array;

if (activeSolutionProjects != null && activeSolutionProjects.Length > 0)

{

activeProject = activeSolutionProjects.GetValue(0) as Project;

}

### MessageBox:

Le MessageBox si possono utilizzare, ma vanno chiamate usando le primitive a basso livello. Per questo motivo conviene racchiudere la chiamata in una funzione:

private int question(string text)

{

IVsUIShell uiShell = (IVsUIShell)GetService(typeof(SVsUIShell));

int result;

Guid clsid = Guid.Empty;

uiShell.ShowMessageBox(

0,

ref clsid,

"AlmaStyleFix",

text,

string.Empty,

0,

OLEMSGBUTTON.OLEMSGBUTTON\_OKCANCEL,

OLEMSGDEFBUTTON.OLEMSGDEFBUTTON\_FIRST,

OLEMSGICON.OLEMSGICON\_INFO,

0, // false

out result);

return result;

}

### Lanciare un comando Visual Studio:

Equivale ad eseguire un comando dal prompt dei comandi di VS. Per aprire il prompt View -> Other Windows -> Command Window. Durante lo sviluppo è sempre preferibile testare il commando in questa finestra prima di lanciarlo da codice, in quanto si ha a disposizione l’intellisense evitando errori di sintassi.

Una volta scelto il comando da usare, è possibile lanciarlo come riportato in seguito:

dte.ExecuteCommand("Edit.FormatDocument", string.Empty);

dove “dte” è la variabile di ambiente che ci permette di accedere a tutte le funzionalità dell’editor. La stringa usata come primo parametro indica il comando da eseguire, mentre quella usata come secondo rappresenta i parametri da applicare al comando lanciato.

### Ottenere la versione del progetto corrente:

prj.properties.Item("AssemblyVersion").Value

Dove “prj” è il riferimento al progetto corrente.

## Aggiungere un nuovo comando ad una estensione Estensione:

La procedura è semplice ma macchinosa. Occorre aprire il file VSCT (contiene un XML) creato dal template, ed aggiungere un nuovo “bottone” nella sezione “buttons”. Il bottone viene tradotto come un nuovo elemento del menù. Nelle proprietà del nuovo “bottone” saranno specificate:

* Il padre (il gruppo di menuItem che a sua volta appartiene al menu “tools”).
* L’identificativo del comando da eseguire.
* Il testo del menù.

Il nuovo “bottone” dovrà essere materalmente collegato al menu “tool”. All’interno del file CS principale, esiste un metodo creato automaticamente chiamato “Inizialize()”. Nel metodo è già presente il codice necessario a creare il primo menù. Per aggiungere altri elementi è sufficiente edeguire un copia-incolla del codice e cambiare l’ID del comando ed il nome dello handler che avrà il compito di eseguire il corpo del comando stesso.

## Il Menù Opzioni di VS:

Il menù è accedibile dall’utente secondo il percorso Tools -> Options. Al suo interno compaiono una serie di opzioni che sono proprie di VS, a cui è possibile aggiungere quelle delle estensioni.

Per aggiungere una nuova voce al menù occorre creare alcune classi:

1. Una classe che contiene la struttura dati da esporre.
2. Una classe di tipo grafico che realizzi l’interfaccia con l’utente.

La prima classe conterrà una proprietà per ogni dato che si vuole esporre. DEVE derivare da un oggetto di tipo DialogPage. I dati inseriti o modificati tramite VS sono memorizzati in modo permanente!!

La seconda classe è un contenitore di controlli. Conterrà il codice dei controlli utente che si interfacciano con i dati sottostanti. La classe DEVE derivare da un oggetto di tipo UserControl.

Per collegare la classe dati ai controlli, occorre ridefinire la proprietà protected override IWin32Window Window appartenente alla classe DialogPage, forzando VS ad utilizzare la nostra interfaccia utente.

protected override IWin32Window Window

{

get

{

ToolOptionsUI page = new ToolOptionsUI();

page.TheOptions = this;

page.Initialize();

return page;

}

}

Dove ToolOptionUI è la classe di interfaccia che abbiamo realizzato.

Una volta costruite le due classi, bisogna aggiungere un nuovo menù alle opzioni di VS. Per realizzarlo si aggiunge un attributo alla classe Package (contenuta nel file <Nome>Package.cs).

[ProvideOptionPage(typeof(ToolsOptions), "AlmaStyleFix", "General", 1001, 1002, true)]

Dove i parametri rappresentano:

1. La classe che contiene I dati.
2. Il nome del menù di primo livello.
3. Il nome del menù di secondo livello.
4. Un identificativo per (2) (va bene mettere un numero a caso)
5. Un identificativo per (3) (va bene mettere un numero a caso)
6. Un booleano che indica se sono supportate le automazioni. Mettere sempre a TRUE.

## Leggere i valori delle opzioni:

Durante l’esecuzione dell’algoritmo si può ottenre una istanza della classe delle opzioni usando:

ToolsOptions myOptions = (ToolsOptions)GetDialogPage(typeof(ToolsOptions));

Dove ToolsOptions è la classe dati. Il metodo “GetDialogPage” crea l’istanza e la inizializza con i dati inseriti dall’utente e memorizzati in modo PERSISTENTE da VS.

# Problemi Noti:

Da prove sperimentali emerge un bug sulla deserializzazione delle classi che si verifica solo in questo tipo di progetti. Visual Studio crea una copia della libreria che chiede la de-serializzazione (NARRANGE in questo caso), generando poi eccezione quando si cerca di eseguire il cast dell’oggetto ottenuto (Per VS si tratta di due librerie differenti). Per ovviare è stato necessario caricare la classe direttamente da codice piuttosto che da file XML. Il bug NON SI VERIFCA AD OGNI ESECUZIONE, il che complica non poco la sua individuazione. Si raccomanda la massima attenzione quando è necessario utilizzare il serialzzatore nei progetti di tipo Extension!!!