ADOBE® ILLUSTRATOR® CS3

USING THE ADOBE TEXT ENGINE

Adobe

© 2007 Adobe Systems Incorporated. All rights reserved.

Using the Adobe Text Engine

Technical Note #10502

Adobe, the Adobe logo, and Illustrator are either registered trademarks or trademarks of Adobe Systems Inc. in the United States and/or other countries. Microsoft is either a registered trademark or trademark of Microsoft Corp. in the United States and/or other countries. All other trademarks are the property of their respective owners.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Inc. Adobe Systems Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Contents

Using the Adobe Text Engine

This document describes how to use the Adobe text engine—the text API provided by the Adobe® Illustrator® CS3 SDK—in your Illustrator plug-ins.

The Adobe text engine (ATE) is the component that supports text in Illustrator CS3. This document contains text-related use cases that solve typical programming problems like inserting, deleting, and styling text.

Terminology and notational convention

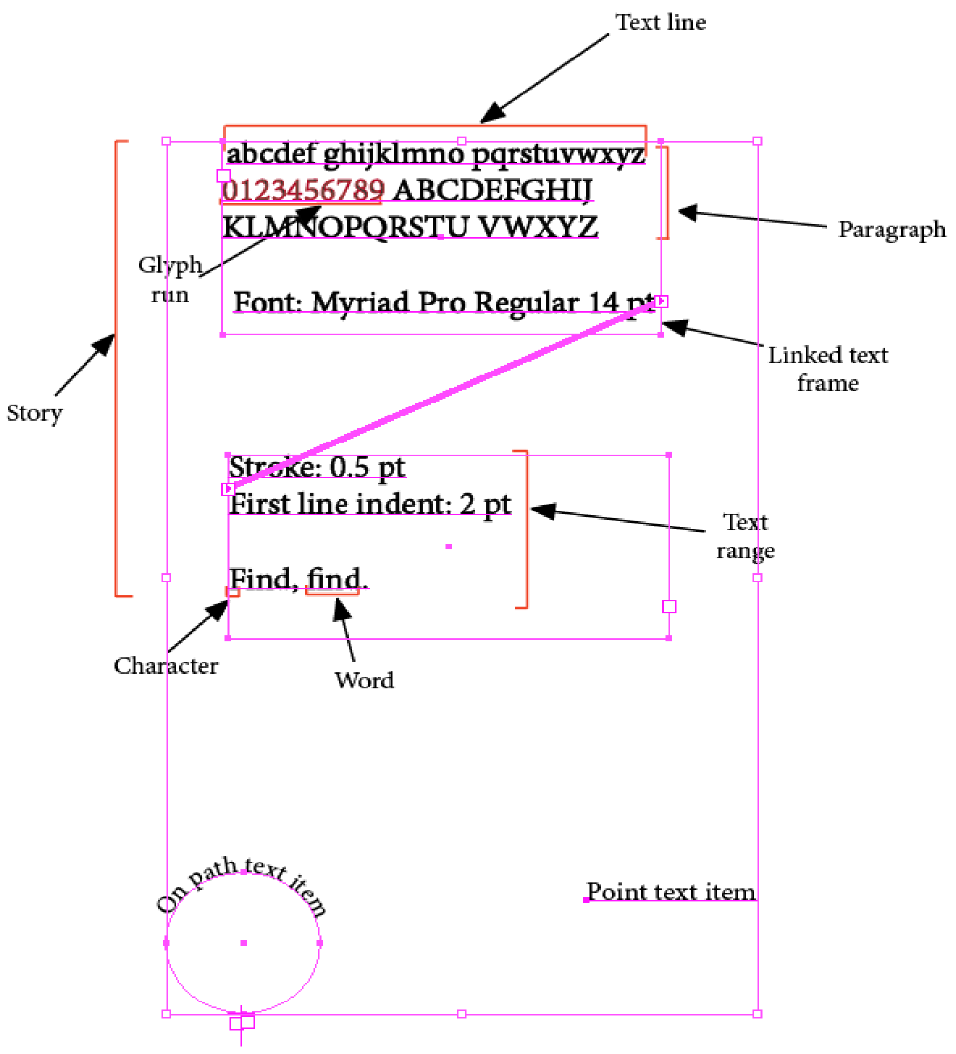
This document uses the following terms:

- *ATE* — Adobe Text Engine.
- *DOM* — Document Object Model.
- *Feature* — A visual attribute applied to a paragraph or character, like justification or font size.
- *Glyph run* — A composed range of text, converted to glyphs and ready to draw.
- *Legacy* — Illustrator 10 or earlier.
- *SDK* — The software development kit for Illustrator CS3.
- *SLO* — The Former name for the Adobe text engine. In the context of Illustrator, this refers to the internal Adobe text engine library.
- *Story* — A container for a range of text flowing over one or more text frames.
- *Style* — A named container for a set of features.
- *Text frame* — An object that displays a range of text.
- *Text line* — A line of text composed to fit the width of a text frame.
- *Text run* — A non-composed range of text with the same features.
- *Visual C++* — Microsoft® Visual Studio 2005, using the C++ environment.

This document uses the following notational convention: <*SDK*> indicates your locally installed SDK root folder. The actual root folder depends on the installation and operating system.

Figure 1 illustrates some terminology used throughout this document.

FIGURE 1 *Terminology*



About the Adobe text engine

The Adobe text engine is the library that provides support for text to Adobe Illustrator. The API described here is specific to Adobe Illustrator.

Some major features the Adobe text engine supports are as follows:

- Unicode.
- OpenType.

- Advanced typography like optical kerning, optical margin alignment, automatic glyph replacement, and glyph scaling.
- Character and paragraph styles.
- Asian text features like MojiKumi, Kinsoku, and composite fonts.

The Adobe text engine was introduced in Illustrator CS (Illustrator 11.0) to replace the suites that provided text support in earlier versions of the product.

Text API components

The text API comprises suites and wrapper classes that together provide an interface to the text in a document. Most of the API features are provided through the wrapper classes, with the suites providing extra support and functionality.

Illustrator text suites

Illustrator provides several text-related suites, notably the following:

- AITextFrameSuite — Provides management functions for kTextFrameArt art objects and allows access to the ITextFrame object associated with a kTextFrameArt art object.
- AITextFrameHitSuite — Provides a reference to which element of a text frame was hit, given an AIHitRef containing positional information.

Adobe text engine wrapper classes

The C++ helper classes declared in the header file <*SDK*>/illustratorapi/ate/ITexh.h should be used in your plug-in code to work with text programatically. These helper classes include ITextFrame, ITextLine, ITextRange, ICharFeatures, ICharInspector, ICharStyle, IParaFeatures, IParaInspector, and IParaStyle.

The following are some key wrapper classes within the Adobe text engine API:

- *ITextFrame* — The main class controlling the layout of text in the document. ITextFrame provides access to the contained text range, lines, and the parent story.
- *IStory* — A flow of text in a document. This flow can be spread across many lines, paragraphs, and text frames. An IStory provides access to the contained text range, paragraphs, words, text runs, and text frames related to the IStory, as well as all the other stories contained in the current document.
- *ITextRange/ITextRanges* — A *text range* is a range of characters from a start offset to an end offset, which can flow over words, paragraphs, text frames, and stories. The ITextRange and ITextRanges classes provide access to iterators that traverse the contained words, text runs, paragraphs, stories, frames, and lines and can access the glyphs, features, and styles that are used.
- *IDocumentTextResources* — Provides access to text resources in a document, such as fonts and styles.

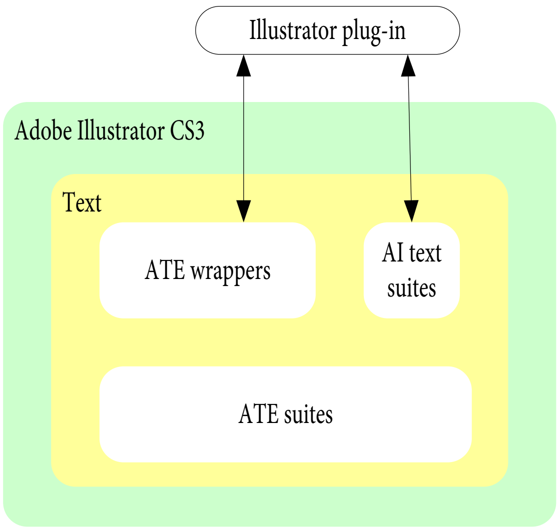
Adobe text engine suites

The Adobe text engine suites in <*SDK*>/illustratorapi/ate/ATESuites.h provide the low-level interface to text. Normally, these suites should not be called directly in your plug-in code; instead, use the Adobe text engine wrapper classes provided by IText.h/IText.cpp. The wrappers call the suites for you and make text programming easier.

NOTE: Adobe text engine wrapper classes are compiled by your plug-in, so you must add IText.cpp to your project to use them. Adobe text engine suites are part of Illustrator, and the suites declared in the header file ATESuites.h allows them to be called.

Figure 2 is an overview of the Adobe text engine API.

FIGURE 2 ***The Illustrator CS3 Adobe text engine API***



Getting started with the text API in your plug-in

Exploring text with SnippetRunner

SnippetRunner is a plug-in that lets you run code snippets provided in the SDK. SnpText is one of several code snippets provided in the SDK that demonstrate the manipulation of text objects in a document.

Procedure

1. Run Illustrator CS3 with the SnippetRunner plug-in loaded.
2. If the SnippetRunner panel is not visible, select Window > SDK > SnippetRunner.

3. In the SnippetRunner panel, expand the hierarchical list of operations under the Text item.

4. Familiarize yourself with the operations.

5. Browse through the sample code of the text snippet in <*SDK*>/samplecode/codesnippets/SnpText.cpp.

For more examples of manipulating text items, see code snippets SnpTextIterator, SnpTextStyler, and SnpTextStyles.

Sample code

See the following code snippets:

- SnpTextIterator — Shows how to find text in Illustrator documents.
- SnpText — Shows how to create, link, and delete text frames, plus how to insert, delete, replace, and move characters.
- SnpTextStyler — Shows how to modify the visual appearance of text, through applying and clearing character and paragraph features.
- SnpTextStyles — Shows how to create, edit, apply, clear, and delete named paragraph and character styles.
- SnpTextException — Shows how to throw and catch an Adobe text engine exception.

References

For instructions on loading an Illustrator plug-in, see *Getting Started with Adobe Illustrator CS3 Development*.

Adding text support to your plug-in

For your plug-in to use the Adobe text engine API, it must compile the Adobe text engine wrapper classes and acquire the necessary suites for these classes during start-up.

In Visual C++, follow these steps:

1. Add <*SDK*>/illustratorapi/ate/IText.cpp to the list of project source files.

2. Right-click on IText.cpp in the Solution Explorer, and select Properties.

3. Under C/C++ > Precompiled Headers, set Create/Use Precompiled Header to Not Using Precompiled Headers.

4. Repeat steps 1–3, this time with <*SDK*>/illustratorapi/ate/IThrowException.cpp.

In Xcode, follow this step:

- Add IText.cpp and IThrowException.cpp to the project source files.

In the source code, follow these steps:

1. Add the following include instruction to the file that contains the definition of the suite pointers. (See <*SDK*>/samplecode/MarkedObjects/Source/MarkedObjectsSuites.h.)

   ```
   #include "ATETextSuitesImportHelper.h"
   ```

2. Add EXTERN_TEXT_SUITES to the list of external suite pointers. (See <*SDK*>/samplecode/MarkedObjects/Source/MarkedObjectsSuites.cpp.)
3. Add IMPORT_TEXT_SUITES to the list of suites and suite versions to be imported. (See <*SDK*>/samplecode/MarkedObjects/Source/MarkedObjectsSuites.cpp.)

NOTE: If you use a structure to pass in all the suite names, versions, and pointers, it must be as follows to match the text-wrapper suite pointers:

```
typedef struct {
char* name;
int version;
void* suite;
} ImportSuite;
```

4. Add any Illustrator text suites you normally require, such as AITextFrameSuite. (See <*SDK*>/samplecode/MarkedObjects/Source/MarkedObjectsSuites.cpp.)

Handling errors from the text API

The Adobe text engine provides an Exception class that reports an error of type ATEErr if an exception is thrown when using Adobe text engine wrapper classes. Using the ATE::Exception class in your code enables your plug-in to catch any unexpected runtime errors from the Adobe text engine.

Procedure

- Wrap all code using the Adobe text engine wrappers in a try block.
- Add a catch block that catches an ATE::Exception and reports its internal error to the plug-in.

API Reference

Exception

Sample code

SnpTextException::ThrowATEException

Accessing text using selection

To access the selected text in the current document, use AIDocumentSuite to get the TextRangesRef, then create a new ITextRanges object using the TextRangesRef.

The new ITextRanges object provides access to the selected text and the containing text frames. By traversing the other containers, like IStory and IStories, you also can access all the unselected text in the document.

The following code samples are taken from SnpTextIterator::IterateSelectedTextFrames and shows how to create an ITextRanges object containing the selected text:

```
TextRangesRef rangesRef = NULL;
ASErr result = sAIDocument->GetTextSelection(&rangesRef);
aisdk::check_ai_error(result);
ITextRanges ranges(rangesRef);
```

To get the text frames containing the selected text, iterate through each ITextRange in the ITextRanges set, and get the text frames associated with each text range:

```
ITextRange range = ranges.Item(rangeIndex);
ITextFramesIterator framesIter = range.GetTextFramesIterator();
```

Alternately, to access the collection of stories in the current document, using code similar to SnpTextIterator::IterateSelectedStories, get the first ITextRange from the ITextRanges object, get the associated IStory for the ITextRange, and then get the IStories set from the IStory object:

```
ITextRange range = ranges.Item(0);
IStory story = range.GetStory();
IStories stories = story.GetStories();
```

Accessing text using the artwork tree

To access the text in a document through the artwork tree, you must first find the text frame art in the document, then convert the AIArtHandle for each text frame to an ITextFrame object. The ITextFrame object provides access to its associated ITextRange. The following code samples are taken from SnpTextIterator::IterateTextFrames.

Create an art set containing all the text art in the current document:

```
AIArtSpec specs[1] = {{kTextFrameArt, 0, 0}};
SnpArtSetHelper textFrameArtSet(specs, 1);
```

For each art item found, convert to an ITextFrame then get the ITextRange:

```
AIArtHandle textFrameArt = textFrameArtSet[artIndex];
TextFrameRef textFrameRef = NULL;
ASErr result = sAITextFrame->GetATETextFrame(textFrameArt, &textFrameRef);
aisdk::check_ai_error(result);
ITextFrame textFrame(textFrameRef);
ITextRange textRange = textFrame.GetTextRange();
```

Examining other text-related sample code

The following samples in the Illustrator CS3 SDK also use the text API:

- MarkedObjects
- TextFileFormat

Using the text documentation in the API Reference

The Adobe text engine wrappers and Illustrator text suites are documented in the API Reference, <*SDK*>/docs/references/index.chm. The individual HTML files are provided in the sdkdocs.tar archive in the same folder.

To view the text documentation, follow these steps:

1. Open the compiled API reference file, index.chm.
2. In the API Contents page, click on the link for Adobe Text Engine (ATE). This takes you to the main Adobe text engine page, which provides further links to Adobe text engine wrappers and the Illustrator text suites.

To quickly find text-related types, follow these steps:

1. In the compiled API reference file, select the Index tab.
2. Type in the name of the type/class/function, etc. you are looking for. The list of items in the index automatically updates to show the closest matching items found.
3. Go to the item declaration, by double clicking on it in the index list.

Iterating through text

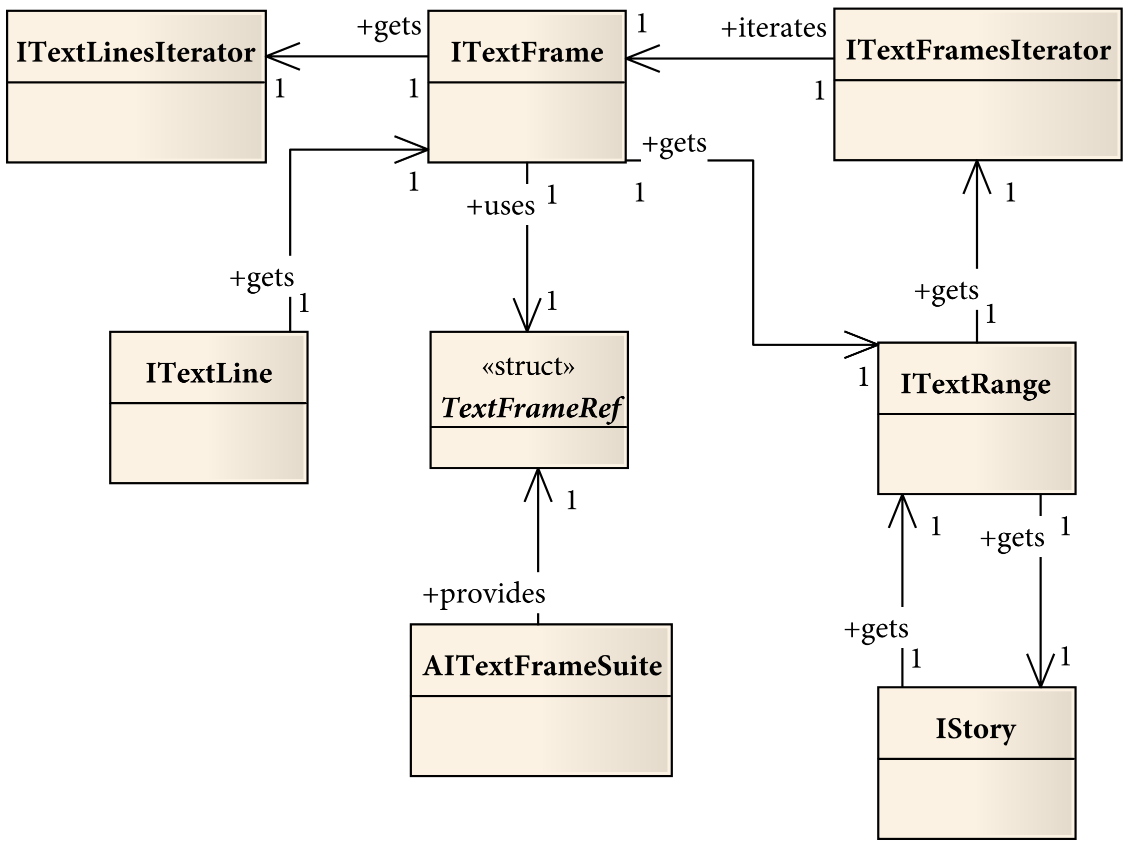
This section describes how to find and examine text objects—text frames, lines, stories, paragraphs, words, and characters—in Illustrator documents.

Iterating through text frames

A text frame is represented by an ITextFrame object. Its purpose is to control the layout of a text range into lines and columns. A text frame can contain several text lines and a text range, which is the text currently displayed in the text frame. When text frames are linked (threaded) together, the content they display comes from one associated story. See Figure 3.

NOTE: Text that overruns the text frame is not included in the contents of the text frame's text range.

FIGURE 3 *ITextFrame context*



Procedure

1. Find the text frames to iterate. You can do this via the current selection using AIDocumentSuite::GetTextSelection, then get the text frames from the text ranges. Alternately, you can get the selected kTextFrameArt using AIMatchingArtSuite::GetMatchingArt or by traversing the artwork tree.
2. If you are working with ITextRanges, visit each text frame in a text range using an ITextFramesIterator, by calling ITextRange::GetTextFramesIterator. If you are working with an art set, access each text frame through the art-set index.
3. If you are working with an AIArtHandle, use AITextFrameSuite::GetATETextFrame to get a TextFrameRef then construct a new ITextFrame object using the TextFrameRef.
4. Get the text range inside the text frame, using ITextFrame::GetTextRange.
5. Get the string contents, using ITextRange::GetContents.

API Reference

- AIDocumentSuite
- AIMatchingArtSuite
- ITextFrame
- AITextFrameSuite
- ITextRange

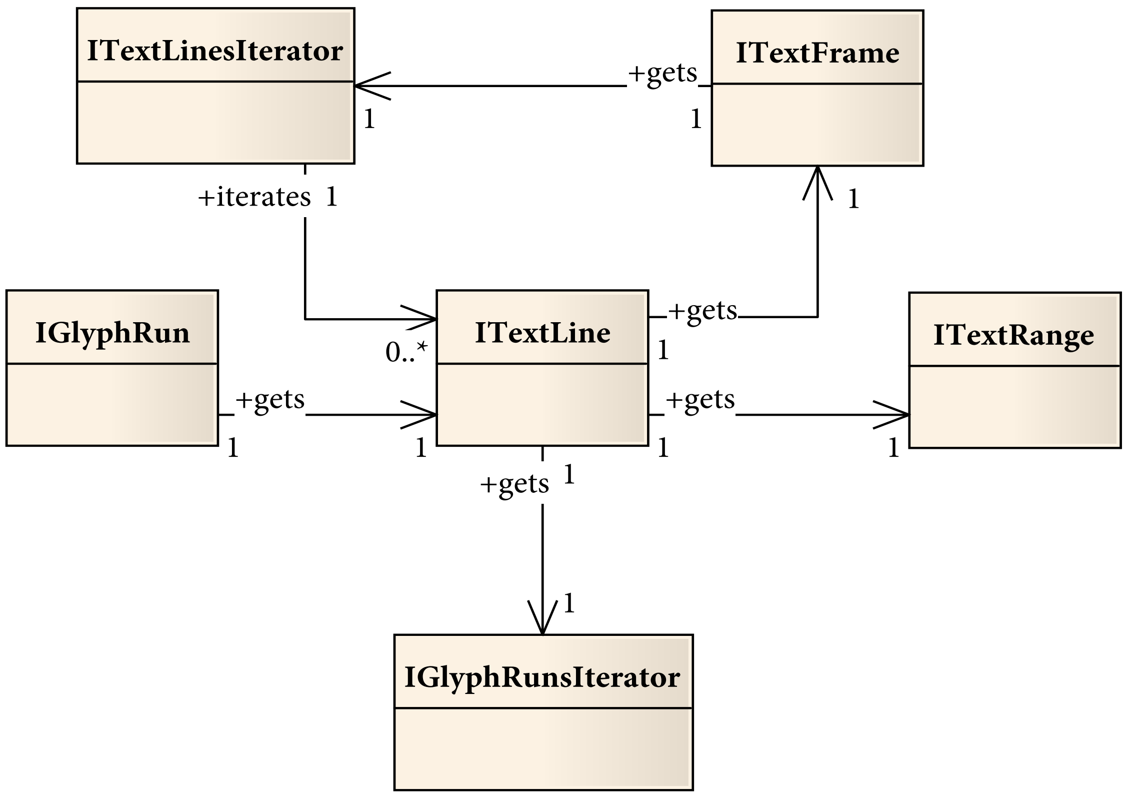
Sample code

- SnpText::LinkTextFrames
- SnpTextIterator::IterateSelectedTextFrames
- SnpTextIterator::IterateTextFrames

Iterating through lines

A line of text in a text frame is represented by an ITextLine object. See Figure 4.

FIGURE 4 ***ITextLine context***



Procedure

1. Get the text frame(s), by following the procedure in "Iterating through text frames" on page 12.
2. For each text frame, get the ITextLinesIterator using ITextFrame::GetTextLinesIterator, and use this object to iterate the text lines.
3. Get the text range in the line, using ITextLine::GetTextRange.
4. Get the string contents, using ITextRange::GetContents.

API Reference

- ITextFrame
- ITextFrames
- ITextFramesIterator
- ITextLine
- ITextLinesIterator
- ITextRange

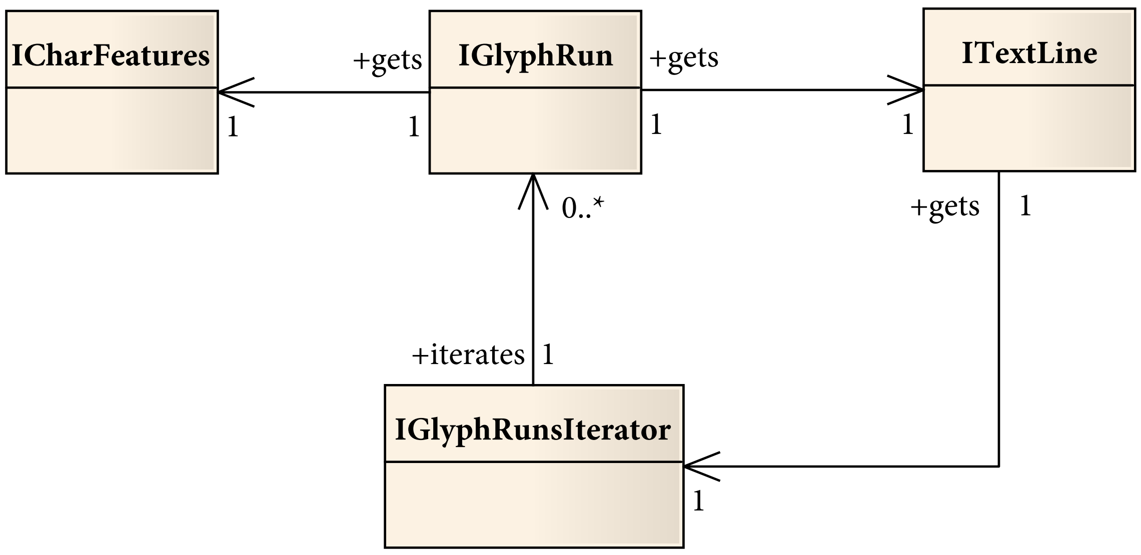
Sample code

- SnpTextIterator::IterateGlyphRuns
- SnpTextIterator::IterateLinesInSelectedFrames

Iterating through glyph runs

A *glyph run* is represented by an IGlyphRun object. It describes characters in a composed form that is ready to be drawn. See Figure 5.

FIGURE 5 *IGlyphRun context*



NOTE: An IGlyphRun differs from an IGlyph in that an IGlyph does not have a set of characters and is a document resource rather than a text container.

Procedure

1. Get the text lines, by following the procedure in "Iterating through lines" on page 14.
2. For each text line, get the IGlyphRunsIterator using ITextLine::GetGlyphRunsIterator.
3. Access each glyph using a while or for loop and IGlyphRunsIterator::IsNotDone, IGlyphRunsIterator::Item, and IGlyphRunsIterator::Next.
4. Get the string contents of each glyph run, using IGlyphRun::GetContents.

API Reference

- IGlyphRun
- IGlyphRunsIterator
- ITextFrame
- ITextFramesIterator
- ITextLine
- ITextLinesIterator

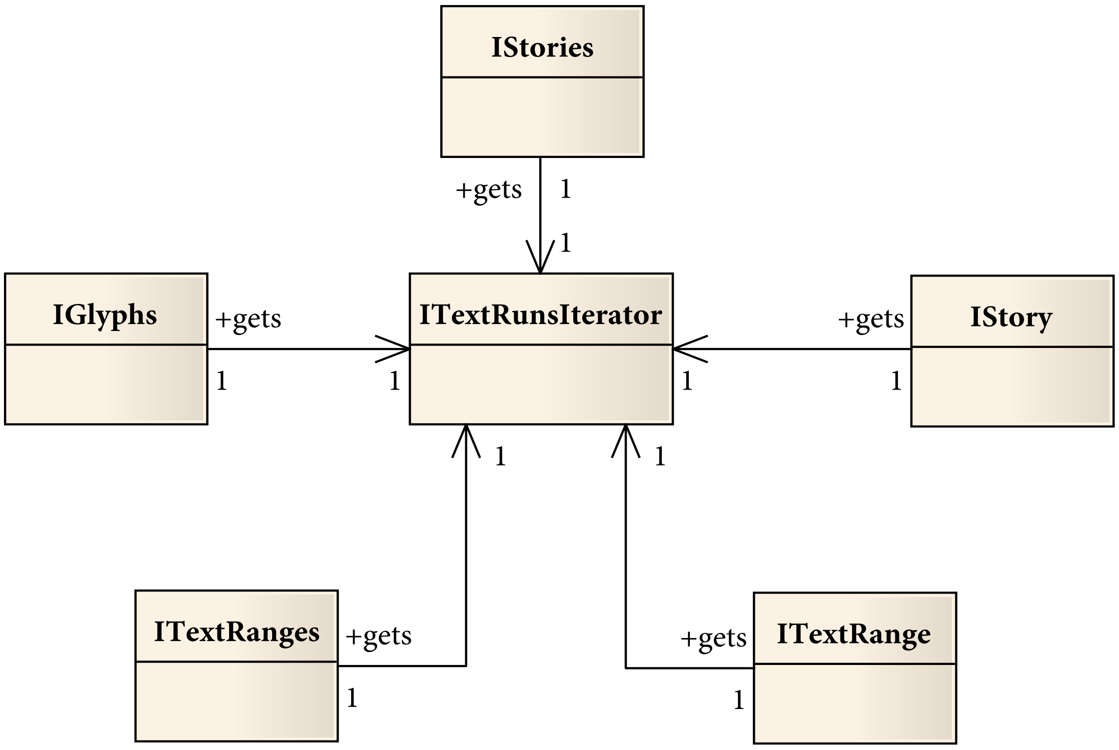
Sample code

SnpTextIterator::IterateGlyphRuns

Iterating through text runs

A *text run* is a range of text that shares one set of stylistic attributes. There is no object representing a single text run; however, there is an ITextRunsIterator object which can be accessed via ITextRanges, ITextRange, IStory, IStories, and IGlyphs objects and provides access to each text run in the containing object. See Figure 6.

FIGURE 6 ***ITextRunsIterator context***



Procedure

1. Find the text runs to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.

2. Get the ITextRunsIterator, using ITextRanges::GetTextRunsIterator.

3. Iterate through each of the text-run text ranges, using a while or for loop and ITextRunsIterator::IsNotDone, ITextRunsIterator::Item, and ITextRunsIterator::Next.

4. Get the string contents of the text range, using ITextRange::GetContents.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges
- ITextRunsIterator

Sample code

SnpTextIterator::IterateTextRuns

Iterating through characters

Procedure

1. Find the text range containing the characters to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Access the character at each index of the ITextRange object.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges

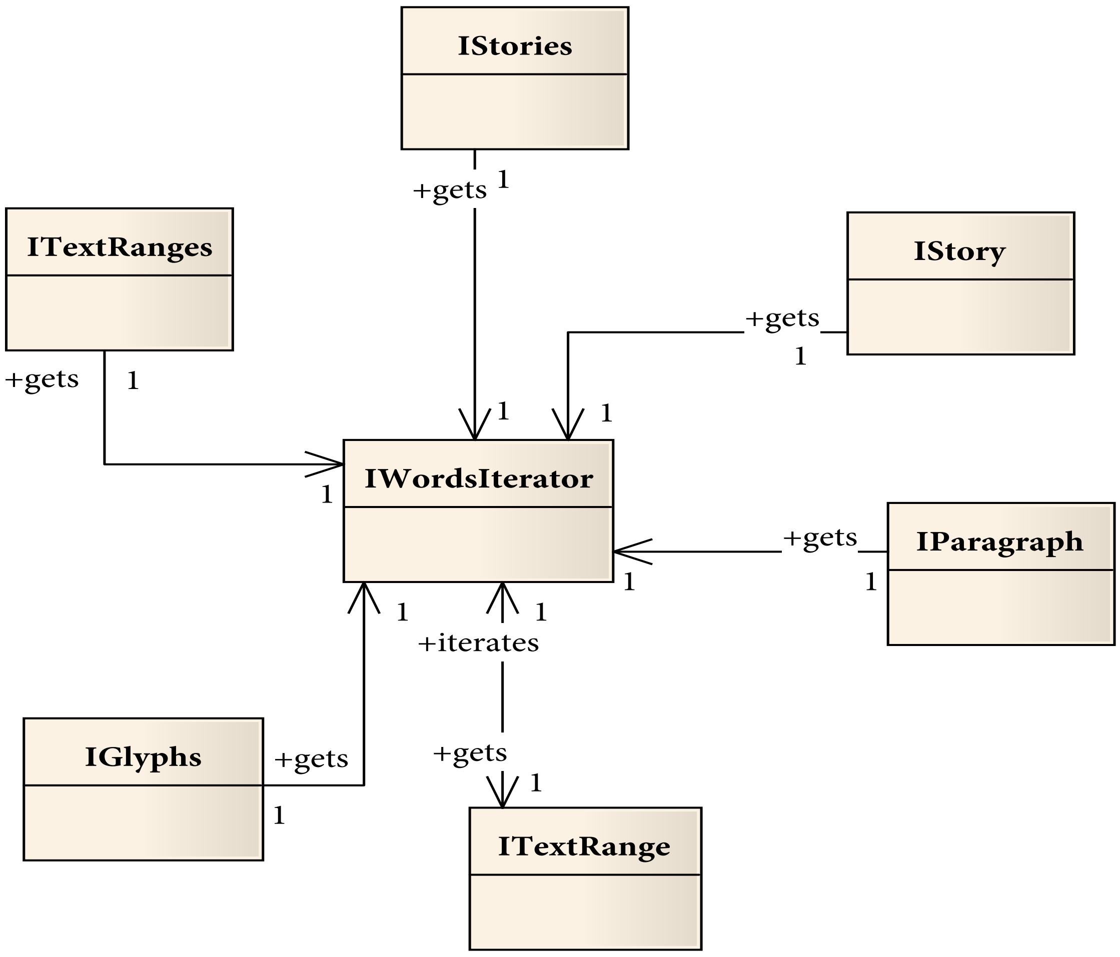
Sample code

SnpTextIterator::IterateSelectedCharacters

Iterating through words

The words within a text range, paragraph, story, or glyph can be accessed through an IWordsIterator object. See Figure 7.

FIGURE 7 *IWordsIterator context*



Procedure

1. Find the text range containing the words to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Iterate words using IWordsIterator.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges
- IWordsIterator

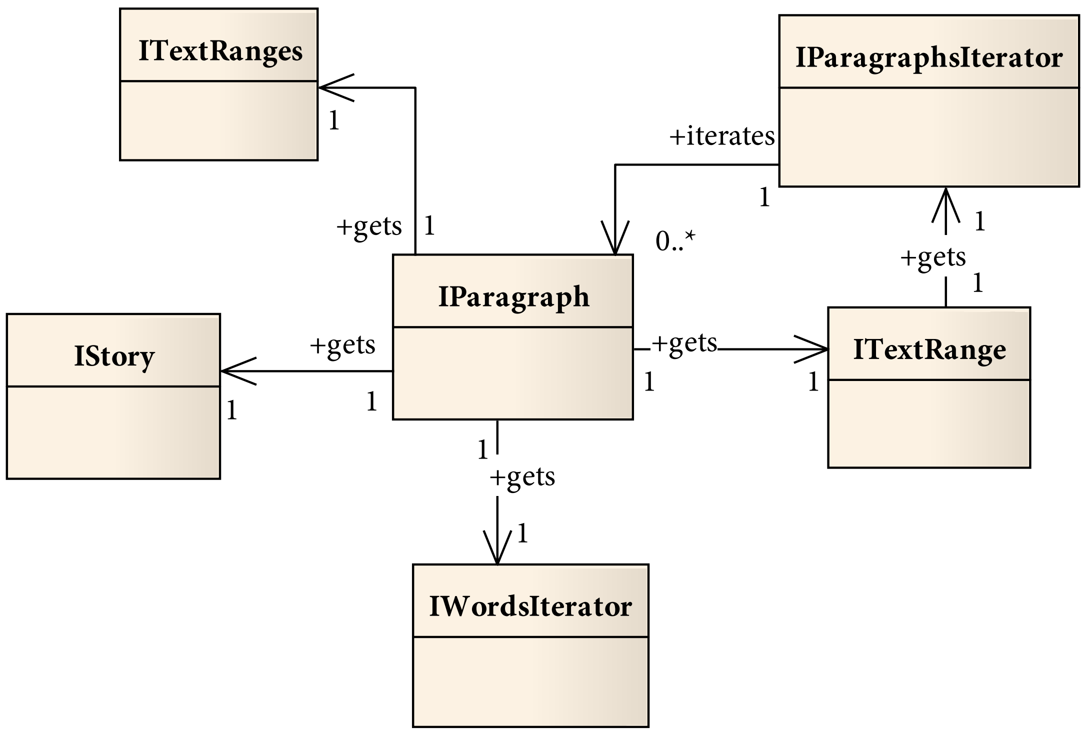
Sample code

SnpTextIterator::IterateSelectedWords

Iterating through paragraphs

A paragraph of text is represented by an IParagraph object, which can be obtained from an IStory, ITextRange, or ITextRanges object, and in turn contains IGlyphs and an IWordsIterator. The paragraphs in a text range can be accessed using an IParagraphsIterator. See Figure 8.

***FIGURE 8** IParagraph context*



Procedure

1. Find the text range containing the paragraphs to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.

2. Get an IParagraphsIterator object by calling a function that returns one, such as ITextRanges::GetParagraphsIterator or IStory::GetParagraphsIterator.

3. Iterate the paragraphs using IParagraphsIterator.

4. Get the string contents of a paragraph using IParagraph::GetContents.

API Reference

- AIDocumentSuite
- IParagraph
- IParagraphsIterator
- ITextRange
- ITextRanges

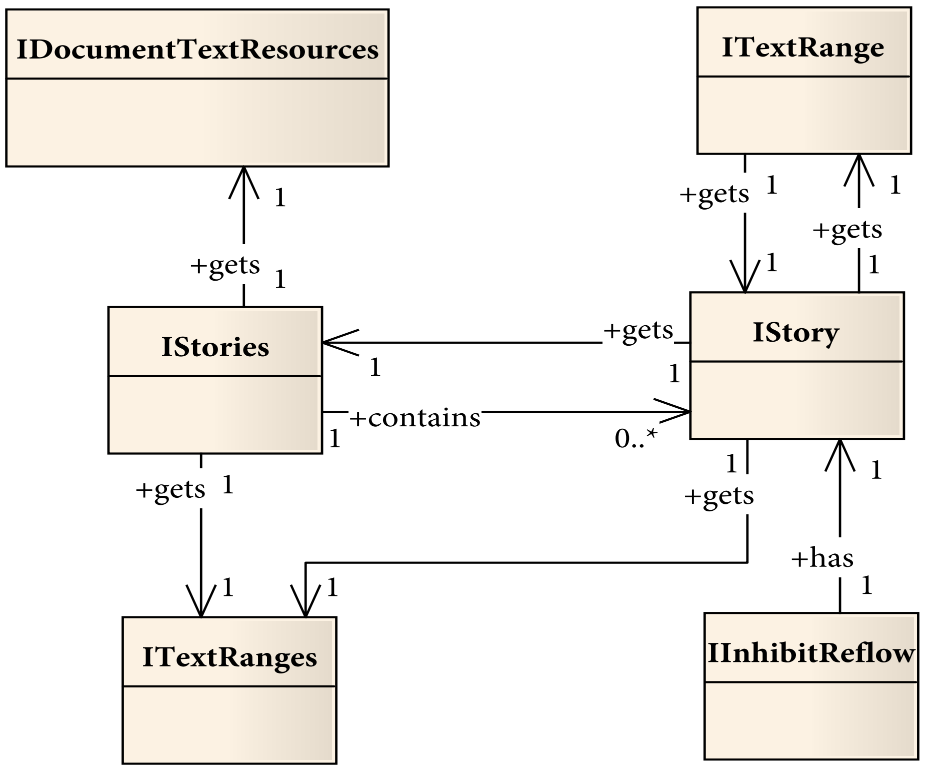
Sample code

SnpTextIterator::IterateSelectedParagraphs

Iterating through stories

A story is represented by an IStory object; a collection of stories, by an IStories object. A story contains text ranges, text frames, text runs, paragraphs, and words. An IStory object can be accessed through an ITextRange object using ITextRange::GetStory. See Figure 9.

FIGURE 9 *IStory/IStories context*



Procedure

1. Find the text ranges containing the stories to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Iterate through each text range, getting that text range's associated story using ITextRange::GetStory.
3. Get the string contents of the story: first get the entire text range of the story using IStory::GetTextRange, then get the text-range contents using ITextRange::GetContents.

API Reference

- AIDocumentSuite
- IStory
- ITextRange
- ITextRanges

Sample code

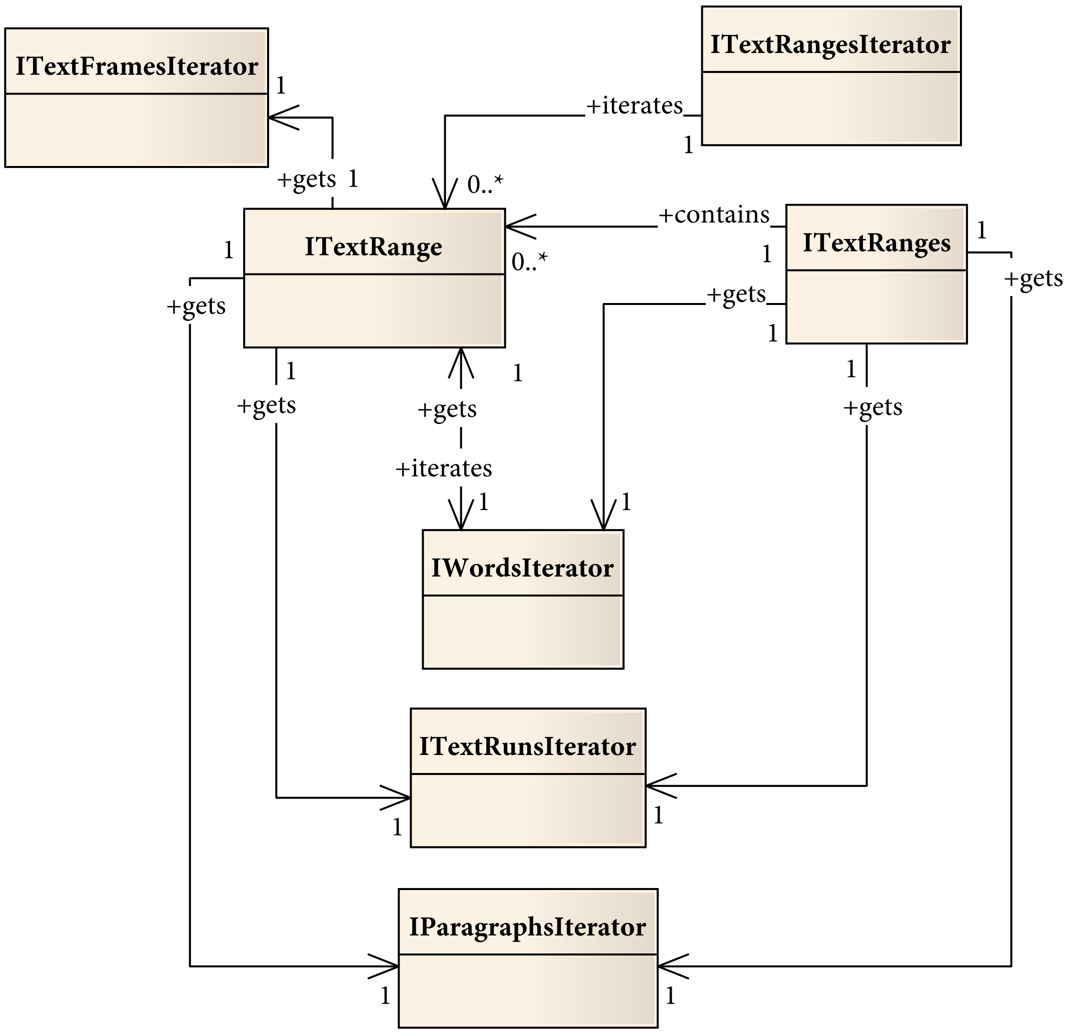
SnpTextIterator::IterateSelectedStories

Iterating through text ranges

A range of text is represented by an ITextRange object. It can flow across several text frames and can contain several paragraphs, words, characters, text runs, and glyphs. Each text range is contained within an IStory. See Figure 10.

Most text-range-related use cases are in "Using text" on page 25.

FIGURE 10 *ITextRange/ITextRanges text content context*



Procedure

1. Find the text ranges to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Access each ITextRange object by indexing the ITextRanges object.
3. Get the string contents of each text range using ITextRange::GetContents.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges

Sample code

- SnpText::DeleteTextRange
- SnpTextIterator::IterateGlyphRuns
- SnpTextIterator::IterateKernTypes
- SnpTextIterator::IterateSelectedStories
- SnpTextIterator::IterateSelectedTextFrames
- SnpTextIterator::IterateSelectedTextRanges
- SnpTextIterator::IterateTextRuns
- SnpTextStyles::ApplyCharacterStyle
- SnpTextStyles::ApplyParagraphStyle

Iterating through kern types

Procedure

1. Find the text range to iterate. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Since the kern types are managed at the story level, get the story using ITextRange::GetStory.
3. Iterate through each character in the story. For each character, get the kern type using IStory::GetModelKernAtChar.

API Reference

- AIDocumentSuite
- IStory
- ITextRange

Sample code

SnpTextIterator::IterateKernTypes

Using text

This section covers basic text-manipulation use cases, including creating, linking, and deleting text frames and inserting, deleting, copying, moving, and selecting text.

Creating point text

You can create a new point-text item in a document.

Procedure

1. Get the group in the layer you want to contain your new text object, using AIArtSuite::GetFirstArtOfLayer.
2. Add the new point-text object to the layer, using AITextFrameSuite::NewPointText.
3. Set the contents of the text range, using AITextFrameSuite::GetATETextRange and either ITextRange::InsertAfter or ITextRange::InsertBefore.

API Reference

- AIArtSuite
- AITextFrameSuite
- ITextRange

Sample code

SnpText::CreatePointText

Creating in-path text

You can create a new in-path text item in a document.

Procedure

1. Get the group in the layer you want to contain your new text object, using AIArtSuite::GetFirstArtOfLayer.
2. Create a new path item on the current layer, using AIArtSuite::NewArt.
3. Add the new in-path text item to the current layer using AITextFrameSuite::NewInPathText, and set its path item to the newly added path.
4. Set the contents of the text range using AITextFrameSuite::GetATETextRange and either ITextRange::InsertAfter or ITextRange::InsertBefore.

API Reference

- AIArtSuite
- AITextFrameSuite
- ITextRange

Sample code

- SnpText::ArtHandleFromRect
- SnpText::CreateInPathText

Creating on-path text

You can create a new on-path text item in a document.

Procedure

1. Get the group in the layer you want to contain your new text object, using AIArtSuite::GetFirstArtOfLayer.
2. Create a new path item on the current layer, using AIArtSuite::NewArt.
3. Add the new on-path text item to the current layer using AITextFrameSuite::NewOnPathText, and set its path item to the newly added path.
4. Set the contents of the text range using AITextFrameSuite::GetATETextRange and either ITextRange::InsertAfter or ITextRange::InsertBefore.

API Reference

- AIArtSuite
- AITextFrameSuite
- ITextRange

Sample code

- SnpText::ArtHandleFromArc
- SnpText::CreateOnPathText

Creating threaded in-path text

You can create several linked in-path text items in a document which display a single story.

You can link text frames to allow one story to be associated with more than one text frame. Once linked, the story text is displayed across all the frames.

Procedure

1. Get the group in the layer you want to contain your new text object, using AIArtSuite::Get-FirstArtOfLayer.
2. Create a new path item on the current layer to display the start of your threaded text, using AIArtSuite::NewArt.
3. Add the new in-path text item to the current layer using AITextFrameSuite::NewInPath-Text, and set its path item to the newly added path.
4. Set the contents of the text range using AITextFrameSuite::GetATETextRange and either ITextRange::InsertAfter or ITextRange::InsertBefore.
5. Create another path item on the current layer to continue displaying the threaded text, using AIArtSuite::NewArt.
6. Add another in-path text item to the current layer, using AITextFrameSuite::NewInPath-Text. Set its path item to the newly added path, and set its prep and base frame to the previous in-path text item.
7. Repeat steps 5 and 6 for each path item you want to continue displaying the text story.

API Reference

- AIArtSuite
- AITextFrameSuite
- ITextRange

Sample code

- SnpText::ArtHandleFromRect
- SnpText::CreateThreadedInPathText

NOTE: To create threaded on-path text, follow the same steps as above, but replace all occurances of NewInPathText with NewOnPathText, and add the extra parameters for the start and end segments.

Selecting text

You can highlight a range of text in the current document using ITextRanges::Select or ITextRange::Select.

Procedure

1. To highlight text in the currently selected text frame, follow the instructions in "Accessing text using selection" on page 11 to find the selected text range.
2. To highlight any text range, regardless of whether it is selected in the current document, follow the instructions in "Accessing text using the artwork tree" on page 11 to find a text range.

3. Once you have an ITextRanges or ITextRange object, select the text in the text range using ITextRanges::Select or ITextRange::Select, respectively.

API Reference

- AIDocumentSuite
- ITextRanges

Sample code

- SnpText::SelectTextRange
- SnpTextIterator::IterateSelectedTextFrames
- SnpTextIterator::IterateTextFrames

Setting text focus

If a document has text focus, it is ready for text to be input into a text frame. This is different than selecting text, as selecting text does not necessarily mean the document has text focus. For example, calling ITextRange::Select highlights the text in a text range but does not set the text focus. Text focus is set through the AIDocumentSuite at the story level.

Procedure

1. Find the text range to give text focus to. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Ensure the current document does not already have text focus, by selecting a non-text tool in the tool palette using AIToolSuite::SetSelectedTool.
3. Get the story containing the text range to gain text focus, using ITextRange::GetStory.
4. Set the text focus to the beginning of the story using AIDocumentSuite::SetTextFocus, passing in the StoryRef.

API Reference

- AIDocumentSuite
- AIToolSuite
- IStory
- ITextRange
- ITextRanges

Sample code

SnpText::SetTextFocus

Losing text focus

You can remove text focus from a document.

Procedure

1. Find out if the current document has text focus, using AIDocumentSuite::HasTextFocus.
2. Lose the text focus using AIDocumentSuite::LoseTextFocus.

API Reference

AIDocumentSuite

Sample code

SnpText::LoseTextFocus

Inserting text

You can insert a range of text into a selected text range.

Procedure

1. Find the text range where the text is to be inserted. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Once the correct area is found, insert the text using ITextRange::InsertBefore or ITextRange::InsertAfter.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges

Sample code

SnpText::InsertText

Copying text

You can copy a selected range of text to a new text item.

Procedure

1. Find the text range to be copied. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the text frame associated with the text range, using ITextRange::GetStory::GetFrame, ITextFrame::GetRef, and AITextFrameSuite::GetAITextFrame.
3. Create a new path item to contain the copied text frame using AIArtSuite::NewArt. USe dimensions similar to the text frame containing the text range being copied.
4. Create a new text item to contain the copied range, using AITextFrameSuite::NewInPathText, AITextFrameSuite::NewPointText, or AITextFrameSuite::NewOnPathText.
5. Copy the text range using ITextRange::Clone.
6. Insert the text in the copied text frame, using AITextFrameSuite::GetATETextRange and either ITextRange::InsertAfter or ITextRange::InsertBefore.

API Reference

- AIDocumentSuite
- AITextFrameSuite
- ITextFrame
- ITextRange
- ITextRanges

Sample code

- SnpText::ArtHandleFromRect
- SnpText::CopyText

Moving text

You can move a selected range of text from one text item to a new text item.

Procedure

1. Find the text range to be moved. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. To move the text range within the current story bounds, use ITextRange::Move, specifying the number of units the range should be moved in a positive (toward the end) or negative

(toward the start) direction. To move the text range into a new or existing story, follow the instructions in "Copying text" on page 30, but with the added step of deleting the text range from its original position using ITextRange::Remove.

API Reference

- AIDocumentSuite
- AITextFrameSuite
- ITextFrame
- ITextRange
- ITextRanges

Sample code

- SnpText::ArtHandleFromRect
- SnpText::MoveText

Replacing text

You can replace the selected text range with a new text range.

Procedure

1. Find the text range to be replaced. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Remove the text range using ITextRange::Remove.
3. Insert the new text range using ITextRange::InsertBefore or ITextRange::InsertAfter.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges

Sample code

SnpText::ReplaceText

Deleting text

You can delete the selected text range.

Procedure

1. Find the text range to be deleted. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Remove the text range using ITextRange::Remove.

API Reference

- AIDocumentSuite
- ITextRange
- ITextRanges

Sample code

SnpText::DeleteTextRange

Linking text frames

You can create a link between the selected text frames; their contents become a single story.

Procedure

1. Find the text frames to link. You can do this either via the current selection using AIArtSet-Suite::MatchingArtSet with an AIArtSpec specifying selected kTextFrameArt, or by tra-versing the artwork tree.
2. Link a text frame with another using AITextFrameSuite::Link and passing in the two text frames to link.

API Reference

- AIArtSetSuite
- AIArtSpec
- AITextFrameSuite

Sample code

SnpText::LinkTextFrames

Unlinking text frames

You can remove any links between the selected text frames and split the text frames' contents into separate stories.

Procedure

1. Find the text frames to unlink. You can do this either via the current selection using AIArtSet::MatchingArtSet with an AIArtSpec specifying selected kTextFrameArt, or by traversing the artwork tree.
2. Check whether the text frame is linked to another frame, using AITextFrameSuite::PartOfLinkedText and passing in the text frame in question.
3. Unlink the text frame using AITextFrameSuite::Unlink.

API Reference

- AIArtSet
- AIArtSpec
- AITextFrameSuite

Sample code

SnpText::UnlinkTextFrames

Deleting text frames

You can delete a text-frame art item from a document.

Procedure

1. Find the text frame to delete. You can do this via the current selection using AIDocumentSuite::GetTextSelection, then either getting the text frames from the text ranges or getting the selected kTextFrameArt using AIMatchingArtSuite::GetMatchingArt. Alternately, you can traverse the artwork tree.
2. If you are working with an ITextFrame object, get the AIArtHandle for the text frame by first calling ITextFrame::GetRef, then AITextFrameSuite::GetAITextFrame.
3. If you are working with an art object, delete it using AIArtSuite::DisposeArt.

NOTE: As a text frame is an art object, delete the object using the AIArtSuite.

API Reference

- AIArtSuite
- AIDocumentSuite
- AITextFrameSuite
- ITextFrame

Sample code

- SnpText::LinkTextFrames
- SnpText::UnlinkTextFrames
- SnpTextIterator::IterateSelectedTextFrames

Converting legacy text

You can convert legacy text in a document to work with the new text APIs.

Procedure

1. Convert all legacy text in the current document using AILegacyTextConversionSuite::ConvertAllToNative, or convert a single text item using AILegacyTextConversionSuite::ConvertToNative.

API Reference

AILegacyTextConversionSuite

Sample code

SnpText::ConvertLegacyText

Setting kern type

A kern type is an Illustrator constant that refers to the algorithm used to calculate the spacing between characters. There are three types of kerning:

- *kNoAutoKern* — There is no automatic altering of the spacing between characters to improve their appearance together.
- *kMetricKern* — Uses a metrics table to determine the amount of space each character requires.
- kOpticalKern — Uses the glyphs' shapes to kern the characters as they appear to the eye.

Procedure

1. Find the text range to edit. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. As the kern types are managed at the story level, get the story using ITextRange::GetStory.
3. Set the kern type of the text range of the story using IStory::SetKernForSelection.

API Reference

- AIDocumentSuite
- IStory
- ITextRange

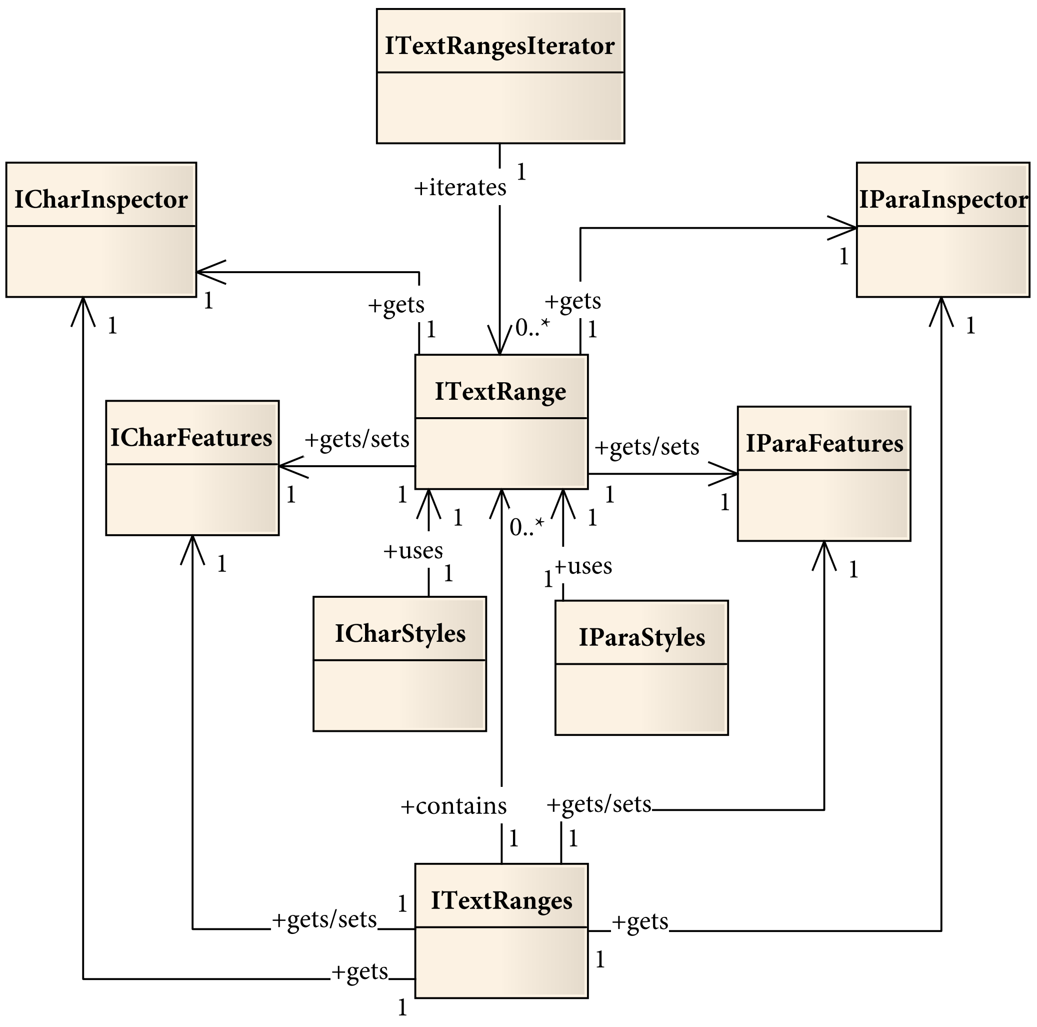
Sample code

SnpTextStyler::SetKernType

Styling text

This section describes how to examine, create, update, and delete character and paragraph styles and how to style text. See Figure 11.

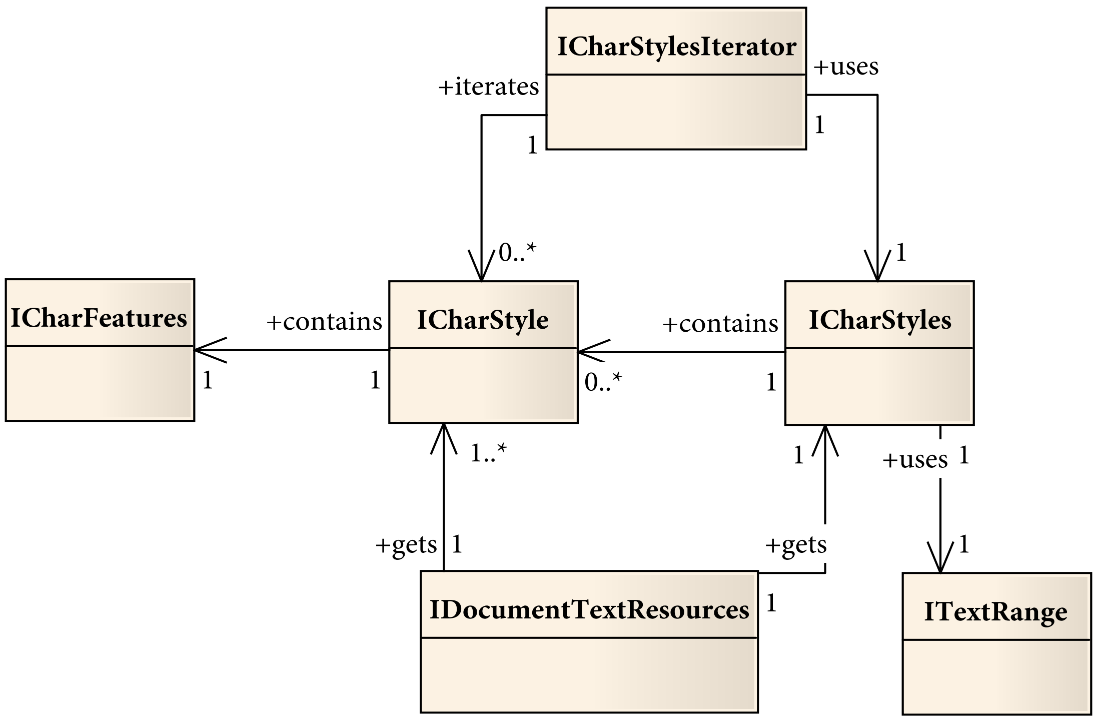
FIGURE 11 ***ITextRange/ITextRanges text-styling context***



Iterating through character styles

The character styles associated with a document are contained in an ICharStyles set that contains zero or more ICharStyle objects. These objects can be accessed through the ITextRange and IDocumentTextResources objects. To access the entire set of character styles for a document, use IDocumentTextResources; to access the character styles applied to particular text ranges, use ITextRange. See Figure 12.

FIGURE 12 *ICharStyle/ICharStyles context*



A named character style contains an ICharFeatures object, which contains the attributes to be applied to the characters. When applied, it overrides the character attributes inherited from the Normal character style.

Procedure

1. Get the current document text resources set using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
2. Get the documents' ICharStyles object using IDocumentTextResources::GetCharStylesInDocument.
3. Create a new ICharStylesIterator using the ICharStyles object.
4. Iterate through each ICharStyle in the ICharStyles object using ICharStylesIterator::MoveToFirst, ICharStylesIterator::Item, and ICharStylesIterator::Next.

API Reference

- AIDocumentSuite
- ICharStyle
- ICharStyles

- ICharStylesIterator
- IDocumentTextResources

Sample code

SnpTextStyles::IterateCharacterStyles

Creating a character style

You can add a new named character style to a document's text resources.

Procedure

1. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
2. Create a new ICharStyle object using IDocumentTextResources::CreateCharStyle.
3. Create a new ICharFeatures object, and set the desired features using the ICharFeatures' members.
4. Set the features of the ICharStyle using ICharStyle::SetFeatures, passing in the ICharFeatures object.

API Reference

- AIDocumentSuite
- ICharFeatures
- ICharStyle
- IDocumentTextResources

Sample code

SnpTextStyles::CreateCharacterStyle

Getting the current character style

You can find the character style currently in use in a document.

Procedure

1. Get the CharStyleRef to the current style applied to new text items, using AIATECurrentTextFeaturesSuite::GetCurrentCharStyle.
2. Create a new ICharStyle object from the CharStyleRef.
3. Access the features of the style in use using ICharStyle::GetFeatures, or access the name using ICharStyle::GetName.

API Reference

- AIATECurrentTextFeaturesSuite
- ICharStyle

Sample code

SnpTextStyles::GetCurrentCharacterStyle

Deleting a character style

You can delete a named character style from a document's text resources.

Procedure

1. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
2. Delete the desired ICharStyle using IDocumentTextResources::RemoveCharStyle, passing in the style name as a parameter.

API Reference

- AIDocumentSuite
- ICharStyle
- IDocumentTextResources

Sample code

SnpTextStyles::DeleteCharacterStyle

Applying a character style

You can apply a named character style to a range of text.

Procedure

1. Find the text range to apply the character style to. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
3. Get the ICharStyle you want applied to the text range using IDocumentTextResources::GetCharStyle, passing in the name of the character style.
4. Apply the character style to the text range using ITextRange::SetNamedCharStyle.

API Reference

- AIDocumentSuite
- ICharStyle
- IDocumentTextResources
- ITextRange

Sample code

SnpTextStyles::ApplyCharacterStyle

Clearing a character style

You can clear a named character style from a range of text.

Procedure

1. Find the text range from which to clear the character style. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Clear the character style from the text range, using ITextRange::ClearNamedCharStyle.

NOTE: Clearing the character style from a text range using this function only disassociates the text range with the character style,. The character features are still applied to the text range.

3. Clear the overriding character features, returning the text range to the Normal character style using ITextRange::ClearLocalCharFeatures.

API Reference

- AIDocumentSuite
- ITextRange

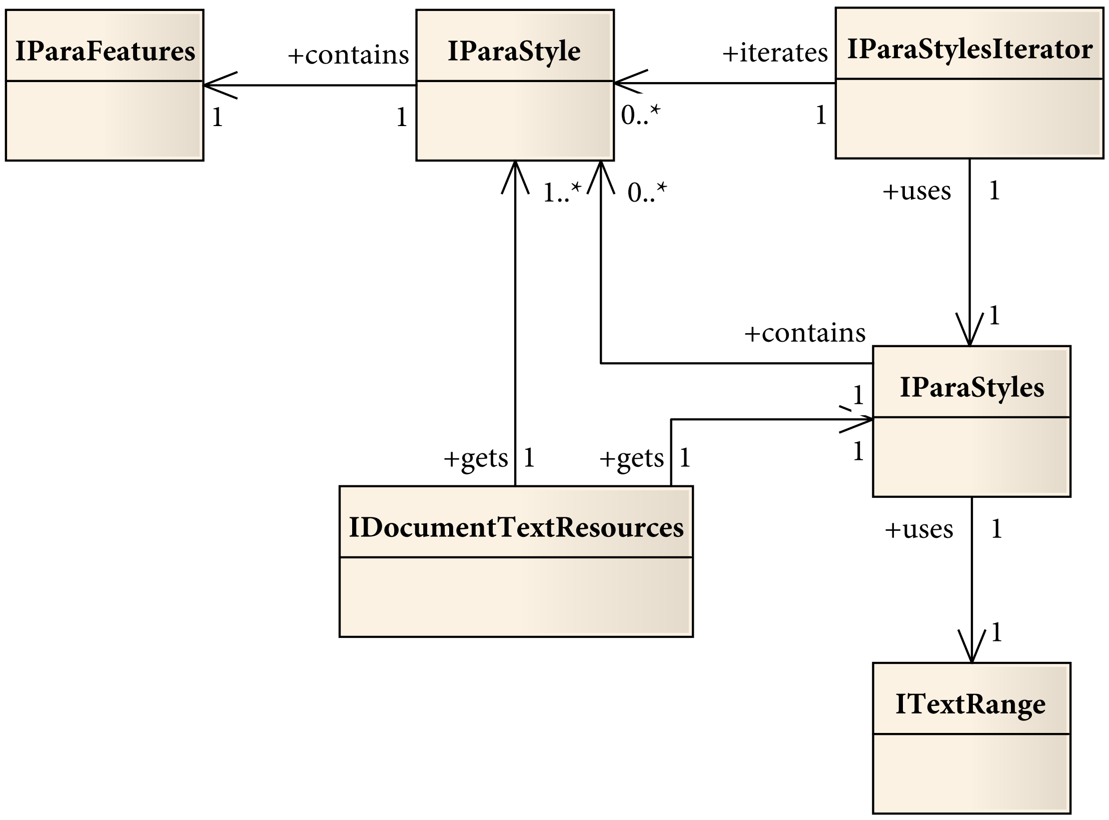
Sample code

SnpTextStyles::ClearCharacterStyle

Iterating through paragraph styles

The paragraph styles associated with a document are contained in an IParaStyles set, which contains zero or more IParaStyle objects. These objects can be accessed through the ITextRange and IDocumentTextResources objects. To access the entire set of paragraph styles for a document, use IDocumentTextResources; to access the paragraph styles applied to particular text ranges, use ITextRange. See Figure 13.

FIGURE 13 ***IParaStyle/IParaStyles context***



A named paragraph style contains an IParaFeatures object that contains the attributes to be a applied to the paragraphs. When applied, it overrides the paragraph attributes inherited from the Normal paragraph style.

Procedure

1. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
2. Get the document's IParaStyles object, using IDocumentTextResources::GetParaStylesInDocument.
3. Create a new IParaStylesIterator, using the IParaStyles object.
4. Iterate through each IParaStyle in the IParaStyles object, using IParaStylesIterator::MoveToFirst, IParaStylesIterator::Item, and IParaStylesIterator::Next.

API Reference

- AIDocumentSuite
- IDocumentTextResources
- IParaStyle
- IParaStyles
- IParaStylesIterator

Sample code

SnpTextStyles::IterateParagraphStyles

Creating a paragraph style

You can add a new named paragraph style to a document's text resources.

Procedure

1. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
2. Create a new IParaStyle object, using IDocumentTextResources::CreateParaStyle.
3. Create a new IParaFeatures object, and set the desired features using the IParaFeatures' members.
4. Set the features of the IParaStyle using IParaStyle::SetFeatures, passing in the IParaFeatures object.

API Reference

- AIDocumentSuite
- IDocumentTextResources
- IParaFeatures
- IParaStyle

Sample code

SnpTextStyles::CreateParagraphStyle

Getting current paragraph style

You can find the paragraph style currently in use in the current document.

Procedure

1. Get the ParaStyleRef to the current style applied to new text items, using AIATECurrent-TextFeaturesSuite::GetCurrentParaStyle.
2. Create a new IParaStyle object from the ParaStyleRef.
3. Access the features of the style in use using IParaStyle::GetFeatures, or access the name using IParaStyle::GetName.

API Reference

- AIATECurrentTextFeaturesSuite
- IParaStyle

Sample code

SnpTextStyles::GetCurrentParagraphStyle

Deleting a paragraph style

You can delete a named paragraph style from a document's text resources.

Procedure

1. Get the current document text resources set, using AIDocumentSuite::GetDocumentTex-tResources to get the DocumentTextResourcesRef, and then create a new IDocumentTex-tResources object using the DocumentTextResourcesRef.
2. Delete the desired IParaStyle using IDocumentTextResources::RemoveParaStyle, passing in the style name as a parameter.

API Reference

- AIDocumentSuite
- IDocumentTextResources
- IParaStyle

Sample code

SnpTextStyles::DeleteParagraphStyle

Applying a paragraph style

You can apply a named paragraph style to a range of text.

Procedure

1. Find the paragraph to which to apply the paragraph style. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the current document text resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
3. Get the IParaStyle you want applied to the paragraph using IDocumentTextResources::GetParaStyle, passing in the name of the paragraph style.
4. Apply the paragraph style to the text range, using ITextRange::SetNamedParaStyle.

API Reference

- AIDocumentSuite
- IDocumentTextResources
- IParaStyle
- ITextRange

Sample code

SnpTextStyles::ApplyParagraphStyle

Clearing a paragraph style

You can clear a named paragraph style from a range of text.

Procedure

1. Find the paragraph from which to clear the paragraph style. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Clear the paragraph style from the text range, using ITextRange::ClearNamedParaStyle.

NOTE: Clearing the paragraph style from a text range using this function only disassociates the text range with the paragraph style. The paragraph features are still applied to the text range.

3. Clear the overriding paragraph features, returning the text range to the Normal character style using ITextRange::ClearLocalParaFeatures.

API Reference

- AIDocumentSuite
- ITextRange

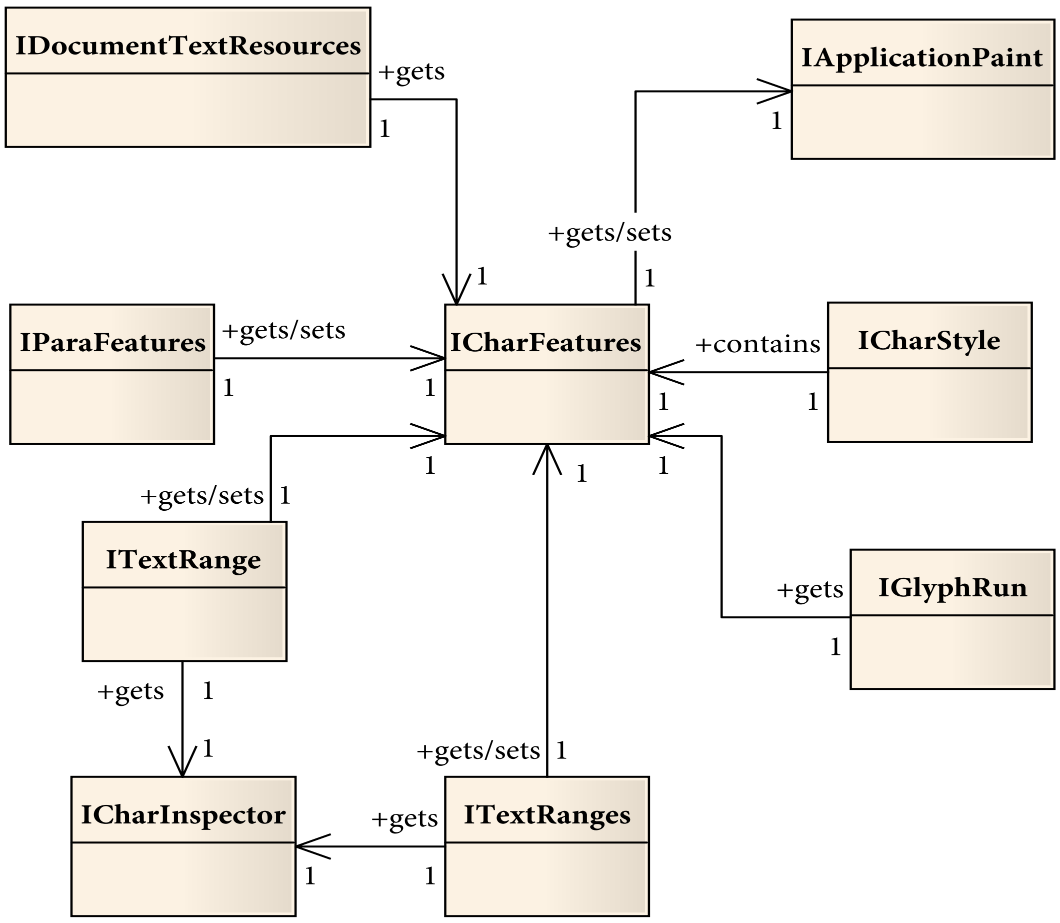
Sample code

SnpTextStyles::ClearParagraphStyle

Getting character features

Characters initially inherit the Normal style, but these features can be overridden at the character or character-style level. This use case examines the styling applied to a range of characters. See Figure 14.

FIGURE 14 ***ICharFeatures and ICharInspector context***



Procedure

1. Find the text range containing the characters whose features you want. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the character features used in the text range, using either ITextRange::GetUniqueCharFeatures (to get the character features that have the same value across all text runs in the text range) or ITextRange::GetUniqueLocalCharFeatures (to get the overriding features that have the same value across all text runs in the text range). Alternately, use ITextRange::GetCharInspector to return an ICharInspector object that provides access to the features of all characters in the text range.
3. Use the returned ICharFeatures or ICharInspector object to access and edit the individual features.

API Reference

- AIDocumentSuite
- ICharFeatures
- ICharInspector
- ITextRange
- ITextRanges

Sample code

- SnpTextStyler::GetCharacterFeatures
- SnpTextStyler::InspectSelectedCharacterFeatures

Applying character features

You can apply a set of styling attributes to a range of characters.

Procedure

1. Find the range of characters whose features you want to edit. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Create an ICharFeatures object, and use this object's members to set the features you want.
3. Apply the features to the range of characters using ITextRange::SetLocalCharFeatures, passing in your feature set.

NOTE: Only the features specified in the ICharFeatures set are modified; other features are unchanged.

API Reference

- AIDocumentSuite
- ICharFeatures
- ICharStyle
- ITextRange

Sample code

SnpTextStyler::ApplyLocalCharacterFeatures

Clearing character features

You can clear styling attributes from a range of characters.

Procedure

1. Find the range of characters whose features you want to edit. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Clear the local features applied to the text range using ITextRange::ClearLocalCharFeatures, returning the features to the Normal character style.
3. To remove a single feature, create a new ICharFeatures object, set the feature you want to be removed to its value in the Normal character style, then apply the feature set using ITextRange::SetLocalCharFeatures.

API Reference

- AIDocumentSuite
- ICharFeatures
- ITextRange

Sample code

SnpTextStyler::ClearLocalCharacterFeatures

Setting current character overrides

You can set the overriding styling attributes applied to new characters.

Procedure

1. Create a new ICharFeatures object, and set the desired individual features using the ICharFeatures member functions.
2. Get the CharFeaturesRef object from the ICharFeatures object, using ICharFeatures::GetRef.

3. Set the features to be applied to new text items, using AIATECurrentTextFeatures-Suite::SetCurrentCharOverrides.

API Reference

- AIATECurrentTextFeaturesSuite
- ICharFeatures

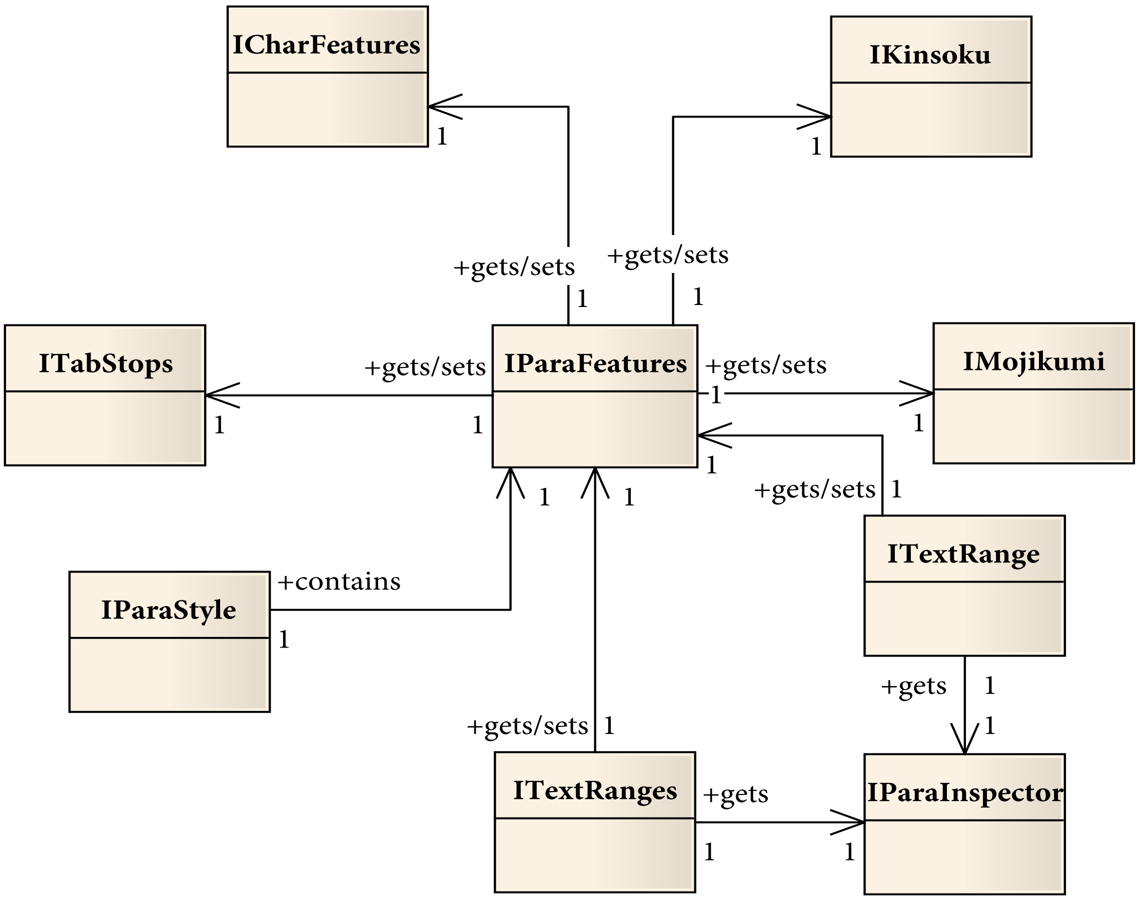
Sample code

SnpTextStyler::SetCurrentCharacterOverrides

Getting paragraph features

Paragraphs initially inherit the Normal style, but these features can be overridden at the paragraph or paragraph-style level. This use case examines the styling applied to a structured paragraph of text. See Figure 15.

FIGURE 15 ***IParaFeatures and IParaInspector context***



Procedure

1. Find the text range containing the paragraphs whose features you want. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the paragraph features used in the text range, using either ITextRange::GetUniqueParaFeatures (to get the paragraph features used in the text range that have the same value across all text runs in the text range) or ITextRange::GetUniqueLocalParaFeatures (to get the overriding features that have the same value across all text runs in the text range). Alternately, use ITextRange::GetParaInspector to return an IParaInspector object that provides access to the features of all paragraphs in the text range.
3. Use the returned IParaFeatures or IParaInspector object to access and edit the individual features.

API Reference

- AIDocumentSuite
- IParaFeatures
- IParaInspector
- ITextRange
- ITextRanges

Sample code

SnpTextStyler::InspectSelectedParagraphFeatures

Applying paragraph features

You can apply a set of styling attributes to a set of paragraphs.

Procedure

1. Find the range of paragraphs whose features you want to edit. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Create an IParaFeatures object, and use this object's members to set the features you want.
3. Apply the features to the range of paragraphs using ITextRange::SetLocalParaFeatures, passing in your feature set.

NOTE: Only the features specified in the IParaFeatures set are modified; other features are unchanged.

API Reference

- AIDocumentSuite
- IParaFeatures
- IParaStyle
- ITextRange

Sample code

SnpTextStyler::ApplyLocalParagraphFeatures

Clearing paragraph features

You can clear a set of styling attributes from a set of paragraphs.

Procedure

1. Find the range of paragraphs whose features you want to edit. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Clear the local features applied to the text range using ITextRange::ClearLocalParaFeatures, returning the features to the Normal paragraph style.
3. To remove a single feature, create a new IParaFeatures object, set the feature you want to be removed to its value in the Normal paragraph style, and then apply the feature set using ITextRange::SetLocalParaFeatures.

API Reference

- AIDocumentSuite
- IParaFeatures
- ITextRange

Sample code

SnpTextStyler::ClearLocalParagraphFeatures

Setting current paragraph overrides

You can set the overriding styling attributes applied to new paragraphs.

Procedure

1. Create a new IParaFeatures object, and set the desired individual features using the IParaFeatures' member functions.
2. Get the ParaFeaturesRef object from the IParaFeatures object, using IParaFeatures::GetRef.

3. Set the paragraph features to be applied to new text items, using AIATECurrentTextFeaturesSuite::SetCurrentParaOverrides.

API Reference

- AIATECurrentTextFeaturesSuite
- IParaFeatures

Sample code

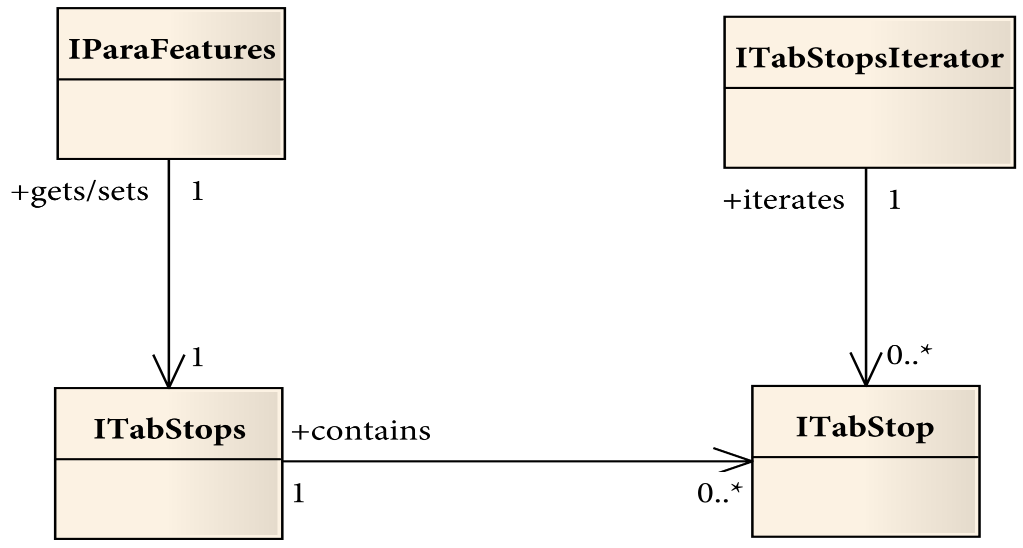
SnpTextStyler::SetCurrentParagraphOverrides

Adding tab stops

You can add tab stops to a paragraph or set of paragraphs.

A tab stop is represented by an ITabStop object. Tab stops are added to and removed from paragraphs through an IParaFeatures object. A set of tab stops is represented by an ITabStops object and can be iterated using an ITabStopsIterator. See Figure 16.

FIGURE 16 *ITabStop/ITabStops context*



Procedure

1. Find the paragraph to which the tab stop should be added. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.

2. Create a new ITabStop object.

3. Set the position of the new tab stop using ITabStop::SetPosition.
4. Create a new ITabStops object.
5. Add the tab stop to the new tab stops set, using ITabStops::ReplaceOrAdd.
6. Get the text range of the paragraph, using IParagraph::GetTextRange.
7. Get the IParaFeatures of the text range, using ITextRange::GetUniqueParaFeatures.
8. Set the tab stops attribute of the text-range features, using IParaFeatures::SetTabStops.
9. Apply the new features to the text range, using ITextRange::SetLocalParaFeatures.

API Reference

- AIDocumentSuite
- IParaFeatures
- IParagraph
- ITabStop
- ITabStops
- ITextRange

Sample code

SnpTextStyler::AddTabStops

Removing tab stops

You can remove tab stops from a paragraph or set of paragraphs.

Procedure

1. Find the paragraph from which to remove the tab stop. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the paragraph text range, using IParagraph::GetTextRange.
3. Get the text range's IParaFeatures object, using ITextRange::GetUniqueParaFeatures.
4. Get the ITabStops object associated with the text range, using IParaFeatures::GetTabStops.
5. Remove a single tab stop using ITabStops::Remove, passing in the index of the tab stop to remove, or remove all tab stops using ITabStops::RemoveAll.
6. Set the tab stops of the IParaFeatures object to the edited tab-stop set, using IParaFeatures::SetTabStops.
7. Apply the edited feature set to the paragraph text range, using ITextRange::SetLocalParaFeatures.

API Reference

- AIDocumentSuite
- IParaFeatures
- IParagraph
- ITabStops
- ITextRange

Sample code

SnpTextStyler::RemoveTabStops

Inspecting tab stops

You can find the tab stop in a paragraph or set of paragraphs.

Procedure

1. Find the paragraph to inspect. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the text range of the paragraph, using IParagraph::GetTextRange.
3. Get the IParaInspector object for the text range, using ITextRange::GetParaInspector.
4. Get the array of tab stops objects using IParaInspector::GetTabStops, returning an IArrayTabStopsRef object.
5. Iterate through the IArrayTabStopsRef object, getting each ITabStops object using IArrayTabStopsRef::Item.
6. Iterate through each item of the ITabStops set, getting each individual ITabStop object using ITabStops::Item.
7. Get the tab-stop information, using the members provided in the ITabStop class.

API Reference

- AIDocumentSuite
- IArrayTabStopsRef
- IParaInspector
- IParagraph
- ITabStop
- ITabStops
- ITextRange

Sample code

SnpTextStyler::InspectSelectedParagraphTabStops

Using document and application text resources

This section describes how to work with text resources like fonts and text services like spell checking and find-and-replace.

IDocumentTextResources provides access to the text resources of a document, such as fonts, styles, and text-related services like spell checking and find-and-replace.

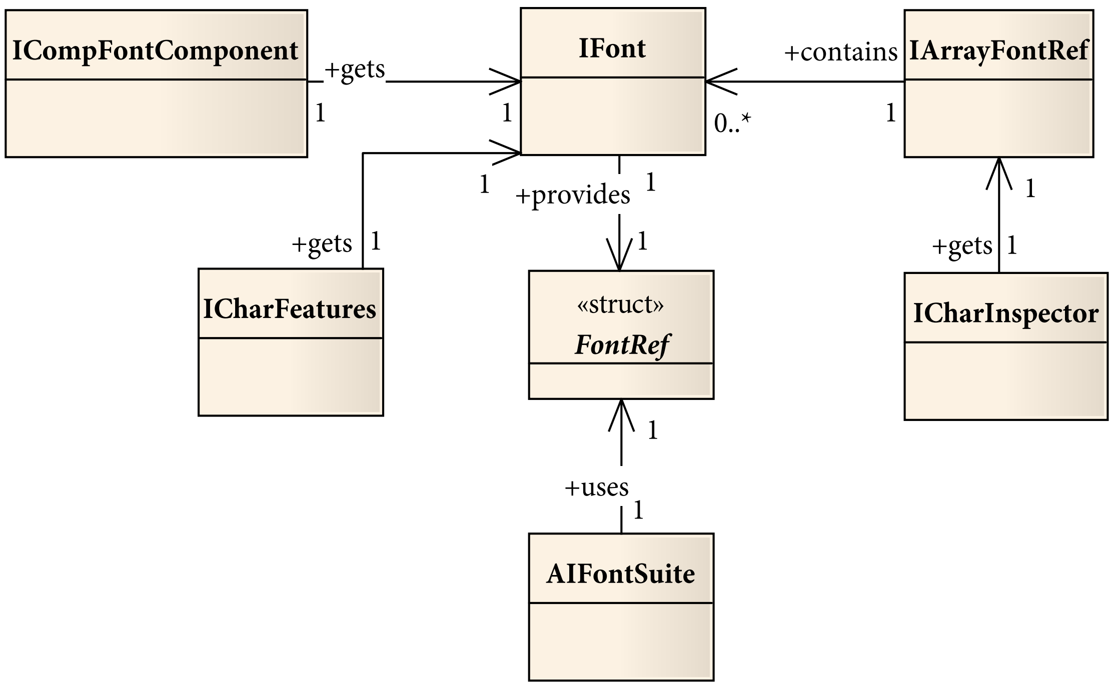
IApplicationTextResources provides access to the application's Asian text resources such as IMojiKumiSet, IKinsokuSet, and ICompFontSet.

Iterating through fonts

This use case shows how to iterate through each font currently in use in the current documents' text items. See Figure 17.

NOTE: Use AIFontSuite::CountFonts and AIFontSuite::IndexFontList to iterate through all fonts available in the document.

FIGURE 17 *IFont context*



Procedure

1. Find the first text frame in the document, using the instructions in "Accessing text using the artwork tree" on page 11.
2. Use the ITextFrame object to get the IStories set for the document, by first accessing the IStory for the text frame, and then getting the IStories container for the IStory object.
3. Get the ITextRanges object from the IStories object, using IStories::GetTextRanges.
4. Get the ICharInspector for the ITextRanges, to gain access to all the character features used in the document.
5. Get the IArrayFontRef container, using ICharInspector::GetFont.
6. Iterate through the IArrayFontRef container. For each FontRef, get the associated AIFont-Key using AIFontSuite::FontKeyFromFont and the FontRef.

API Reference

- AIFontSuite
- IArrayFontRef
- ICharInspector
- IStories

- IStory
- ITextFrame
- ITextRanges

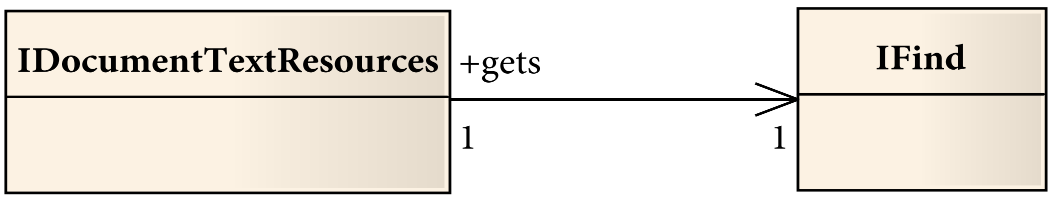
Sample code

- SnpText::IterateAllFonts
- SnpText::IterateUsedFonts

Finding and replacing text

The find-and-replace Adobe text engine feature, represented by the IFind object, allows you to search through text items for specific text strings and, if desired, replace each occurrence with another text string. See Figure 18.

Figure 18 ***IFind context***



Procedure

1. Find the paragraph to inspect. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the current document's text-resources set, using AIDocumentSuite::GetDocumentTextResources to get the DocumentTextResourcesRef, and then create a new IDocumentTextResources object using the DocumentTextResourcesRef.
3. Create a new IFind object, using IDocumentTextResources::GetFind.
4. Set the search string, using IFind::SetSearchChars.
5. Set the replace string, using IFind::SetReplaceChars.
6. Save the current and start positions of the text range, using IFind::GetPreReplaceAllSettings.
7. Set the text range to search, using IFind::SetSearchRange.

8. Loop through the text range, searching and replacing with the specified strings, using IFind::FindMatch and IFind::ReplaceMatch.
9. Restore the current and start positions of the text range, using IFind::RestorePreReplaceAllSettings.

API Reference

- AIDocumentSuite
- IDocumentTextResources
- IFind

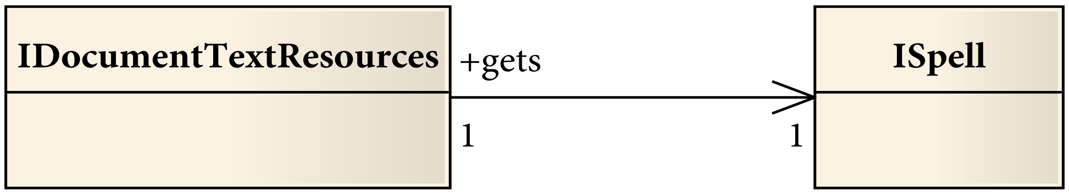
Sample code

SnpText::FindAndReplace

Checking spelling

The Adobe text engine's spell checker, represented by the ISpell object, allows you to configure and perform spell checks on text items in an Illustrator document. It supports 46 languages and language variants. See Figure 19.

FIGURE 19 ***ISpell context***



Procedure

1. Find the paragraph to inspect. You can do this either via the current selection using AIDocumentSuite::GetTextSelection (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the folder to the Illustrator dictionary folder, using AIFolderSuite::FindFolder and passing in kAIDictionariesFolderType.
3. Define a new SpellRef, using AITextUtilSuite::GetSpellFile and passing in the ai::FilePath to the Illustrator dictionary folder.
4. Create a new ISpell object from the SpellRef.

5. Loop through the text range, searching for unknown words, using ISpell::FindOneMisspelledWord.
6. Get the Illustrator dictionary's list of alternate suggestions, using ISpell::GetWordListContents.

API Reference

- ai::FilePath
- AIDocumentSuite
- AIFolderSuite
- AITextUtilSuite
- ISpell

Sample code

SnpText::RunSpellCheck

Porting from the legacy text API

This section shows a mapping between the legacy text API and the new text API.

The following suites became obsolete in Illustrator CS:

- AITextSuite
- AITextFaceStyleSuite
- AITextLineSuite
- AITextPathSuite
- AITextRunSuite
- AITextStreamSuite
- AITextStyleSuite

If your plug-in uses any of the above suites, you need to update your plug-in to work with the new text API in Illustrator CS and later. Table 1 maps the legacy text API members to the new text APIs.

TABLE 1 Mapping from legacy text API to new text API

Legacy text API	New text API
AIArtSuite::NewArt	AITextFrameSuite::NewPointText
AIArtSuite::NewArt	AITextFrameSuite::NewInPathText
AIArtSuite::NewArt	AITextFrameSuite::NewOnPathText
AIArtSuite::NewArt	AITextFrameSuite::NewOnPathText2
AITextFaceStyleSuite	AIFontSuite
AITextLineSuite	ATE::ITextLine
AITextLineSuite::GetFirstTextLine	ATE::ITextLinesIterator::Item
AITextLineSuite::GetNextTextLine	ATE::ITextLineIterator::Next
AITextLineSuite::GetTextLineOrientation	AITextFrameSuite::GetOrientation
AITextPathSuite	AITextFrameSuite
AITextPathSuite::GetTextPathMatrix	ATE::ITextFrame::GetMatrix
AITextPathSuite::GetTextPathOrientation	AITextFrameSuite::GetOrientation
AITextPathSuite::SetTextPathOrientation	AITextFrameSuite::SetOrientation
AITextRunSuite	ATE::ITextRange
AITextRunSuite::GetTextRunCharStyle	ATE::ICharStyle
AITextRunSuite::GetTextRunParaStyle	ATE::IParaStyle
AITextRunSuite::SetTextRunCharStyle	ATE::ICharStyle
AITextRunSuite::SetTextRunParaStyle	ATE::IParaStyle
AITextStreamSuite	ATE::ITextRange
AITextStreamSuite::GetCharStyleStream	ATE::ICharStyle
AITextStreamSuite::GetParaStyleStream	ATE::IParaStyle
AITextStreamSuite::GetSelectionStream	AITextFrameSuite::GetATETextSelection
AITextStreamSuite::GetSelectionStreamChar	AITextFrameSuite::GetATETextSelection
AITextStreamSuite::SetCharStyleStream	ATE::ICharStyle
AITextStreamSuite::SetParaStyleStream	ATE::IParaStyle
AITextStyleSuite::GetCurrentCharStyle	ATE::ICharStyle
AITextStyleSuite::GetCurrentFaceStyle	AIFontSuite
AITextStyleSuite::GetCurrentParaStyle	ATE::IParaStyle

Legacy text API	New text API
AITextStyleSuite::GetDefaultCharStyle	ATE::ICharStyle
AITextStyleSuite::GetDefaultParaStyle	ATE::IParaStyle
AITextStyleSuite::SetCurrentCharStyle	ATE::ICharStyle
AITextStyleSuite::SetCurrentFaceStyle	AIFontSuite
AITextStyleSuite::SetCurrentParaStyle	ATE::IParaStyle
AITextSuite::DoTextHit	AITextFrameSuite::DoTextFrameHit
AITextSuite::GetTextSelection	AITextFrameSuite::GetATETextSelection
AITextSuite::GetTextSelectionChar	AITextFrameSuite::GetATETextSelection
AITextSuite::GetTextType	AITextFrameSuite::GetType
AITextSuite::SetTextType	(Set at creation time.)