ADOBE® ILLUSTRATOR®

USING THE ADOBE TEXT ENGINE

Adobe

Using the Adobe Text Engine with Illustrator 2021

Technical Note #10502

1 About Adobe Text Engine

This document describes how to use the Adobe text engine—the text API provided by the Adobe® Illustrator® 2021 SDK—in your Illustrator plug-ins. It describes text-related use cases that solve typical programming problems such as inserting, deleting, and styling text.

The Adobe Text Engine (ATE) is the library that provides support for text to Adobe Illustrator. ATE was introduced in Illustrator CS (Illustrator 11.0) to replace the suites that provided text support in earlier versions of the product. The API described here is specific to Adobe Illustrator, and supports these major features:

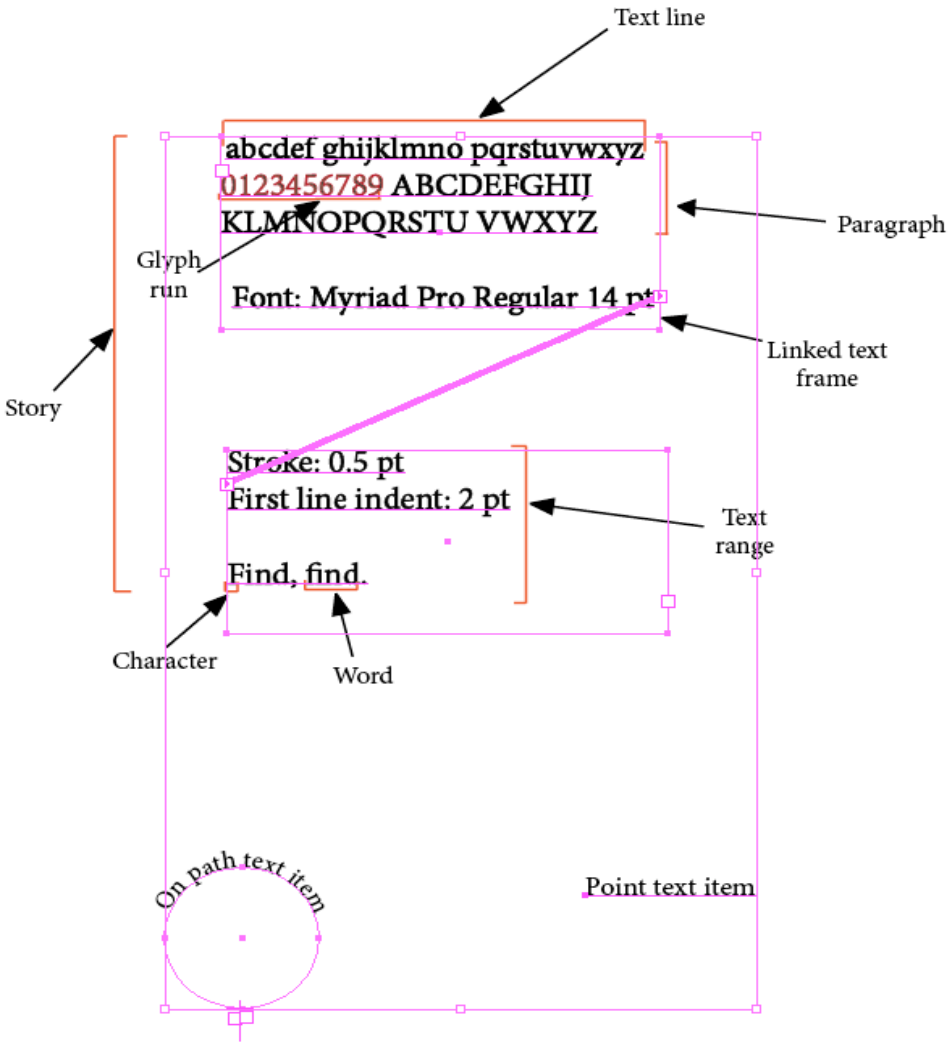
- Unicode.
- OpenType.
- Advanced typography like optical kerning, optical margin alignment, automatic glyph replacement, and glyph scaling.
- Character and paragraph styles.
- Asian text features like MojiKumi, Kinsoku, and composite fonts.

Terminology

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 CS5 or earlier.
SDK	The software development kit for Illustrator 2020. **NOTE:** In paths, *<SDK>* indicates your locally installed SDK root folder. The actual root folder depends on the installation and operating system.
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 2017, using the C++ environment.



NOTE: Figures in this document that describe ATE components and their relationships generally figures conform to a basic UML class-diagram specification; for readability. they have been limited to show only class names, associations, cardinality and a one-word description of the association. For more information on UML see http://www.uml.org/.

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

You plug-in code should use the C++ helper classes declared in the header file `<SDK>/illustratorapi/ate/ITexh.h` to work with text programatically. All C++ helper classes provided by `IText.h` begin with the letter `I`:

```
ITextFrame
ITextLine
ITextRange
ICharFeatures
ICharInspector
ICharStyle
IParaFeatures
IParaInspector
IParaStyle
```

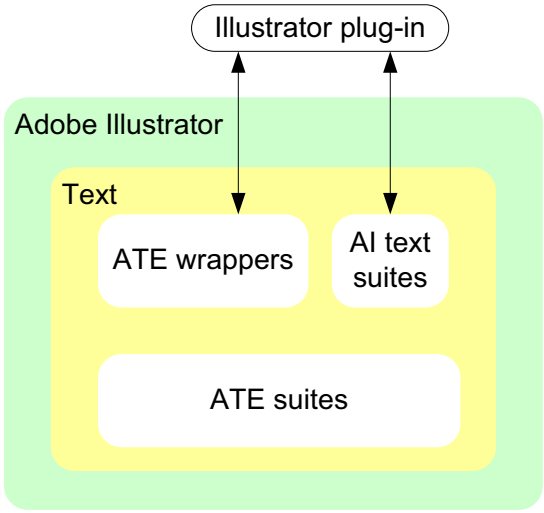
These are some key wrapper classes within the Adobe text engine API:

- `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, your plug-in code should not call these suites directly; 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.

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.



Using API documentation

The Adobe text engine wrappers and Illustrator text suites are documented in the API Reference, which is provided with the SDK. See *Getting Started with Adobe Illustrator 2021 Development* for details of how to access and use this documentation; it is available both as a searchable, compiled help file, and as straight, browsable HTML.

In the step-by-step instructions for various text operations, this manual lists the API suites and classes of interest, as well as the particular code samples that illustrate the operation.

2 Getting started with the text API in your plug-in

This chapter will help you get started with samples. It explains how to configure your project to use the Adobe text engine, and how to perform basic text operations with the API.

Exploring text with SDK samples

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.

To run snippet samples:

1. Run Illustrator 2021 with the SnippetRunner plug-in loaded. For instructions on loading an Illustrator plug-in, see *Getting Started with Adobe Illustrator 2020 Development*.
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 these 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.

These non-snippet samples in the Illustrator 2020 SDK also use the text API:

- `TextFileFormat`
- `MarkedObjects`

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 to add the required source code:

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

2. Right-click `IText.cpp` in the Solution Explorer and choose Properties.
3. Under C/C++ > Precompiled Headers, set Create/Use Precompiled Header to Not Using Precompiled Headers.
4. Repeat with `<SDK>/illustratorapi/ate/IThrowException.cpp`.

- In Xcode, 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`.)

   If you use a structure to pass in all the suite names, versions, and pointers, it must 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.

- 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.

Refer to the API reference documentation for `ATE::Exception`, and examine the sample code in `SnpTextException::ThrowATEException`.

Accessing text

This section describes the two basic methods for accessing text in a document; from the current selection, or from the artwork tree.

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 sample is taken from `SnpTextIterator::IterateSelectedTextFrames`. It 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`.

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

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

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

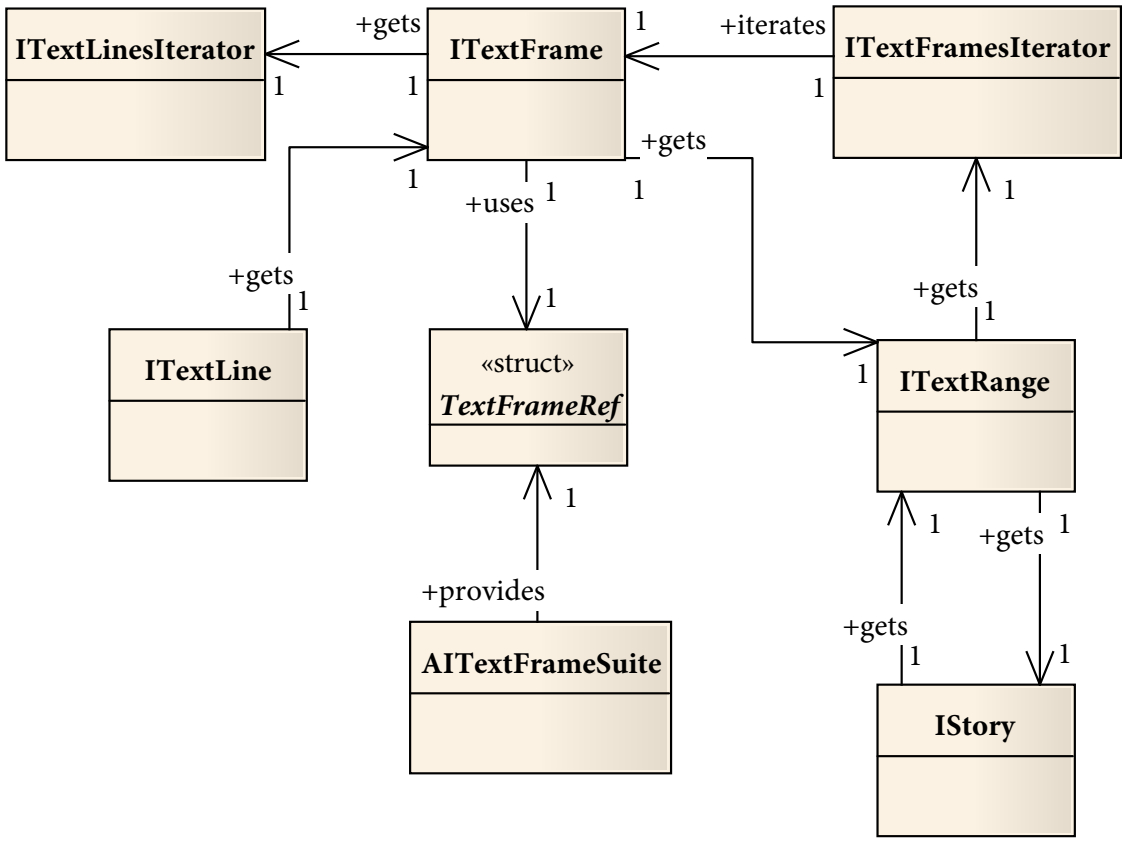
   ```
   AIArtHandle textFrameArt = textFrameArtSet[artIndex];
   TextFrameRef textFrameRef = NULL;
   ASErr result = sAITextFrame->GetATETextFrame(textFrameArt, &textFrameRef);
   aisdk::check_ai_error(result);
   ITextFrame textFrame(textFrameRef);
   ITextRange textRange = textFrame.GetTextRange();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

3 Iterating through text

This chapter 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. (Text that overruns the text frame is not included in the contents of the text frame's text range.)



To work iterate through text frames:

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. Alternatively, 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

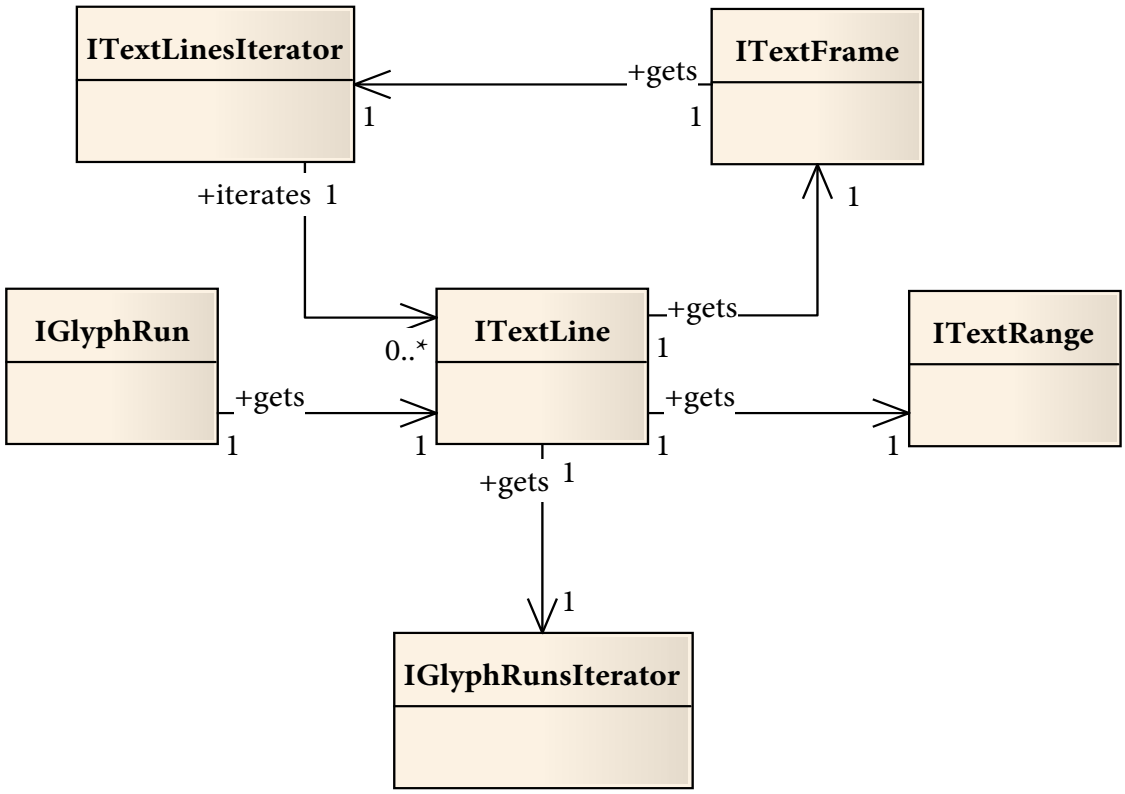
`AIDocumentSuite`
`AIMatchingArtSuite`
`ITextFrame`
`AITextFrameSuite`
`ITextRange`

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.



To iterate through lines:

1. Get the text frame or frames; see "Iterating through text frames" on page 10.

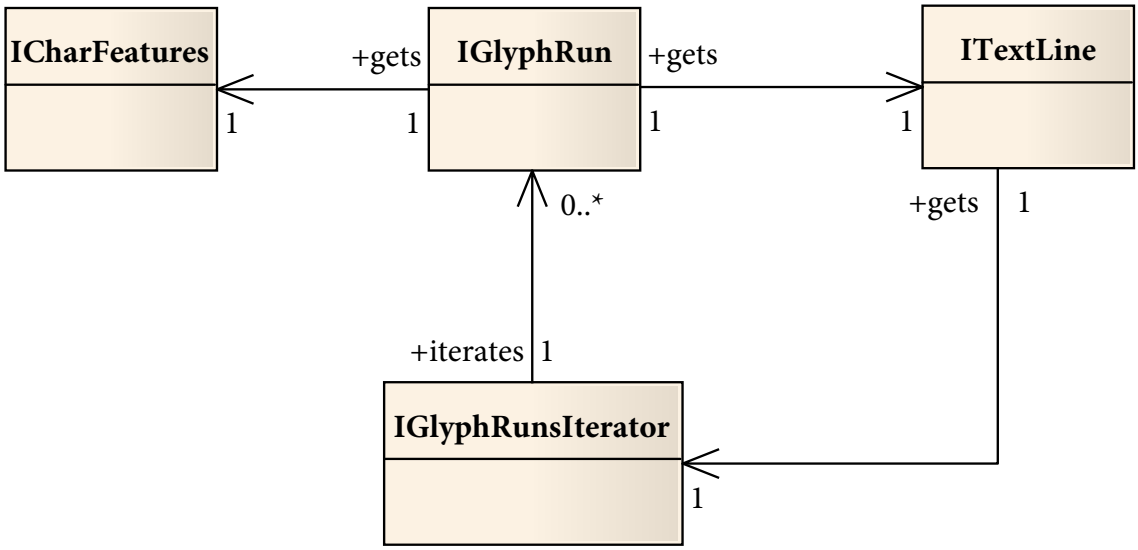
2. For each text frame, get the `ITextLinesIterator` using `ITextFrame::GetTextLinesIterator`, and use this object to iterate the text lines.
3. Get the text range in the line, using `ITextLine::GetTextRange`.
4. Get the string contents, using `ITextRange::GetContents`.

API Reference
`ITextFrame`
`ITextFrames`
`ITextFramesIterator`
`ITextLine`
`ITextLinesIterator`
`ITextRange`

Sample code
`SnpTextIterator::IterateGlyphRuns`
`SnpTextIterator::IterateLinesInSelectedFrames`

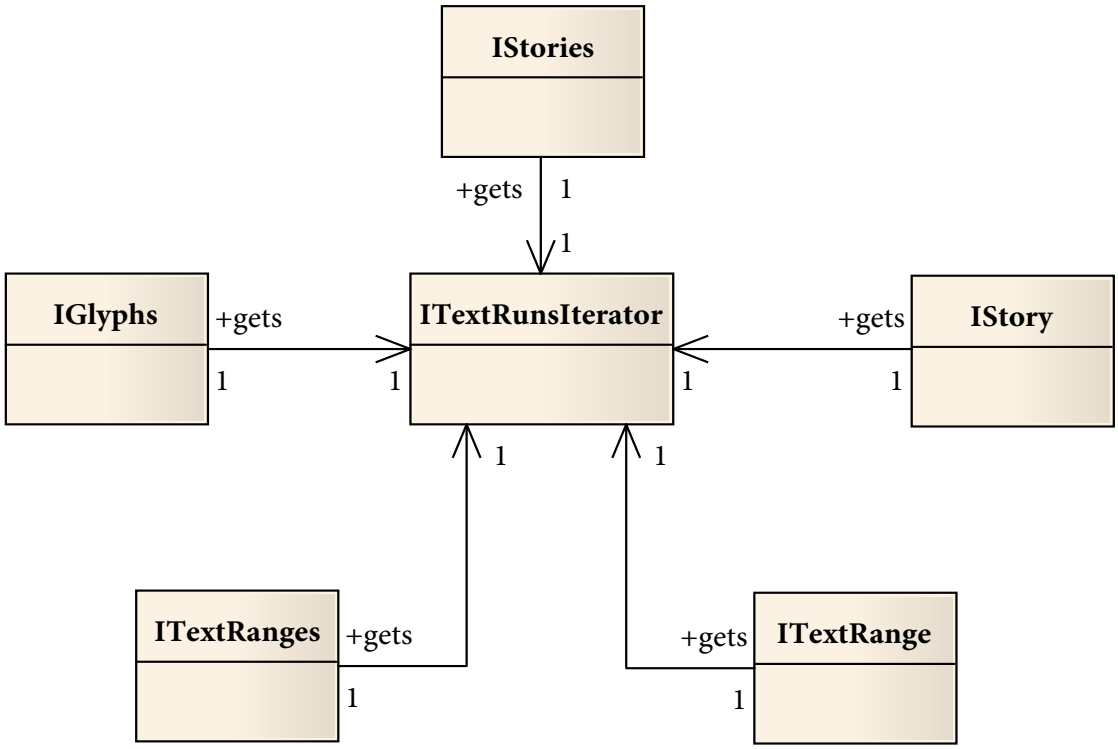
Glyph runs and text runs

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



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.

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 that can be accessed via `ITextRanges`, `ITextRange`, `IStory`, `IStories`, and `IGlyphs` objects, which provides access to each text run in the containing object.



Iterating through glyph runs

To iterate through glyph runs:

1. Get the text lines; see "Iterating through lines" on page 11.
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

To iterate through text runs:

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 iterator object, using `ITextRanges::GetTextRunsIterator`; or, in the case of a line containing a mixture of right-to-left (RTL) and left-to-right (LTR) text, use `ITextRanges::GetVisualGlyphRunsIterator`.
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

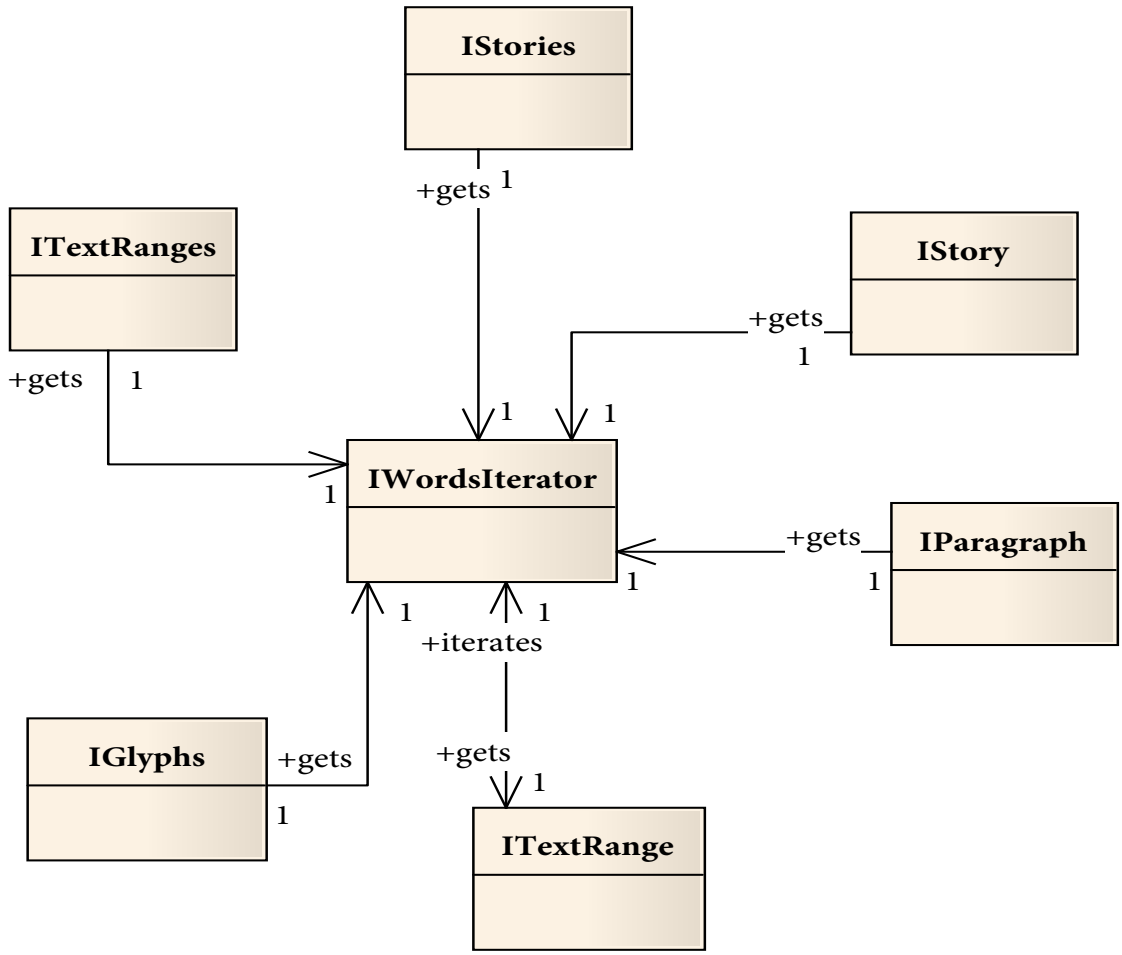
`AIDocumentSuite`
`ITextRange`
`ITextRanges`
`ITextRunsIterator`

Sample code

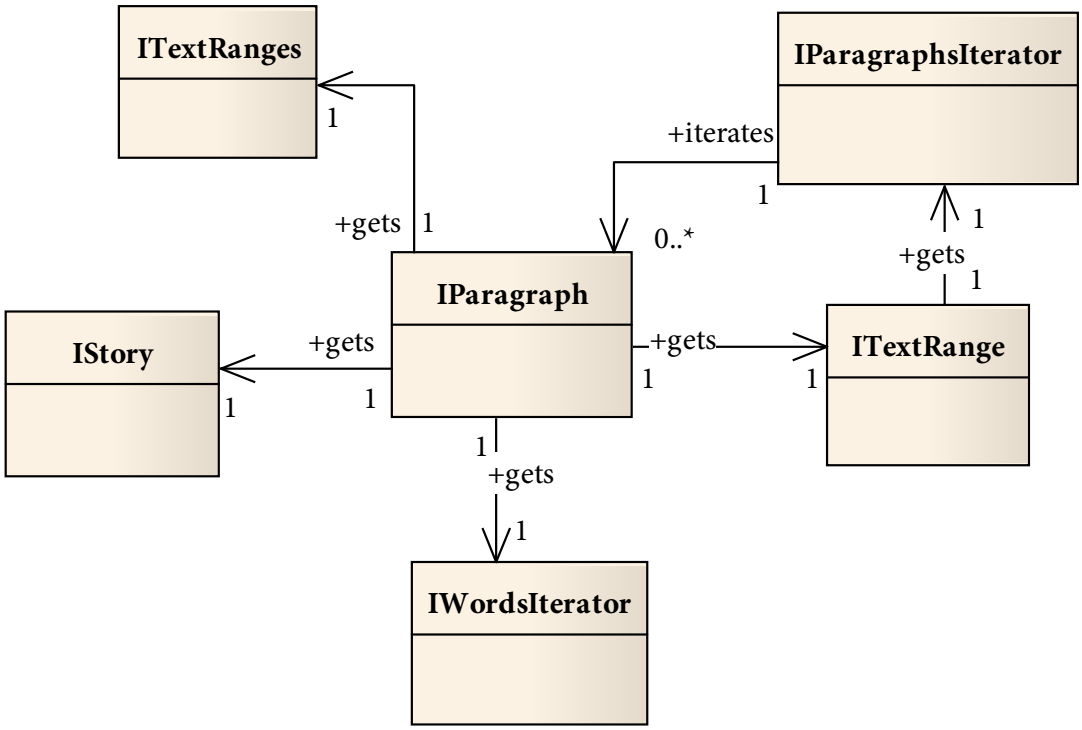
`SnpTextIterator::IterateTextRuns`

Characters, words, and paragraphs

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



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`.



Iterating through characters

To iterate through characters:

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

To iterate through words:

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

To iterate through paragraphs:

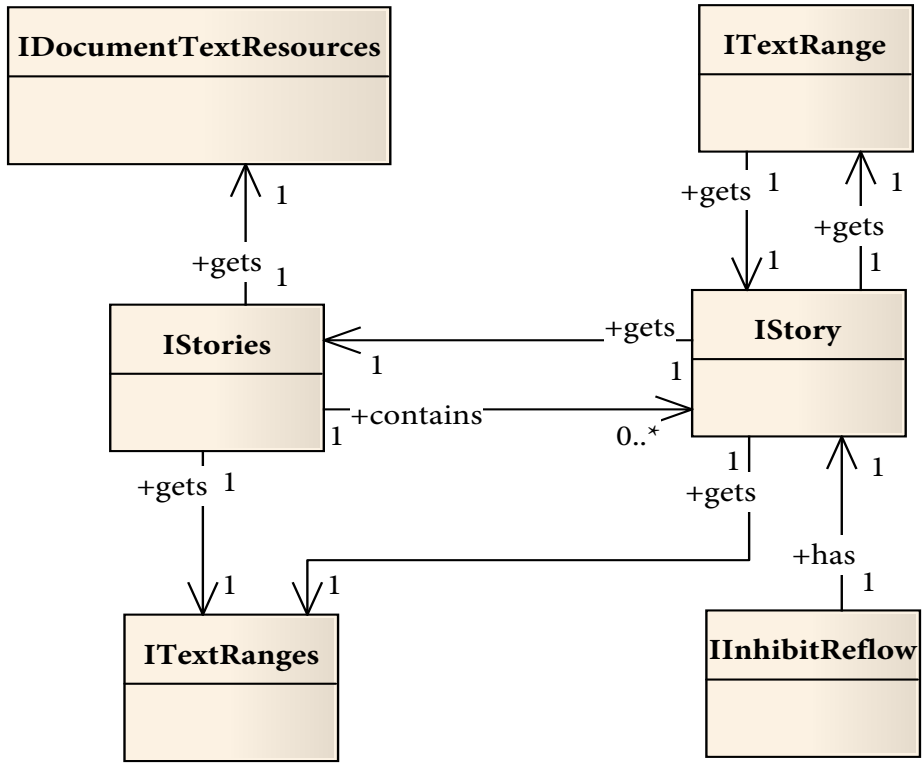
1. Find the text range containing the paragraphs to iterate. You can do this either via the current selection using `AIDocumentSuite::GetTextSelection` (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get an `IParagraphsIterator` object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



To iterate through stories:

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

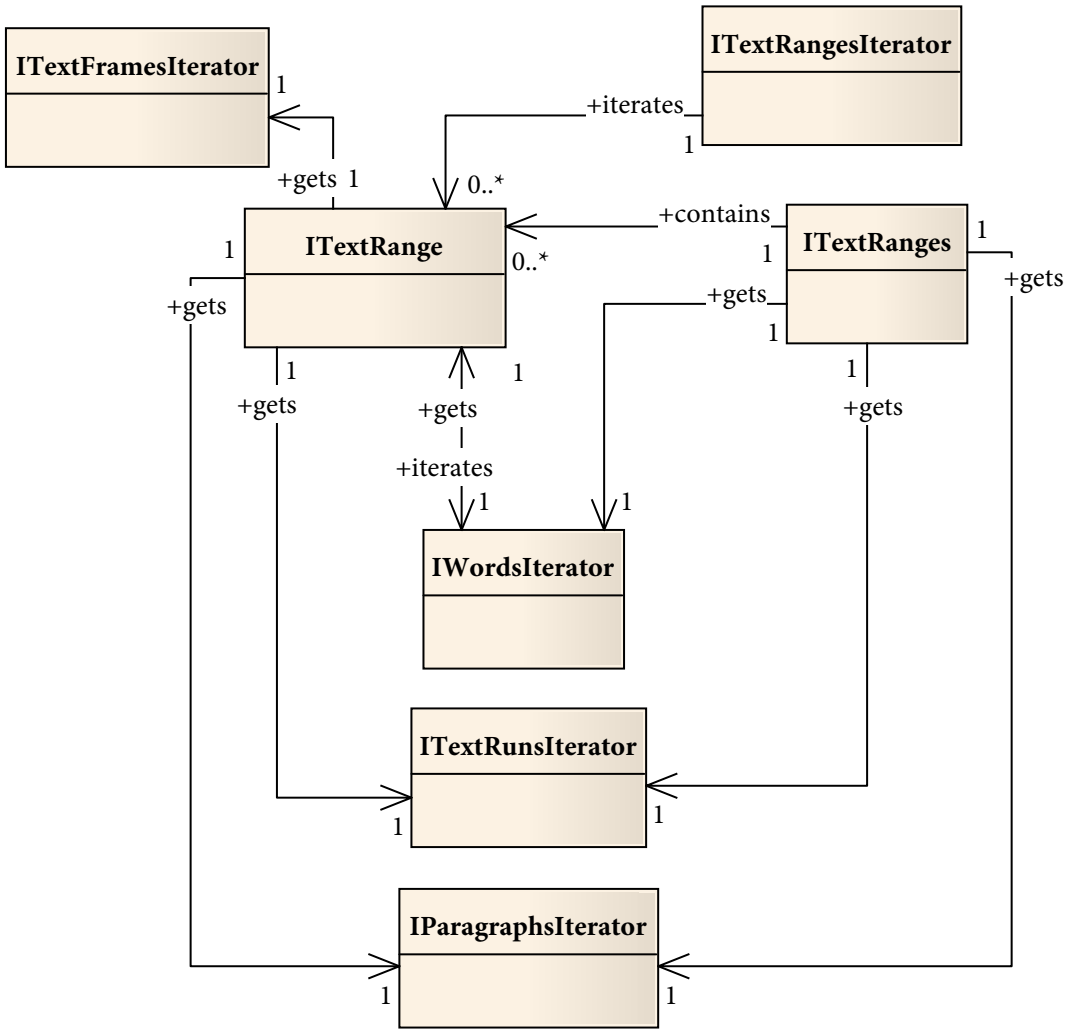
```
AIDocumentSuite
IStory
ITextRange
ITextRanges
```

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`. For a discussion of text-range-related use cases see "Creating text" on page 21.



To iterate through text ranges:

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

Kern types are managed at the story level. See "Setting kern type" on page 27.

To iterate through kern types

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. 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
```

4 Manipulating text

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

Creating text

There are various kinds of text. The following section describe how to create:

- Point text
- In-path text
- On-path text
- Threaded in-path text

Point text

To create a new point-text item in a document:

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`

In-path text

TO create a new in-path text item in a document:

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`

On-path text

T create a new on-path text item in a document:

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`

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.

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 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::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`.
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::NewInPathText`. 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.

To create threaded on-path text, follow the step for On-path text, but replace all occurrences of `NewInPathText` with `NewOnPathText`, and add the extra parameters for the start and end segments.

API Reference
AIArtSuite
AITextFrameSuite
ITextRange

Sample code
SnpText::ArtHandleFromRect
SnpText::CreateThreadedInPathText

Selecting text

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

1. Find the text range.
   - ▷ To highlight text in the currently selected text frame, follow the instructions in "Accessing text using selection" on page 9 to find the selected text range.
   - ▷ 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 9 to find a text range.
2. 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

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.

You can programmatically set or remove text focus. Text focus is set through the AIDocumentSuite at the story level.

Setting text focus

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

Removing text focus

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

To insert a range of text into a selected text range:

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 and moving text

You can copy or move text to a new text item, within story bounds, or into another story.

Copy

To copy a selected range of text to a new text item:

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`.

Move

To move a selected range of text from one text item to a new text item:

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.
3. To move the text range into a new or existing story, follow the instructions for Copy, 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`
`SnpText::CopyText`

Replacing and deleting text

You can programmatically remove text, or remove it and replace it with different text

Delete

To delete the selected text range:

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`.

Replace

To replace the selected text range with a new text range:

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::DeleteTextRange`
`SnpText::ReplaceText`

Linking text frames

When you create a link between selected text frames, their contents become a single story. When you remove links between selected text frames, you split the text frames' contents into separate stories.

Linking

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

Unlinking

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
`AIArtSetSuite`
`AIArtSpec`
`AITextFrameSuite`

Sample code
`SnpText::LinkTextFrames`
`SnpText::UnlinkTextFrames`

Deleting text frames

To delete a text-frame art item from a document:

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. A text frame is also an art object, so you can delete the object using the `AIArtSuite`. If you are working with an art object, delete it using `AIArtSuite::DisposeArt`.

API Reference
`AIArtSuite`
`AIDocumentSuite`
`AITextFrameSuite`
`ITextFrame`

Sample code
`SnpText::LinkTextFrames`
`SnpText::UnlinkTextFrames`
`SnpTextIterator::IterateSelectedTextFrames`

Converting legacy text

To convert legacy text in a document to work with the ATE API:

1. Convert all legacy text in the current document using `AILegacyTextConversionSuite::ConvertAllToNative`.
2. 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. These types of kerning are defined:

- *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.
- *kMetricRomanOnlyKern* — This should be the default kern type for Asian text. It provides kerning to Roman text without affecting any surrounding Asian text.
- *kOpticalKern* — Uses the glyphs' shapes to kern the characters as they appear to the eye.

Kern types are managed at the story level. To set the kern type of a range of text:

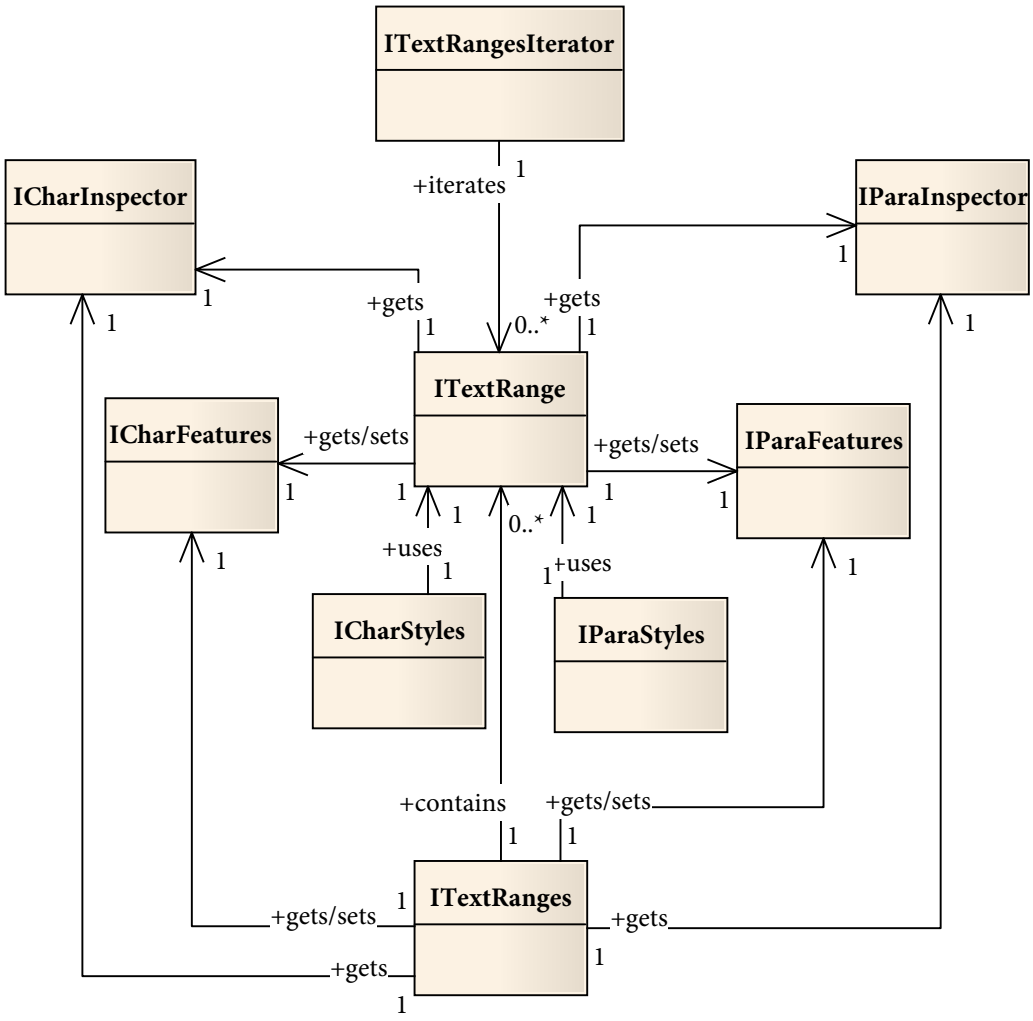
1. Find the text range to edit. You can do this either via the current selection using `AIDocumentSuite::GetTextSelection` (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the story using `ITextRange::GetStory`.
3. Set the kern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

API Reference
`AIDocumentSuite`
`IStory`
`ITextRange`

Sample code `SnpTextStyler::SetKernType`

5 Styling text

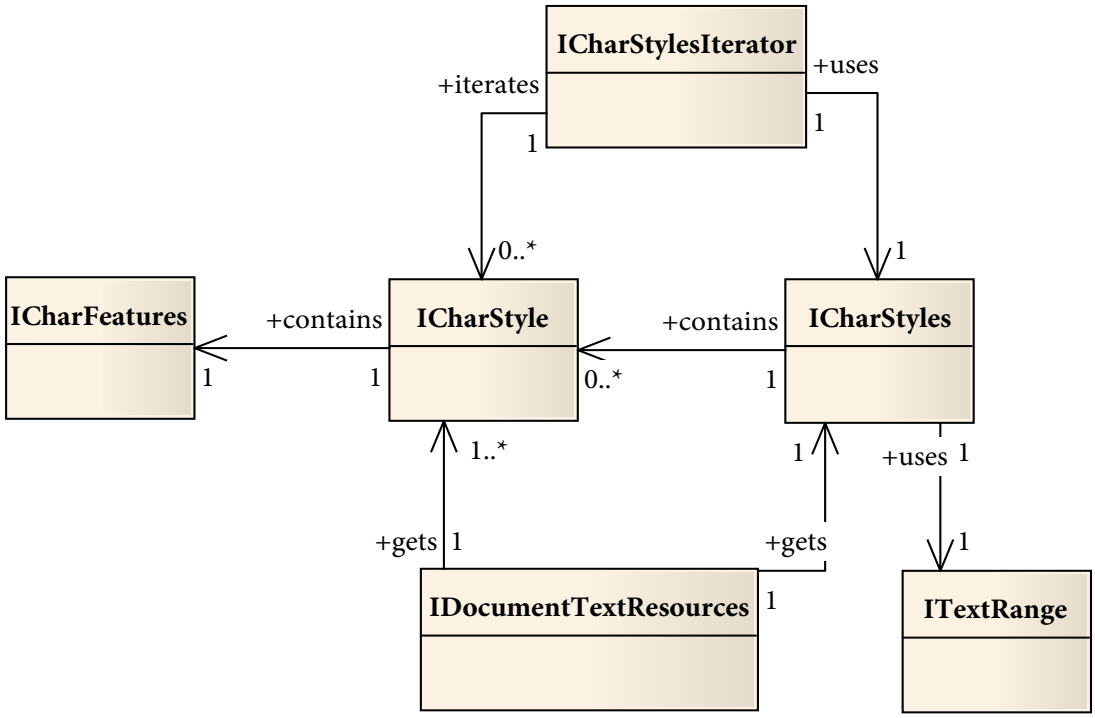
This chapter describes how to examine, create, update, and delete character and paragraph styles and how to style text.



Character styles

The character styles associated with a document are contained in an `ICharStyles` set that contains zero or more `ICharStyle` 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`.



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.

API Reference

```
AIDocumentSuite
ICharStyle
ICharStyles
ICharStylesIterator
IDocumentTextResources
ITextRange
AIATECurrentTextFeaturesSuite
```

Sample code

```
SnpTextStyles::IterateCharacterStyles
SnpTextStyles::CreateCharacterStyle
SnpTextStyles::GetCurrentCharacterStyle
SnpTextStyles::DeleteCharacterStyle
SnpTextStyles::ApplyCharacterStyle
SnpTextStyles::ClearCharacterStyle
```

Iterating through character styles

1. Get the current document text resources set. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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`.

Creating a character style

To add a new named character style to a document's text resources:

1. Get the current document text resources set. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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.

Getting the current character style

To find the character style currently in use in a document:

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`.

Deleting a character style

To delete a named character style from a document's text resources:

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

Applying a character style

To apply a named character style to a range of text:

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. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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`.

Clearing a character style

To clear a named character style from a range of text:

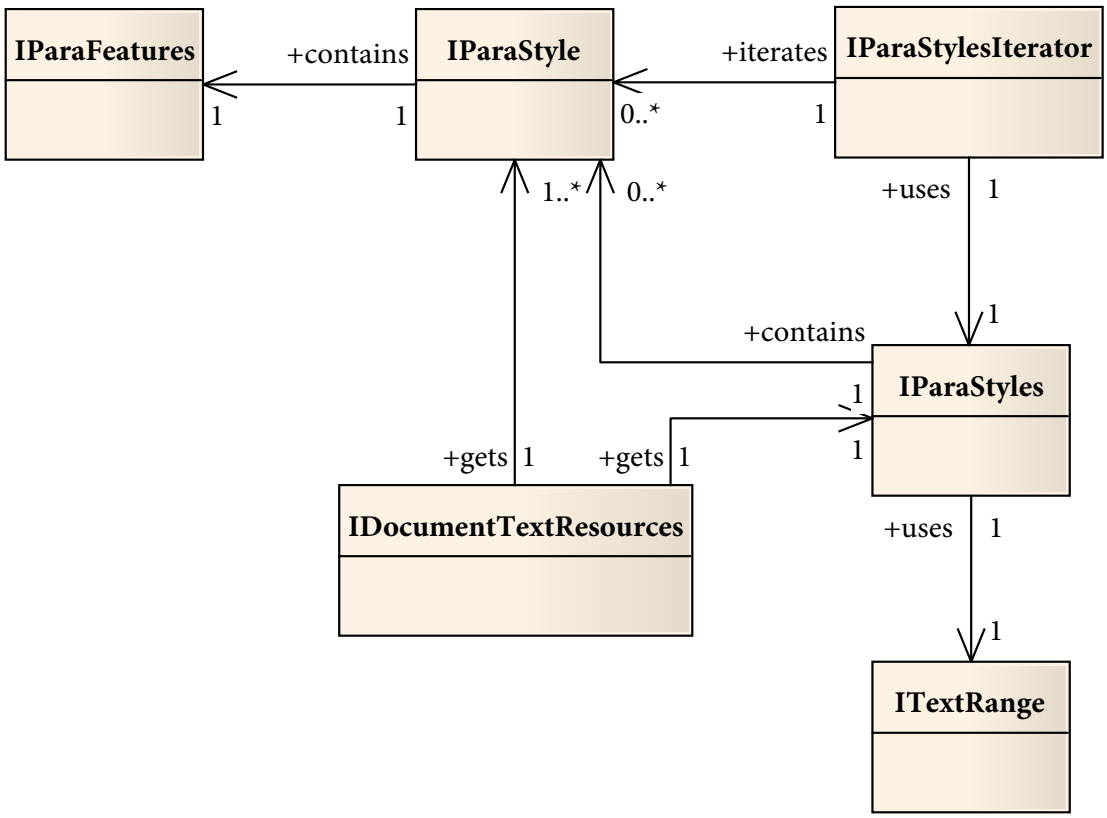
1. Find the text range from which to clear the character style. You can do this either via the current selection using `AIDocumentSuite::GetTextSelection` (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Clear the character style from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

   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`.

Paragraph styles

The paragraph styles associated with a document are contained in an `IParaStyles` set, which contains zero or more `IParaStyle` 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`.



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.

API Reference

```
AIDocumentSuite
IDocumentTextResources
AIATECurrentTextFeaturesSuite
IParaFeatures
IParaStyle
IParaStyles
IParaStylesIterator
ITextRange
```

Sample code

```
SnpTextStyles::IterateParagraphStyles
SnpTextStyles::CreateParagraphStyle
SnpTextStyles::GetCurrentParagraphStyle
SnpTextStyles::DeleteParagraphStyle
SnpTextStyles::ApplyParagraphStyle
SnpTextStyles::ClearParagraphStyle
```

Iterating through paragraph styles

1. Get the current document text resources set. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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`.

Creating a paragraph style

To add a new named paragraph style to a document's text resources:

1. Get the current document text resources set. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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.

Getting current paragraph style

To find the paragraph style currently in use in the current document:

1. Get the `ParaStyleRef` to the current style applied to new text items, using `AIATECurrentTextFeaturesSuite::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`.

Deleting a paragraph style

To delete a named paragraph style from a document's text resources:

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

Applying a paragraph style

To can apply a named paragraph style to a range of text:

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. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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`.

Clearing a paragraph style

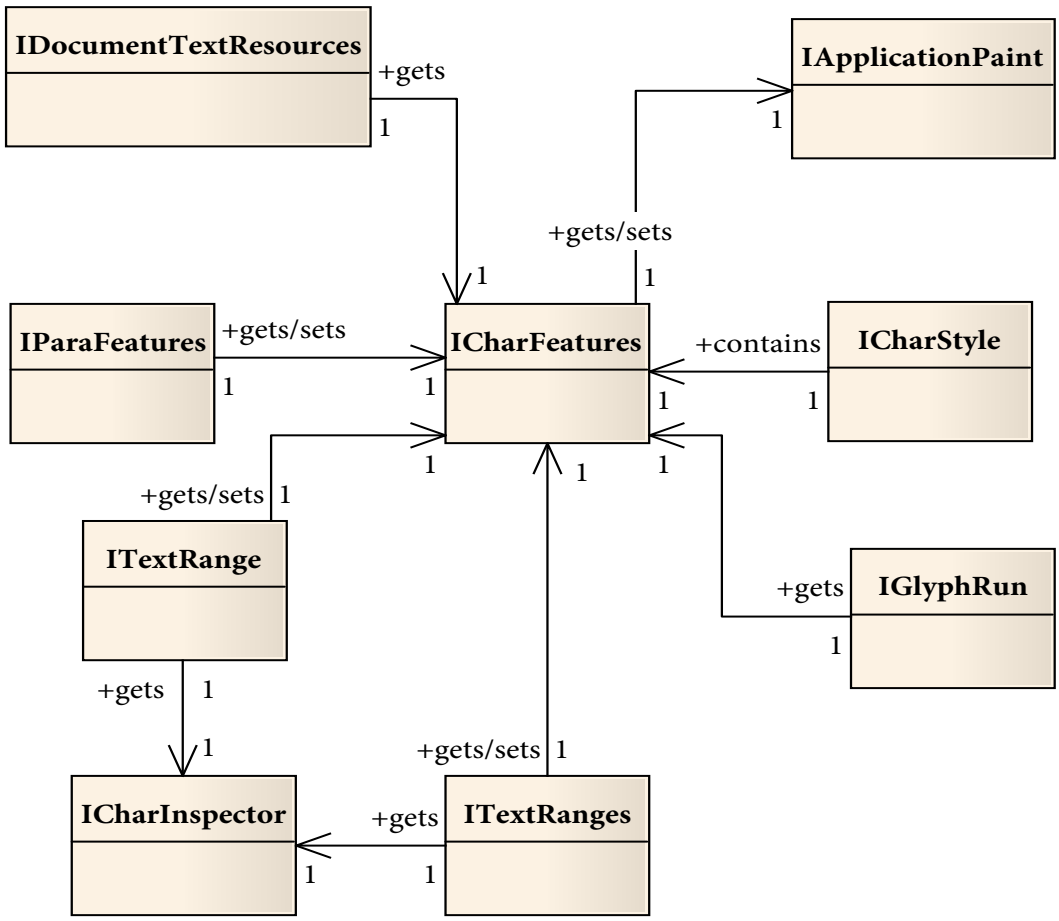
To clear a named paragraph style from a range of text:

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`.

   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`.

Working with 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.



API Reference

```
AIDocumentSuite
ICharFeatures
ICharInspector
ICharStyle
ITextRange
ITextRanges
AIATECurrentTextFeaturesSuite
```

Sample code

```
SnpTextStyler::GetCharacterFeatures
SnpTextStyler::InspectSelectedCharacterFeatures
SnpTextStyler::ApplyLocalCharacterFeatures
SnpTextStyler::ClearLocalCharacterFeatures
SnpTextStyler::SetCurrentCharacterOverrides
```

Getting character features

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).

   Alternatively, 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.

Applying character features

To apply a set of styling attributes to a range of characters:

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.

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

Clearing character features

To clear styling attributes from a range of characters:

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`.

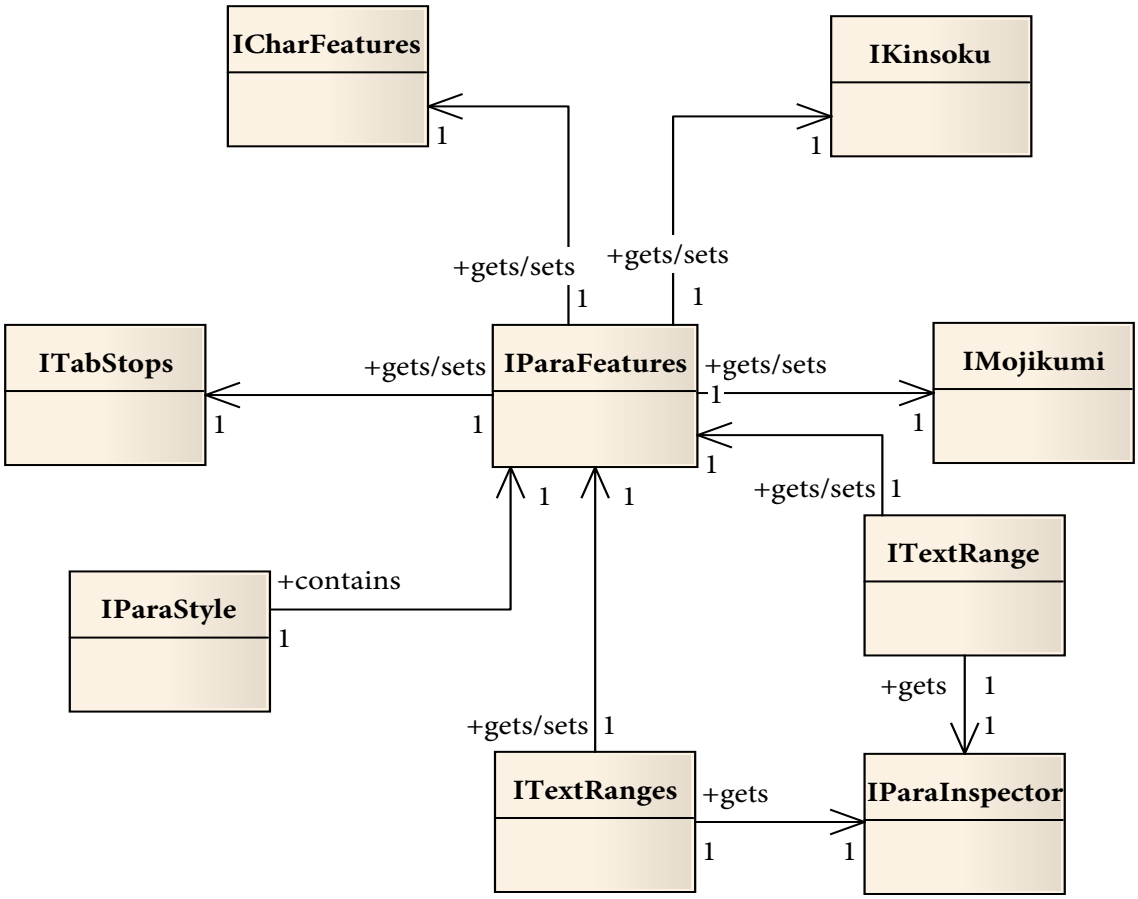
Setting current character overrides

To set the overriding styling attributes applied to new characters:

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 `AIATECurrentTextFeaturesSuite::SetCurrentCharOverrides`.

Working with 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.



API Reference

```
AIDocumentSuite
IParagraph
IParaFeatures
IParaInspector
IParaStyle
ITextRange
ITextRanges
AIATECurrentTextFeaturesSuite
```

Sample code

```
SnpTextStyler::InspectSelectedParagraphFeatures
SnpTextStyler::ApplyLocalParagraphFeatures
SnpTextStyler::ClearLocalParagraphFeatures
SnpTextStyler::SetCurrentParagraphOverrides
```

Getting paragraph features

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.

Applying paragraph features

To apply a set of styling attributes to a set of paragraphs:

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.

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

Clearing paragraph features

To clear a set of styling attributes from a set of paragraphs:

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`.

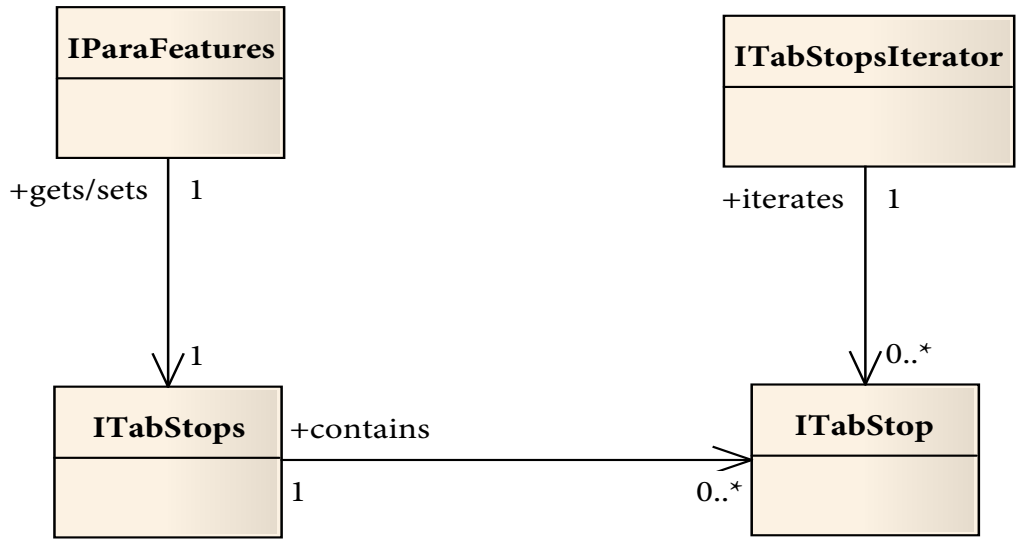
Setting current paragraph overrides

To set the overriding styling attributes applied to new paragraphs:

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`.

Working with tab stops

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`.



API Reference

`AIDocumentSuite`
`IArrayTabStopsRef`
`IParaFeatures`
`IParagraph`
`IParaInspector`
`ITabStop`
`ITabStops`
`ITextRange`
`ITextRanges`

Sample code

`SnpTextStyler::AddTabStops`
`SnpTextStyler::RemoveTabStops`
`SnpTextStyler::InspectSelectedParagraphTabStops`

Adding tab stops

To add tab stops to a paragraph or set of paragraphs:

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`.

Removing tab stops

To remove tab stops from a paragraph or set of paragraphs.

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`.

Inspecting tab stops

To find the tab stop in a paragraph or set of paragraphs:

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.

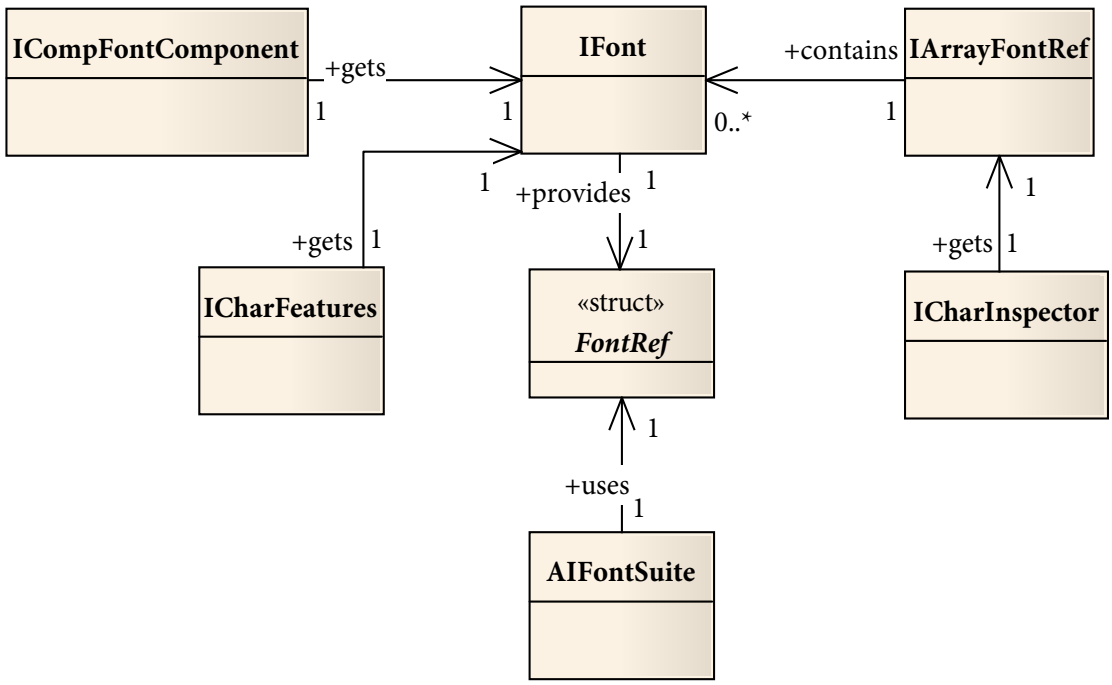
6 Using document and application text resources

This chapter describes how to work with text resources such as fonts, and text services such as 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

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



To iterate through each font currently in use in the current documents' text items:

1. Find the first text frame in the document, using the instructions in "Accessing text using the artwork tree" on page 9.
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`.

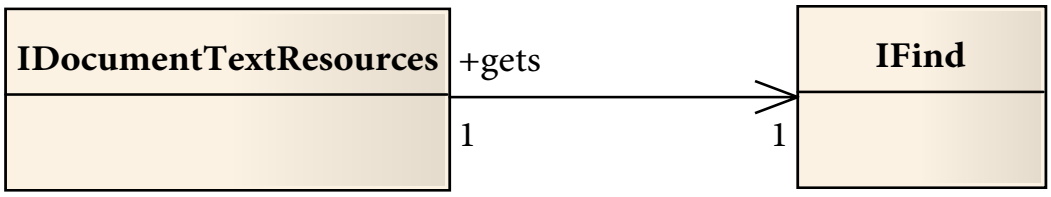
4. Get the `ICharInspector` for the `ITextRanges`, to gain access to all the character features used in the document.
5. Get the `IArrayFontRef` container, using `ICharInspector::GetFont`.
6. Iterate through the `IArrayFontRef` container. For each `FontRef`, get the associated `AIFontKey` using `AIFontSuite::FontKeyFromFont` 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.



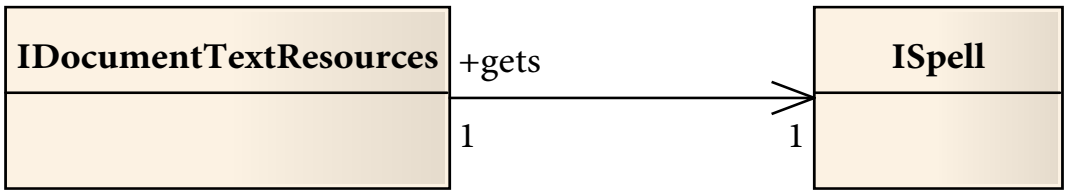
1. Find the paragraph to inspect. You can do this either via the current selection using `AIDocumentSuite::GetTextSelection` (which provides a reference to the text ranges selected in the current document) or by traversing the artwork tree.
2. Get the current document text resources set. Use `AIDocumentSuite::GetDocumentTextResources` to get the `DocumentTextResourcesRef`, and then use it to create a new `IDocumentTextResources` object.
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.



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
```

7 Porting to this Release

This chapter describes changes and new features in this release of the Adobe text engine.

New data types

In this release, we have improved calculation precision in all the mathematical operations, in order both to improve the quality of art and to make operations such as rotation, scaling, and dragging more accurate.

To do this, we have generally replaced the use of the C/C++ `float` data type with the `double` data type. In `AITypes.h`, the type of `AIReal` has been changed from `float` to `double`. Other data types such as `AIRealMatrix`, `AIRealPoint`, and`AIRealRect` have been similarly updated.

The data type `ASReal`, which was defined as a `float`, it has been completely removed from the API, in order to prevent confusion and possible type conflicts. This means that the signature of many functions has changed.

Data types in ATE have been replaced as follows:

Former type	Replaced by type
`ASInt32`	`ATETextDOM::Int32`
`ASUnicode*`	`ATETextDOM::Unicode*`

`ASReal`	`ATETextDOM::Real`
	—or—
	`ATETextDOM::Float`

`ASRealMatrix`	`ATETextDOM::RealMatrix`
	—or—
	`ATETextDOM::FloatMatrix`

`ASRealPoint`	`TETextDOM::FloatPoint*`
	—or—
	`ATETextDOM::RealPoint`

New helper functions

New functions have been added to convert between floating-point and real numbers:

```
void ATEFloatPointToATERealPoint ( const ATETextDOM::FloatPoint *f,
                                   ATETextDOM::RealPoint *r )

void ATEFloatMatrixToATERealMatrix( const ATETextDOM::FloatMatrix *f,
                                    ATETextDOM::RealMatrix *r )
```