Additions to the Apple Information Access Toolkit

Release notes for AIAT 1.1

2/19/98
Technical Publications

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

About This Document

This document describes additions to the Apple Information Access Toolkit (IAT) since version 1.0. These additions include new or modified methods in existing IAT classes, as well as new classes to do the following:

- Isolate a subset of the document for use as a document summary.
- Specify arbitrary document categories and route (sort) documents among them.

This document assumes that you have read the *Apple Information Access Toolkit v.1.0 Programmer's Guide* and are familiar with the various components of the IAT and the terminology used in data searching and information retrieval.

In addition, you should be familiar with C++ and object-oriented programming.

What's In This Document

This document is divided into three chapters:

- Chapter 1, "Document Summarization,"describes how to summarize a document using IAT classes.
- Chapter 2, "Document Routing," describes document routing and how to do so using IAT classes.
- Chapter 3, "Additions and Changes to Existing Classes," describes changes or additions to classes described in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

A glossary and index are provided at the end of the book as well as an appendix indicating this document's revision history.

Conventions Used in This Document

This document uses special conventions to present certain types of information. Words that require special treatment appear in specific fonts or font styles.

Special Fonts

All code listings, reserved words, command options, resource types, and the names of actual libraries are shown in Letter Gothic (`this is Letter Gothic`).

Command Syntax

This document uses the following syntax conventions:

`literal`	Letter Gothic text indicates a word that must appear exactly as shown.
italics	Italics indicate a parameter that you must replace with anything that matches the parameter's definition.

Types of Notes

This document uses two types of notes.

Note
A note like this contains information that is useful but that you do not have to read to understand the main text. ◆

IMPORTANT
A note like this contains information that is crucial to understanding the main text. ▲

CHAPTER 1

Document Summarization

Contents

You can use the IAT to provide a brief summary of a document. For example, when presenting a list of documents as the results of a search, you often want to include brief descriptions of each one so that the end user can easily determine which one to choose. This chapter describes the IAT classes and methods you use to summarize the contents of a given document.

Document Summaries

A **document summary** is a subset of the document that describes its contents. For example, if a document describes how to build a log cabin using an axe, a document summary could be a sentence that contains the words "log cabin," "axe" and "constructing," such as "Constructing a log cabin using only your hands and an axe can be a rewarding experience." You use the IAT to find the portions of the document that best typifies its contents.

To create a document summary, you must first break up the document into an arbitrary number of portions called **text extents**, or simply extents. An extent can be a sentence, or any other grouping of characters (for example, a paragraph, or even simply a block of 72 characters). Given all the available extents, the IAT then ranks them in the order that most closely represent the entire document. You can then use one or more of the most highly ranked extents to summarize the document.

To create a summary for a given document, you must do the following:

- Designate an extent parser to divide the document into extents. The type of extent you wish to use may depend on the type of document.
- Call the IAT document abstractor with the designated document to receive a ranked listing of extents.
- Choose one or more extents to use as the summary.

The following sections describe these steps in more detail.

Extent Parsers

An extent parser breaks up a document into a group of text extents. The IAT provides the abstract class `IAExtentParser` which contains functions you can use to create an extent parser. Your application can implement an extent parser by creating a subclass of `IAExtentParser` and overriding the `GettingNextEvent` method. If you want to parse by sentence, you can use the subclass `IAANSISentenceParser` included in the IAT instead of writing your own parser.

The Document Abstractor

The IAT class `IADocumentAbstractor` contains methods that let you parse a document into text extents and then rank them. The higher the ranking, the more closely the extent resembles the entire document. You can then display one or more of the highest-ranked extents to summarize the contents of the document.

Listing 1-1 shows a function that creates a list of ranked extents from a file.

Listing 1-1 Breaking up a document into ranked extents

```
void DemoAbstractor (StringPtr file) {
    FSSpec      mMacFileSpec;
    short       mDataForkRefNum;

// Data to use with the abstractor's Summarize method
    uint32 numberOfSentences = 1;
    TermIndex* contextIndex = NULL;
    clock_t progFrequency = 10000;
    void* callerData = NULL;
    RankedProgressFn* progressFn = NULL;

// Files used with the EnglishAnalyis object and the sentence parser
    char* stopwordFile = "EnglishStopwords";
    char* stemDictDoc = "EnglishSubstitutions";
```

```
    char* abbrevFile = "EnglishAbbreviations";

// Open the file and read into a buffer
    // Turn the filename into a Mac OS FSSpec
        OSErr iErr = FSMakeFSSpec(0, 0, file, &mMacFileSpec);

    // Open file
        OSErr err = FSpOpenDF(&mMacFileSpec, fsRdPerm, &mDataForkRefNum);
            if (err != noErr) {
                return;
            }

    // Read the file
        Handle dataHandle = nil;

        long FileLength;
        err = GetEOF(mDataForkRefNum, &fileLength);
        if (err != noErr) {
            return;
            }
        dataHandle = NewHandle(fileLength + 1);
        if (dataHandle == nil) return;

        err = SetFPos(mDataForkRefNum, fsFromStart, 0);
        if (err != noErr) {
            return;
            }
        HLock(dataHandle);

        err = FSRead(mDataForkRefNum, &fileLength, *dataHandle);
        if (err != noErr) {
            return;
            }

    // Get pointer to buffer and the buffer length
        *((*dataHandle)+fileLength) = '\0';
        char* buffer = (char*)(*dataHandle);
        uint32 bufferLength = strlen(buffer);


// Create the analysis object to use with the parser
```

```
    EnglishAnalysis* myAnalysis = new EnglishAnalysis(stopwordFile,
                                        stemDictDoc);

// Create the storage object
    IAStorage* myStorage = MakeHFSStorage(0,0,"\ptemp.index");
    IADeleteOnUnwind delStorage(myStorage);

// Designate the extent parser
    IAExtentParser* myParser = new IAANSISentenceParser((byte*)buffer,
                                    bufferLength, abbrevFile);

// Create the abstractor and get the extents
    IADocumentAbstractor MyAbstractor(myParser, myStorage, myAnalysis);
    myAbstractor.Summarize(progressFn, progFrequency, callerData,
        numberOfSentences, contextIndex);

// The DumpInformation function uses GetNumberOfSentences
// and GetSentences to get the top ranked sentence
    uint32 showThisManySentences = 1;
    DumpInformation(myAbstractor, showThisManySentences);

// Cleanup
    delete myParser;
    HUnlock (dataHandle);

// Close the file

    err = FSClose(mDataForkRefNum);

    if (err != noErr) {
        return;
    }
    FlushVol(nil, mMacFileSpec.vRefNum);
}
```

The `DemoAbstractor` function opens the HFS file and reads the contents into a buffer. It initializes the text parser (`myParser`) with the buffer and an abbreviations file. The abbreviations file `EnglishAbbreviations` holds abbreviated words (such as i.e., or e.g.) that would trigger end-of-sentence conditions if they were not called out.

After setting up the sentence parser, `DemoAbstractor` initializes the document abstractor (`myAbstractor`), specifying the extent parser, the storage medium, and the type of analysis to use. This abstractor uses the `EnglishAnalysis` subclass of `IAAnalysis` to analyze the text extents (sentences) and extract the relevant tokens. The filters defined in `EnglishAnalysis` require the text files `EnglishSubstitutions` and `EnglishStopwords` for proper operation.

After creating the abstractor, the `Summarize` function then breaks up the document into a ranked list of sentence extents. As with many other IAT components, you can specify a callback during the `Summarize` call to give time to your application, if desired. This example has no callback, however, so the `progressFn` pointer is set to `NULL`.

The `DemoAbstractor` function calls the `DumpInformation` function, shown in the next section, to display the highest-ranked sentences.

Presenting the Summary

After creating a ranked list of text extents, you can display one or more as the document summary. The example in Listing 1-1 calls the `DumpInformation` method to present the summary. Listing 1-2 shows a possible implementation for `DumpInformation` that cycles through the array of ranked sentences and displays one or more with the highest ranks.

Listing 1-2 Displaying the sentences with the highest rank.

```
void DumpInformation (const IADocumentAbstractor& abstractor,
        uint32 showlevel)
{
    uint32 paragraphNumber = 0;
    bool firstHasBeenShown = false;
    bool showScore = true;
    bool showRank = true;
    uint32 numberTopWords = 5;
    bool showSentences = true;

// Get the number of sentences and the array of pointers to those
// sentences
```

```
    uint32 cnt = abstractor.GetNumberOfExtents();
    IAExtentDoc** sentences = abstractor.GetExtents();

// Now loop and display the sentences or other information
    for(int i=0; i< cnt; i++) {

    // Keep looping until you've displayed the number of sentences
    // requested, or until you run out of sentences.
        if(sentences[i] && (sentences[i]->GetRank() < showlevel) &&
                (sentences[i]->GetLength() > 0)) {

        // Get the sentence from the pointer
            IAExtentDoc* doc = sentences[i];

        // The sentence parser also groups extents.
        // If an extent begins a new group, add a separator.
            if(!firstHasBeenShown){
                firstHasBeenShown = true;
                paragraphNumber =  doc->GetGroupNumber();
            }

            if( doc->GetGroupNumber() > paragraphNumber){
                paragraphNumber = doc->GetGroupNumber();
                printf ("Paragraph Separator\r\r", 2);
            }

        // Display the weighted score for the sentence
            if(showScore){
                printf("(%3.2f) ", doc->GetRankedHit()->GetScore());
            }
        // Display its rank
            if(showRank){
                printf("(%d)    ", 1 + (int)doc->GetRank());
                }

        // Display the top-ranked words in the extent
            if(numberTopWords >0 ) {

                if(showlevel > 0) {
                    printf ("[");
                    }
```

```
                uint32  numberToShow = numberTopWords;
                if (numberToShow >
                            doc->GetRankedHit()->GetMatchingTermsLen()) {
                        numberToShow =
                            doc->GetRankedHit()->GetMatchingTermsLen();
                        }

                for(int w = 0 ; w< numberToShow; w++) {
                    printf ("%s ",
                doc->GetRankedHit()->GetMatchingTerms()[w]->GetData());
                    }

                if(showlevel >0) {
                    printf ("]\n");
                    }
                }

        // Display the sentence
            if(showSentences) {
                char* txt = (char*)doc->GetExtent();
                printf ("Summary => %s\n", txt);
                IAFreeArray(txt);
            }
        }
    }
}
```

The sections that follow describe the classes and methods used for summarizing documents using the IAT.

The IADocumentAbstractor Class

Ancestors	None.
Subclasses	None.
Header file	`Abstractor.h`

Description

The following methods allow you to use the IAT to summarize the contents of a document.

Public Methods

IADocumentAbstractor

Constructor for this class.

```
IADocumentAbstractor (
                    IAExtentParser* parser,
                    IAStorage* storage,
                    IAAnalysis* analysis);
```

`parser` A pointer to the parser to use to break up the document into extents. This parser must instantiated from a subclass of `IAExtentParser`.

`storage` A pointer to the storage instance to use for the summarization.

`analysis` A pointer to the analysis instance used to rank the extents in the document.

~IADocumentAbstractor

Destructor for this class.

```
virtual ~IADocumentAbstractor();
```

Summarize

Summarizes a document as a ranked list of text extents.

```
virtual void Summarize (
                    RankedProgressFn* progressFn,
                    clock_t progFrequency,
                    void* callerData,
                    uint32 maxLevel,
                    TermIndex* contextIndex);
```

`progressFn` A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.

`progFrequency` The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

`callerData` A pointer to application-specific data that is passed to the client application when the callback occurs.

`maxLevel` The maximum number of extents to include in the summary.

`contextIndex` A pointer to the context index to use for this summarization. The context index is used for calculating term scale factors. If you do not specify a context index, this pointer defaults to `NULL`.

DISCUSSION

After creating the ranked list of extents, you can use the `GetExtents` method (page 22) to retrieve them.

SetAnalysis

Specfies the analysis instance used to rank the extents.

```
void SetAnalysis (IAAnalysis* analysis);
```

`analysis` A pointer to the analysis object.

SetStorage

Specifies the storage instance.

```
void SetStorage (IAStorage* storage);
```

`storage` A pointer to the storage object.

SetParser

Specifies the extent parser.

```
void SetParser (IAExtentParser* parser);
```

`parser` A pointer to the extent parser object.

GetNumberOfExtents

Returns the number of extents parsed in the document.

```
uint32 GetNumberOfExtents () const;
```

DISCUSSION

This method returns the number of text extents. The definition of the extent is dependent on the extent parser used.

GetExtents

Gets the parsed extents that make up the document, in ranked order.

```
SentenceDoc** GetExtents () const;
```

method result A pointer to an array of `IAExtentDoc` pointers. You can determine the number of extents in the array by calling the `GetNumberOfExtents` method (page 22).

DISCUSSION

This method returns an array of pointers to the parsed text extents. The definition of the extent is dependent on the extent parser used.

The IAExtentParser Class

Ancestors None.

Subclass `IAANSISentenceParser`

Header file `Abstractor.h`

Description

You use this class to create extent parsers to use with document summarization. Your extent parser must be a subclass of `IAExtentParser`. If you want a sentence to be the size of the extent, you can simply use the `IAANSISentenceParser` subclass instead of writing your own. See "The IAANSISentenceParser Class" (page 34) for more information.

Public Methods

IAExtentParser

Constructor for this class.

```
IAExtentParser (
                    byte* buffer,
                    uint32 bufferLength,
                    bool removezReturns);
```

`buffer` A pointer to a buffer that holds the text to be parsed.

`bufferLength` The size of the buffer.

`removezReturns`
If true, the parser removes carriage returns from the parsed extents. If you do not specify this parameter, the default is to remove returns.

~IAExtentParser

Destructor for this class.

```
virtual ~IAExtentParser();
```

SetBuffer

Specifes the buffer containing the text to be parsed.

```
void SetBuffer (
                    byte* buffer,
                    uint32 bufferLength);
```

`buffer` A pointer to the buffer containing the text to be parsed.

`bufferLength` The length of the buffer.

GetBuffer

Returns a pointer to the buffer containing the text to be parsed.

```
byte* GetBuffer ();
```

method result A pointer to the buffer.

GetBufferLength

Returns the length of the buffer containing the text to be parsed.

```
uint32 GetBufferLength ();
```

method result The length of the buffer.

SetStartOfExtent

Sets the beginning of a text extent.

```
void SetStartOfExtent (byte* start);
```

`start` A pointer to the beginning of a text extent.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `GetStartOfExtent` method (page 26).

GetStartOfExtent

Returns the beginning of a text extent.

```
byte* GetStartOfExtent ();
```

method result A pointer to the beginning of a text extent.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `SetStartOfExtent` method (page 25).

SetExtentLength

Sets the length of a text extent.

```
void SetExtentLength (uint32 length);
```

`length` The length of a text extent.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `GetExtentLength` method (page 27).

GetExtentLength

Returns the length of a text extent.

```
uint32 GetExtentLength ();
```

method result The length of the text extent.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `SetExtentLength` method (page 26).

SetStartOfNext

Sets the beginning of the next extent.

```
void SetStartOfNext (byte* start);
```

`start` A pointer to the beginning of the next extent.

DISCUSSION

If you are creating a subclass of the `IAExtentParser` class, you may want to use this method to keep track of the beginning of your own extent (such as a paragraph).

SEE ALSO

The `GetStartOfNext` method (page 28).

GetStartOfNext

Returns the beginning of the next extent.

```
byte* GetStartOfNext ();
```

method result A pointer to the beginning of the next extent.

DISCUSSION

If you are subclassing the `IAExtentParser` class, you may want to use this method to keep track of the beginning of your own parser-defined extent (such as a paragraph).

SEE ALSO

The `SetStartOfNext` method (page 27).

SetBytesLeft

Sets the number of bytes left in the buffer.

```
void SetBytesLeft (uint32 length);
```

`length` The number of bytes remaining to be parsed in the buffer.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `GetBytesLeft` method (page 29).

GetBytesLeft

Returns the number of bytes left in the buffer.

```
uint32 GetBytesLeft ();
```

method result The number of bytes remaining to be parsed in the buffer.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

SEE ALSO

The `SetBytesLeft` method (page 28).

SetGroupNumber

Assigns a group number to a text extent.

```
void SetGroupNumber (uint32 gNum);
```

`gNum` The group number to assign.

DISCUSSION

A group is an arbitrary set of extents. For example, if the extent is a sentence, a paragraph is a group of extents. An extent contained in the fifth paragraph of a document could have a group number of 5.

SEE ALSO

The `GetGroupNumber` method (page 30).

The `SetOrderNumber` method (page 30).

GetGroupNumber

Returns the group number assigned to an extent.

```
uint32 GetGroupNumber ();
```

method result The group number assigned to the extent.

DISCUSSION

A group is an arbitrary set of extents. For example, if the extent is a sentence, a paragraph is a group of extents. An extent contained in the fifth paragraph of a document could have a group number of 5.

SEE ALSO

The `SetGroupNumber` method (page 29).

The `GetOrderNumber` method (page 31).

SetOrderNumber

Assigns an order number to a text extent.

```
void SetOrderNumber (uint32 oNum);
```

`oNum` The order number to assign.

DISCUSSION

Extents contained in groups often have order numbers to make tracking them easier. For example, the third sentence (extent) in a paragraph (group) might have the order number 3.

SEE ALSO

The `GetOrderNumber` method (page 31).

The `SetGroupNumber` method (page 29).

GetOrderNumber

Returns the order number assigned to an extent.

```
uint32 GetOrderNumber ();
```

method result The order number assigned to the extent.

DISCUSSION

Extents contained in groups often have order numbers to make tracking them easier. For example, the third sentence (extent) in a paragraph (group) might have the order number 3.

SEE ALSO

The `SetOrderNumber` method (page 30).

The `GetGroupNumber` method (page 30).

IsCancelled

Determines whether the parsing process was cancelled.

```
bool IsCancelled ();
```

method result True if the parsing process was cancelled by the client.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

AreReturnsRemoved

Determines whether carriage returns were removed from the document.

```
bool AreReturnsRemoved ();
```

method result True if carriage returns were removed.

DISCUSSION

In most cases, you need to use this method only if you are creating a subclass of `IAExtentParser`.

GetNextExtent

Returns the next text extent.

```
virtual IAExtentDoc* GetNextExtent (
                    RankedProgressFn* progressFn,
                    clock_t  progFrequency,
                    void* callerData);
```

`progressFn` A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.

`progFrequency` The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

`callerData`	A pointer to application-specific data that is passed to the client application when the callback occurs.
method result	A pointer to the next parsed extent.

DISCUSSION

This method calls the protected method `GettingNextExtent` (page 33), which does the actual work of parsing the next text extent.

Protected Method

GettingNextExtent

Returns the next text extent.

```
virtual IAExtentDoc* GettingNextExtent (
                     RankedProgressFn* progressFn,
                     clock_t progFrequency,
                     void* callerData,
                     bool *endBuffer) = 0;
```

`progressFn`	A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.
`progFrequency`	The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).
`callerData`	A pointer to application-specific data that is passed to the client application when the callback occurs.
`endBuffer`	If true, the parser has reached the end of the buffer
method result	A pointer to the next parsed sentence.

DISCUSSION

This is a pure virtual method that does the actual work for the public method `GetNextExtent` (page 32). When implementing your own extent parsers, you must override this method.

The IAANSISentenceParser Class

Ancestor	`IAExtentParser`
Subclasses	None.
Header file	`Abstractor.h`

Description

The `IAANSISentenceParser` class is a subclass of `IAExtentParser` that breaks up a document into text extents that are sentences. To parse other types of extents, you must write your own extent parser by creating a subclass of `IAExtentParser`.

Public Methods

IAANSISentenceParser

Constructor for this class.

```
IAANSISentenceParser (
                    byte* buffer,
                    uint32 bufferLength,
                    const char* abbrevFile,
                    bool removezReturns);
```

`buffer` A pointer to the buffer that holds the text to be parsed.

`bufferLength` The size of the buffer.

`abbrevFile` A pointer to a string containing the path to an abbreviations file. This text file contains a list of abbreviated words. The sentence parser assumes that a punctuation mark such as a period (.) signals the end of a sentence unless the word containing the mark is contained in this file (for example, the abbreviation e.g. should go in the file). In general this path should point to the file `EnglishAbbreviations`.

`removezReturns`
If true, the parser removes carriage returns from the parsed sentences. If you do not specify this parameter, the default is to remove returns.

~IAANSISentenceParser

Destructor for this class.

```
virtual ~IAANSISentenceParser();
```

SetRequiredUpper

Sets whether a parsed sentence must begin with an upper-case letter.

```
void SetRequiredUpper (bool value);
```

`value` If true, the parser requires a sentence to begin with an upper-case letter. If you do not specify this parameter, the default setting is true.

SEE ALSO

The `IsRequiredUpper` method (page 36).

IsRequiredUpper

Specifies whether a parsed sentence is required to start with an upper-case letter.

```
bool IsRequiredUpper () const;
```

method result If true, the parser requires a sentence to begin with an upper-case letter.

SEE ALSO

The `SetRequiredUpper` method (page 35).

Protected Method

GettingNextExtent

Returns the next parsed sentence in the document.

```
virtual IAExtentDoc* GettingNextExtent (
                    RankedProgressFn* progressFn,
                    clock_t progFrequency,
                    void* callerData,
                    bool* endBuffer);
```

`progressFn` A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.

`progFrequency` The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

`callerData` A pointer to application-specific data that is passed to the client application when the callback occurs.

`endBuffer` If true on output, the parser has reached the end of the buffer

method result A pointer to the next parsed sentence.

DISCUSSION

This method is called by the public method `GetNextExtent` (page 32).

The IAExtentCorpus Class

Ancestor `IACorpus`

Subclasses None.

Header file `Abstractor.h`

Description

The `IAExtentCorpus` class is a subclass of `IACorpus` that represents a set of strings in memory (such as a collection of text extents). The actual body of the strings is maintained in a buffer managed by the client. For more information about corpus classes, see Chapter 8, "Corpus Category," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Methods

IAExtentCorpus

Constructor for this class.

```
IAExtentCorpus (uint32 corpType);

IAExtentCorpus (byte* buffer, uint32 corpType);
```

`buffer`	A pointer to the buffer holding the text associated with this corpus.
`corpType`	An integer specifying the type of extents this corpus handles. If you do not specify this parameter, the default corpus type is `ExtentCorpusType`.

~IAExtentCorpus

Destructor for this class.

```
virtual ~IAExtentCorpus();
```

GetProtoDoc

Tells the index the type of documents this corpus represents.

```
IADoc* GetProtoDoc ();
```

method result	A pointer to a new instance of the document type (for example, an instance of `IAExtentDoc`).

GetDocText

Retrieves the text associated with an extent.

```
IADocText* GetDocText (const IADoc* doc);
```

doc	The extent whose text you want to retrieve.
method result	A pointer to the extent text.

GetBuffer

Retrieves the buffer associated with the corpus.

```
byte* GetBuffer ();
```

method result A pointer to the buffer.

The IAExtentDoc Class

Ancestor	`IADoc`
Subclasses	None.
Header file	`Abstractor.h`

Description

The `IAExtentDoc` class is a subclass of `IADoc` that represents a text extent in a corpus represented by `IAExtentCorpus`. See "The IAExtentCorpus Class" (page 37) for more information about `IAExtentCorpus`.

Public Methods

IAExtentDoc

Constructor for this class.

```
IAExtentDoc (
                    const byte* buffer,
                    uint32 offset,
                    uint32 textLength,
                    uint32 extentNumber,
```

```
                    uint32 groupNumber,
                    uint32 rank,
                    RankedHit* rankHitVal);
```

`buffer` A pointer to the buffer containing the text extent.

`offset` The offset of the text extent in the buffer, in bytes.

`textLength` The length of the text extent, in bytes.

`extentNumber` The extent number assigned to the extent.

`groupNumber` The group number assigned to the extent.

`rank` The rank assigned to the extent. If you do not specify this parameter, the default rank is 0.

`rankHitVal` The ranked hit value assigned to the extent. If you do not specify this parameter, the default ranked hit value is `NULL`.

SEE AlSO

"The IADocumentAbstractor Class" (page 19).

"The IAExtentParser Class" (page 23).

~IAExtentDoc

Destructor for this class.

```
virtual ~IAExtentDoc();
```

DeepCopy

Creates a deep copy of the extent.

```
IAStorable* DeepCopy () const;
```

method result Apointer to the copy of the extent.

DISCUSSION

Calling `DeepCopy` creates a copy of the extent as well as copies of any objects referenced by the extent.

StoreSize

Stores the size of the extent.

```
uint32 StoreSize () const;
```

method result The size of the extent, in bytes.

Store

Stores the extent.

```
void Store (IAOutputBlock *output) const;
```

`output` A pointer to the output block in which to store the extent.

SEE ALSO

The `Restore` method (page 41).

Restore

Restores the extent from a given block.

```
IAStorable* Restore (IAInputBlock *input) const;
```

`input` A pointer to the block containing the extent.

method result A pointer to the restored extent.

SEE ALSO

The `Store` method (page 41).

LessThan

Checks if the extent is less than another.

```
bool LessThan (const IAOrderedStorable neighbor) const;
```

`neighbor` The extent you want to compare against.

method result True if the extent is less than that of `neighbor`.

Equal

Checks to see if two extents are equal.

```
byte* Equal (const IAOrderedStorable neighbor) const;
```

`neighbor` The extent you want to compare against.

method result True if the two extents are equal.

GetText

Gets the buffer associated with the extent.

```
byte* GetText () const;
```

method result A pointer to the buffer containing the text.

GetLength

Gets the length of an extent.

```
uint32 GetLength () const;
```

method result The length of the extent, in bytes.

GetOffset

Gets the offset of the extent within the buffer.

```
uint32 GetOffset () const;
```

method result The offset of the extent, in bytes.

GetExtent

Gets the extent as a null-terminated string.

```
byte* GetExtent () const;
```

method result A pointer to the extent.

DISCUSSION

Space for the returned extent is allocated by `IAMallocArray`, so you must call `IAFreeArray` to free the memory when you no longer need the extent.

SetRank

Sets the rank of an extent.

```
void SetRank (uint32 rank) const;
```

`rank` The rank to assign to the extent.

SEE ALSO

The `GetRank` method (page 44).

GetRank

Gets the rank of an extent.

```
uint32 GetRank () const;
```

method result The rank assigned to the extent.

SEE ALSO

The `SetRank` method (page 44).

SetRankedHit

Sets the ranked hit value of an extent.

```
void SetRank (uint32 rankHitVal) const;
```

`rankHitVal` The ranked hit value to assign to the extent.

SEE ALSO

The `GetRankedHit` method (page 45).

GetRankedHit

Gets the ranked hit value of an extent.

```
uint32 GetRankedHit () const;
```

method result The ranked hit value assigned to the extent.

SEE ALSO

The `SetRankedHit` method (page 44).

GetExtentNumber

Gets the extent number assigned to the extent.

```
uint32 GetExtentNumber () const;
```

method result The extent number assigned to the extent.

SEE ALSO

"The IAExtentParser Class" (page 23).

GetGroupNumber

Gets the group number assigned to the extent.

```
uint32 GetGroupNumber () const;
```

method result The group number assigned to the extent.

SEE ALSO

"The IAExtentParser Class" (page 23).

Document Routing

Contents

In addition to searching for documents, you can use the IAT to route documents into particular categories. For example, say you have a collection of unsorted documents related to transportation that you want to divide into three categories: planes, trains, and automobiles. After setting up the various categories, you can use the IAT to compare documents from the unsorted collection with examples for each category. For each category you must have specified one or more documents as examples of what it should contain; that is, you would "prime" the trains category with documents heavily-related to trains, and so on. The category that provides the best fit (the highest ranked score) is the most appropriate place to put the unsorted document.

This chapter describes how to use the IAT to define document categories and route documents among those categories.

Clusters

A category for related documents is called a **cluster**. Clusters are represented by the `IACluster` class, which must be subclassed to handle particular document types. For example, the IAT provides the subclass `HFSCluster`, which represents a cluster of HFS documents (that is, Mac OS files). When subclassing `IACluster`, you must override the `GetNextDoc` method, which returns the next document in the cluster, and the `Reset` method, which resets the iterator.

For a given cluster, you must provide one or more example documents, which are used to establish weighting criteria when comparing the cluster to an unsorted document. For an HFSCluster, these documents are contained in the folder associated with the cluster.

The Document Router

After establishing your clusters, the IAT **router**, defined in the class `IARouter`, lets you identify the cluster that offers the best fit for an unsorted document.

Note
The router does not copy or move the unsorted document; it only identifies the cluster to which the document belongs. Any moving or copying actions must be done by your application. ◆

When seeking the best fit for a document, you can add the weighting (that is, the normalized term-weighted vector, or TWVector) of that document to the weighting of the appropriate cluster, if desired. That is, each additional document that fits in the cluster helps define what should be there.

Listing 2-1 shows an example that displays the best fit cluster for each of a number of unsorted documents.

Listing 2-1 Identifying best fit clusters for documents using the router

```
// Enter the name of folder containing the items to route
    StringPtr unSortedItems = "\pMyDisk:Folders:Disorganized Stuff";

// Enter name of the index
    StringPtr singleIndexName = "\pMyDisk:Folders:test.index";

// Enter the names of the router folders
    StringPtr clusterFolders[] = {
        "\pMyDisk:Folders:Planes",
        "\pMyDisk:Folders:Trains",
        "\pMyDisk:Folders:Automobiles",
        "\p" // empty string to mark end
        };


void DemoRouting()
{
    FSSpec fsSpec;
    char str[256];

// Create/initialize our index
    (void)FSMakeFSSpec(0, 0, singleIndexName, &fsSpec);

    IAStorage* myStorage = MakeHFSStorage(fsSpec.vRefNum, fsSpec.parID,
                                fsSpec.name);
```

```
    myStorage->Initialize();

    HFSCorpus myCorpus = new HFSCorpus;
    IAAnalysis* myAnalysis = new SimpleAnalysis();

    VectorIndex* myIndex = new VectorIndex(myStorage,myCorpus,
                                myAnalysis);
    myIndex->Initialize();


// Setup clusters
    uint32 clusterCount = 0;
    for (clusterCount = 0; clusterFolders[clusterCount][0] != 0;
            clusterCount++ ) {}

    HFSCluster** folders = new HFSCluster*[clusterCount];

    for (uint32 i  = 0; i < clusterCount; i++ ) {
        folders[i] = new HFSCluster(myIndex, clusterFolders[i]);
        }


// Instantiate a router and initialize with the corpuses representing
// our clusters.
    IARouter myRouter (myIndex);
    myRouter.InitializeClusters((IACluster**)folders, clusterCount);

    AddItemsToIndex(unSortedItems, myIndex);
    myIndex->Flush();

    HFSTextFolderCorpus* source = new HFSTextFolderCorpus(unSortedItems);

    IADocIterator* docs = source->GetDocIterator();
    IADoc* doc = docs->GetNextDoc();


// Now loop through each unsorted document and find the best cluster
    while (doc) {
        uint32 clusterIndex = myRouter.WhichCluster(doc, false);
        printf ("%s belongs in cluster %d\n",
            PToCStr(((HFSDoc*)doc)->GetFileName(), str), ++clusterIndex);
```

```
        delete doc;
        doc = docs->GetNextDoc();
        }


// Cleanup
    delete docs;
    delete source;

    myRouter.Store();
    myIndex->Flush();
    myStorage->Commit();

    delete myIndex;
    delete myStorage;

    for (uint32 i  = 0; i < clusterCount; i++ ) {
        delete folders[i];
        }
    delete [] folders;
    }
// End DemoRouting


// This method is called by the DemoRouting method.
void AddItemsToIndex(StringPtr folderPathName, VectorIndex* inIndex)
{
    FSSpec myFsSpec;
    OSErr err = FSMakeFSSpec(0, 0, folderPathName, &myFsSpec);
    IAAssertion(err == noErr, "Can't get folder", IAAssertionFailure);

    HFSIterator folderIterator(fsSpec.vRefNum,FSSpecToDirID(&myFsSpec));

    IATry {
        while (folderIterator.Increment()) {
            CInfoPBRec* pb = folderIterator.GetPBRec();

            HFSDoc* doc = new HFSDoc((HFSCorpus*)inIndex->GetCorpus(),
                pb->hFileInfo.ioVRefNum, pb->hFileInfo.ioFlParID,
                pb->hFileInfo.ioNamePtr);
            inIndex->AddDoc(doc);
```

```
            }
        }
    IACatch (const IAException& exception) {
        printf("%s, %s\n", exception.What(), exception.GetLocation());
        }
    }
// End AddItemsToIndex
```

In this example, after creating the index, the corpus, and specifying the type of analysis, the `DemoRouting` method sets up clusters based on what was defined in `clusterFolders[]`. Each folder in the array should contain example documents defining the type of document that should belong to the cluster. Documents to be routed should be in the `unSortedItems` folder.

After initializing the clusters by calling the `InitializeClusters` method, the router (`myRouter`) then simply cycles through the corpus representing the contents of `unSortedItems` and calls the `WhichCluster` method for each document. If you set the second parameter in `WhichCluster` to true, the weighting of the document to be routed is added to the appropriate cluster when a match is made.

In this example, after the `DemoRouting` method routes all the documents in `unSortedItems`, it calls the `Store` method before removing all instantiated objects. Doing so saves the cluster information and weightings so you can retrieve them at some later time. If you specified that the cluster accumulate weightings as documents were routed, the saved settings will reflect the additional weightings. If you want to route additional documents later using the stored cluster settings, you call the `Restore` method instead of instantiating clusters and calling `InitializeClusters`.

The sections that follow describe the classes and methods used for routing documents using the IAT.

The IARouter Class

Ancestors	None.
Subclasses	None.
Header file	`IARouter.h`

Description

The following methods allow you to use the IAT to sort documents into clusters. Note that the IAT only specifies which cluster to put a document in; your application must copy or move the document based on the categorization.

Public Methods

IARouter

Constructor for this class.

```
IARouter (
                VectorIndex* index,
                TProgressFn* progressFn,
                clock_t progressFreq,
                void* appData);
```

`index` A pointer to the vector index.

`progressFn` A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.

`progressFreq` The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

`appData` A pointer to application-specific data that is passed to the client application when the callback occurs.

~IARouter

Destructor for this class.

```
virtual ~IARouter();
```

InitializeClusters

Specifies clusters to use in the routing.

```
void InitializeClusters (
            IACluster** clusters,
            uint32 howManyClusters);
```

`clusters` A pointer to an array of `IACluster` pointers specifying the clusters to use.

`howManyClusters`
The number of `IACluster` pointers in the array.

DISCUSSION

You call `InitializeClusters` when you want to route documents using a new set of clusters. If you want to route documents using an older saved set of clusters, you should call the `Restore` method (page 56) instead.

WhichCluster

Specifies the cluster to which a document belongs.

```
uint32  WhichCluster (
                  IADoc* doc,
                  bool accumulate);
```

`doc` A pointer to a document to be routed.

`accumulate` If true, the normalized weighting of the specified document (that is, its TWVector) is added to the weighting of the cluster. If you do not specify this parameter, the default is false.

method result An value specifying the index of the cluster to which the document belongs.

DISCUSSION

The `WhichCluster` method does not move or copy the document to the indicated cluster. If you want to move the document, your application must do so itself.

Store

Stores the router settings.

```
void Store (
                    IAStorage* storage,
                    IABlockID block) const;
```

`storage` A pointer to a `IAStorage*` object. If you do not specify this parameter, the default is `NULL`, and `Store` uses the storage instance that contains the index used by the router.

`block` The ID of the block in which you want to store the router settings. If you do not specify this parameter, the default block ID is 0.

SEE ALSO

The `Restore` method (page 56).

Restore

Restores saved router settings.

```
void Restore (
              IAStorage* storage,
              IABlockID block);
```

`storage` A pointer to a `IAStorage` object containing the router information. If you do not specify this parameter, the IAT attempts to restore the setting from the storage instance used by the index associated with the router.

`block`

The ID of the block containing the router settings you want to retrieve. If you do not specify this parameter, the default block ID is 0.

SEE ALSO

The `Store` method (page 56).

The `InitializeClusters` method (page 55).

StoreSize

Stores the size of the current router.

```
IABlockSize StoreSize () const;
```

method result The size of the router.

GetProgressFn

Returns a pointer to the application-defined progress function.

```
TProgressFn* GetProgressFn () const;
```

method result A pointer to the application-defined function. The IAT calls back to this function periodically to allow the client time to do other things, if desired.

GetProgressData

Returns the progress function data.

```
void* GetProgressData () const;
```

method result A pointer to the data passed to the application at the time of the progress function callback. You specify the location of this data when the `IARouter` constructor is called.

GetProgressFreq

Returns the time between calls to the progress function callback.

```
clock_t GetProgressFreq () const;
```

method result The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

Protected Methods

BestCluster

Returns the best cluster for a given TWVector.

```
uint32 BestCluster (TWVector *vector) const;
```

`vector` A pointer to a `TWVector` object.

method result The index of the cluster to which the TWVector best fits.

DISCUSSION

If you want to subclass `IARouter` and implement your own best cluster algorithm, you may want to override this method.

ClearAccumulator

Clears the accumulator associated with the router.

```
void ClearAccumulator (void);
```

AddDocVectorToAccumulator

Adds a document's TWVector to the accumulator.

```
void AddDocVectorToAccumulator (TWVector* newDocVector);
```

`newDocVector` A pointer to the `TWVector` object representing the document.

SEE ALSO

The `AccumulateDocVector` method (page 59).

AccumulateDocVector

Adds the TWVector representing a document to the accumulator.

```
void AccumulateDocVector(IADoc* doc);
```

`doc` A pointer to a document.

SEE ALSO

The `AddDocVectorToAccumulator` method (page 59).

AddToAccumulator

Adds the specified TWVector to the weighting of a given cluster.

```
void AddToAccumulator (
            uint32 cluster,
            TWVector *docVector);
```

`cluster` The cluster to whose weighting you want to add the TWVector.

`docVector` A pointer to the TWVector for a given document.

DISCUSSION

This method adds a TWVector to the weighting of a particular cluster, not the accumulator that is generated during the `WhichCluster` call.

The IACluster Class

Ancestors	None.
Subclass	`HFSCluster`
Header file	`IARouter.h`

Description

This abstract class represents a cluster of documents. You must subclass this class to represent clusters of an actual document format. For example, the `HFSCluster` class is a subclass of `IACluster` that you use to represent clusters of HFS format documents (that is, Mac OS files).

Public Methods

IACluster

Constructor for this class.

```
IACluster (IAIndex* index);
```

`index` A pointer to the index that contains this cluster.

~IACluster

Destructor for this class.

```
virtual ~IACluster ();
```

GetNextDoc

Gets the next document in the cluster.

```
virtual IADoc* GetNextDoc () const;
```

method result A pointer to the next document in the cluster.

DISCUSSION

The type of document returned depends on the `IACluster` subclass. For example, the `HFSCluster` subclass of `IACluster` returns documents of type `HFSDoc`.

Reset

Resets the iterator.

```
virtual void Reset ();
```

DISCUSSION

After you call `Reset`, the `GetNextDoc` function (page 61) begins with the first document in the cluster.

Protected Method

GetCorpus

Retrieves the corpus in the index associated with the cluster.

```
IACorpus* GetCorpus () const;
```

method result A pointer to the corpus.

The HFSCluster Class

Ancestor `IACluster`

Subclasses None.

Header file `HFSCluster.h`

Description

The `HFSCluster` class is a subclass of `IACluster` that handles clusters of HFS documents (that is, Mac OS files).

Public Methods

HFSCluster

Constructor for this class.

```
HFSCluster (
                IAIndex* index
                StringPtr clusterName);
```

`index` The index to contain this cluster.

`clusterName` A pointer to the pathname of the folder containing document examples for the cluster.

~HFSCluster

Destructor for this class.

```
virtual ~HFSCluster ();
```

GetNextDoc

Gets the next HFS document in the cluster.

```
IADoc* GetNextDoc () const;
```

method result A pointer to the next HFS document in the cluster.

DISCUSSION

The `HFSCluster` subclass of `IACluster` returns documents of type `HFSDoc`.

Reset

Resets the iterator.

```
void Reset ();
```

DISCUSSION

After you call `Reset`, the `GetNextDoc` function (page 63) begins with the first document in the cluster.

Additions and Changes to Existing Classes

Contents

This chapter describes new information about existing IAT classes documented in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Additional Information About the IAMutex Class

As mentioned in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*, you can use the `IAMutex` class to coordinate access to a storage object when using or developing multi-threaded applications. No thread synchronization or access control (thread safety) is necessary when reading a storage object. However, only one thread of a multi-threaded application should be able to write to the storage at any particular time. Doing so prevents lockouts and accidental overwriting of data. The `IAMutex` class contains methods that let you create a mutex (a mutual-exclusion handler or semaphore) that can coordinate write access between multiple threads.

Creating a Subclass of IAMutex

You must create a subclass of the `IAMutex` class if you want to prevent multiple-write access. Your subclass should be able to lock out other threads if one thread is writing to the storage object, and it should be able to transfer write access to another thread when necessary. Listing 3-1 shows an example subclass of `IAMutex`. This example uses the Metrowerks `LMutexSemaphore` class (provided with PowerPlant) to implement the `IAMutex` protocol, but you may choose another implementation if desired.

Listing 3-1 A subclass of IAMutex

```
#include "PowerPlantMutex.h"
#include <LMutexSemaphore.h>
#include <LThread.h>
#include <UException.h>
#include <UThread.h>

class PowerPlantMutex : public IAMutex, public LMutexSemaphore {
public:
```

```
    void        Lock();
    void        Unlock();
    void        Signal();
    };

void PowerPlantMutex::Lock() {
    ThrowIfOSErr_(Wait());
    }

void PowerPlantMutex::Unlock() {
    Signal();
    }

void PowerPlantMutex::Signal() {
// code copied from LMutexSemaphor::Signal(), except this version
// does not call LThread::Yield()
    LThread *thread = LThread::GetCurrentThread();
    {
        StCritical critical; // disable preemption within this block
        if (thread != mOwner) {
            Throw_(errSemaphoreNotOwner);
        } else if (mNestedWaits > 0) {
            --mNestedWaits;
        } else if (mExcessSignals < 0) {
            THREAD_ASSERT(mThreads.qHead != NULL);
            mOwner = UnblockThread(mThreads.qHead, noErr);
        } else {
            THREAD_ASSERT(mExcessSignals == 0);
            mOwner = NULL;
            ++mExcessSignals;
        }
    }
    //LThread::Yield();
}
```

The `PowerPlantMutex` class inherits from both the `IAMutex` and PowerPlant `LMutexSemaphore` classes.

The `Lock` method takes one of two actions:

- If no other thread currently has the semaphore (allowing it write access, in this case) then it gives the current thread the semaphore and puts the others in a wait state.
- If another thread has the semaphore, put the current thread in a wait state.

The `Unlock` method calls the `Signal` method, which indicates that the current thread is giving up its semaphore. The semaphore can then pass to a waiting thread (if any).

Registering Your Mutex With the IAT

After implementing your `IAMutex` subclass, you must register the class with the IAT. This allows the IAT to create mutex instances as necessary for each thread.

To register your mutex, you must set the `IANewMutex` variable. `IANewMutex` is declared as a pointer to a function with the following prototype:

```
IAMutex* myFuncName();
```

For example, if you wanted to register the `PowerPlantMutex` class in Listing 3-1, you would have to cast it as a function that matches the required `IANewMutex` prototype:

```
IAMutex* NewPowerPlantMutex() {
    return new PowerPlantMutex();
    }
```

Then, to register `PowerPlantMutex`, you would define `IANewMutex` as follows:

```
IANewMutex = &NewPowerPlantMutex;
```

You should register your mutex as part of any global initializations in your application and before you create your index. The IAT can then create and remove mutexes as needed.

Note
For single-threaded applications, the `IANewMutex` variable automatically defaults to the address of `IADefaultMutexConstructor`, which implements a dummy (no-op) mutex. ◆

The TWVector Class

Ancestors None.

Subclasses None.

Header file `TWVector.h`

Description

The `TWVector` class now contains additional methods.

For more information about TWVectors and the other available methods, see Chapter 6, "Accessor Category," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Methods

HasNegativeComponents

Determines whether the vector has components with negative weight.

```
bool HasNegativeComponents () const;
```

method result If true, the vector contains components with negative weight.

StoreSize

Stores the size of the vector.

```
IABlockSize StoreSize () const;
```

method result The size of the vector, in blocks.

DeepCopy

Creates a deep copy of the vector.

```
TWVEctor* DeepCopy () const;
```

method result A pointer to the copy of the vector.

DISCUSSION

Calling `DeepCopy` creates a copy of the vector as well as copies of any objects referenced by the vector.

Store

Stores the vector.

```
void Store (IAOutputBlock *output) const;
```

`output` A pointer to the output block in which to store the vector.

SEE ALSO

The `Restore` method (page 71).

Restore

Restores the vector from a given block.

```
TWVector* Restore (IAInputBlock *input) const;
```

`input` A pointer to the block containing the vector.

method result A pointer to the restored vector.

SEE ALSO

The `Store` method (page 71).

SortByTerm

Sorts the vector by term.

```
void SortByTerm ();
```

SortByWeight

Sorts the vector by weight.

```
void SortByWeight ();
```

SEE ALSO

The `SortByAbsoluteWeight` method (page 72).

SortByAbsoluteWeight

Sorts the vector by the absolute value of the component weight.

```
void SortByAbsoluteWeight ();
```

SEE ALSO

The `SortByWeight` method (page 72).

Truncate

Removes terms with lower weights.

```
void Truncate (uint32 maxTerms);
```

`maxTerms` The maximum number of terms to keep. `Truncate` removes terms, starting with the lowest weights, until `maxTerms` or less are left.

SEE ALSO

The `TruncateAbsoluteValue` method (page 73).

TruncateAbsoluteValue

Removes terms with lower absolute weights.

```
void TruncateAbsoluteValue (uint32 maxTerms);
```

`maxTerms` The maximum number of terms to keep. `Truncate` removes terms starting with the lowest absolute weights until `maxTerms` or less are left.

SEE ALSO

The `Truncate` method (page 73).

AddIntoAverage

Adds a new vector to the current one and returns a vector representing the result.

```
TWVector* AddIntoAverage (
                    const TWVector *newVector,
                    uint32 totalVectorCount,
                    bool invertNewVector);
```

`newVector` A pointer to the new vector.

`totalVectorCount`
The total number of vectors in the result.

`invertNewVector`
If true, the new vector is inverted before adding it to the current one.

method result A pointer to the resulting vector.

AddWeighted

Adds a new vector to the current one, allowing weighting adjustments, and returns a new vector representing the result. Both vectors must have positive component weights.

```
TWVector* AddWeighted (
                    const TWVector *vector,
                    float weightFactorCurrent,
                    float weightFactorAdded);
```

`vector` A pointer to the new vector.

`weightFactorCurrent`
The weighting factor for the current vector. The component weight of the current vector is multiplied by this value.

`weightFactorAdded`
The weighting factor for vector being passed in (that is, `*vector`). The component weight of the vector is multiplied by this value.

method result A pointer to the resulting vector.

DISCUSSION

The weight factors for the vectors are used to ensure that the components in the resulting vector have the same level of importance.

This method does not allow either vector's component weight to be less than zero. If either vector has a negative component weight, you must use the `AddWeightedAllowNegatives` method (page 75) instead.

AddWeightedAllowNegatives

Adds a new vector to the current one, allowing weighting adjustments, and returns a vector representing the result.

```
TWVector* AddWeightedAllowNegatives (
                    const TWVector *vector,
                    float weightFactorCurrent,
                    float weightFactorAdded);
```

`vector` A pointer to the new vector.

`weightFactorCurrent`
The weighting factor for the current vector. The component weight of the current vector is multiplied by this value.

`weightFactorAdded`
The weighting factor for vector being passed in (that is, `*vector`). The component weight of the vector is multiplied by this value.

method result A pointer to the resulting vector.

DISCUSSION

The weight factors for the vectors are used to ensure that the components in the resulting vector have the same level of importance. This method allows vectors to have negative component weights.

The IAStruct Class

Ancestors	None.
Subclasses	Most IAT classes.
Header file	`IACommon.h`

Description

This class now contains a virtual destructor.

The `IAStruct` class is the base class for most IAT classes. For more information about the `IAStruct` class, see Chapter 4, "Common Practices in IAT," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Method

~IAStruct

Destructor for this class.

```
virtual ~IAStruct ();
```

The IAAccessor Class

Ancestors	None.
Subclass	`RankedAccessor`
Header file	`IAAccessor.h`

Description

The `IAAccessor` class now contains additional methods.

For more information about the `IAAccessor` class and the other available methods , see Chapter 6, "Accessor Category," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Methods

Store

Stores an accessor.

```
void Store (IAStorage* storage,
                    IABlockID block);
```

`storage`	The storage object to hold the saved accessor. If you do not specify this parameter, `Store` uses the default storage object.
`block`	The block ID of the storage object in which to save the accessor. If you do not specify this parameter, the default ID is 0.

SEE ALSO

The `Update` method (page 78).

Update

Updates an accessor.

```
void Update (
                    IAStorage* storage,
                    IABlockID block);
```

`storage` The storage object to hold the updated accessor. If you do not specify this parameter, `Update` uses the default storage object.

`block` The block ID of the storage object in which to update the accessor. If you do not specify this parameter, the default ID is 0.

DISCUSSION

The `Update` method allows you to update a stored accessor to reflect changes in one or more indexes. For example, if the index has changed since the last time you stored the accessor, you can update the accessor with the changes without having to reinitialize the entire accessor. However, the updates are not stored if you do not call the `Store` method before disposing the accessor.

Calling the `Update` method on an empty accessor initializes the accessor.

SEE ALSO

The `Store` method (page 77).

The HFSIterator Class

Ancestors None.

Subclasses None.

Header file `HFSIterator.h`

Description

The `HFSIterator` class now has a constructor that supports callbacks to an application-defined progress function.

For more information about the `IAAccessor` class and the other available methods, see Chapter 8, "Corpus Category," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Methods

HFSIterator

Constructor for this class.

```
HFSIterator (
                short vRefNum,
                long rootDirId,
                TProgressFn* progressFn,
                clock_t progressFreq,
                void* appData);
```

`vRefNum`	The HFS volume reference number.
`rootDirId`	The directory ID of the highest level folder. If you do not specify this parameter, the default is the volume root.
`progressFn`	A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.
`progressFreq`	The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).
`appData`	A pointer to application-specific data that is passed to the client application when the callback occurs.

DISCUSSION

The callback is invoked during calls to the `GetDirectoryInfo` method.

The HFSTextFolderCorpus Class

Ancestors	`IACorpus --> HFSCorpus`
Subclasses	None.
Header file	`HFSTextFolderCorpus.h`

Description

The `GetDocIterator` method now supports callbacks to an application-defined progress function.

For more information about the `IACorpus` class and the other available methods, see Chapter 8, "Corpus Category," in the *Apple Information Access Toolkit v.1.0 Programmer's Guide*.

Public Methods

GetDocIterator

Constructs a document iterator.

```
IADocIterator * GetDocIterator (
                    TProgressFn* progressFn,
                    clock_t progressFreq,
                    void* appData);
```

`progressFn` A pointer to an application-defined progress function. If not `NULL`, the IAT calls this function periodically to give the client application control.

`progressFreq` The wait time between callbacks, in clock ticks (using the ANSI clocks_per_sec standard).

`appData`	A pointer to application-specific data that is passed to the client application when the callback occurs.

DISCUSSION

The callback is invoked during calls to the `GetDirectoryInfo` method.

Revision History

Table A-1 lists changes made to this document since its creation.

Table A-1 Revision History

Date	Changes and notes
2/18/98	Method `GetNumberOfSentences` now replaced by `GetNumberOfExtents`. Method `GetSentences` replaced by `GetExtents`.
	Added reference material for `IAExtentCorpus` and `IAExtentDoc` classes.
	Tweaked sample code. All `SentenceDoc` objects in sample code replaced by `IAExtentDoc` objects.
2/9/98	Cleaned up and commented sample code. Added glossary and See Also cross-references in reference sections. Revised text based on editorial comments.
1/28/98	First public release.

APPENDIX A

Revision History

Glossary

cluster A category of related documents. A cluster is represented by objects of the `IACluster` class.

document summary A subset of a document that best typifies the document's contents. A document summary is typically made up of one or more text extents. See also **text extent.**

extent parser A mechanism that searches for and isolates text extents. See also **text extent.**

mutex A mutual exclusion semaphore, used to ensure that two threads cannot perform the same action at the same time. For example, in the IAT, you can use the mutex to ensure that two different threads do not attempt to write to the same storage object at the same time.

router A mechanism for selecting the best category to assign to a given document. Typically a router compares an unsorted documents with sample documents from each available category and determines the best fit.

semaphore A special flag that can be set by a thread in a multithreaded application. For example, possession of the semaphore could indicate that that thread currently has the attention of the processor and all other threads must wait.

term-weighted vector (TWVector) A vector constructed from the weighted terms (or tokens) in a document. You can determine the similarity of two documents by seeing how closely their TWVectors match.

text extent A logical subset of a document. For example, a text extent could be a sentence, or a paragraph, or even just a block of 40 characters.

Index

I

L

M

P

R

S

T

U

W

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Jun Suzuki

PEER EDITORS
Steve Evangelou and Lisa Karpinski

PRODUCTION EDITOR
Glen Frank

Special thanks to Farzin Maghoul and Wayne LoofBourrow.

Acknowledgments to Donna S. Lee and Tony Francis.

