Using iKnow

Version 2012.2
13 September 2017
# Table of Contents

About This Book .......................................................................................................................... 1

1 Conceptual Overview .................................................................................................................. 3
  1.1 A Simple Use Case .................................................................................................................. 115
  1.2 What is iKnow? ....................................................................................................................... 4
    1.2.1 What iKnow Isn’t .............................................................................................................. 115
  1.3 Logical Text Units Identified by iKnow .................................................................................. 5
    1.3.1 Entities ............................................................................................................................. 115
    1.3.2 CRCs, CCs, and Paths ....................................................................................................... 115
  1.4 Smart Indexing ....................................................................................................................... 6
  1.5 Smart Matching ..................................................................................................................... 6

2 iKnow Implementation ................................................................................................................. 7
  2.1 A Note on Program Examples ................................................................................................. 7
  2.2 A Note on %Persistent Object Methods ............................................................................... 115
  2.3 A Note on %iKnow and %SYSTEM.iKnow ........................................................................... 115
  2.4 Space Requirements and iKnow Globals ............................................................................... 8
    2.4.1 Batch Load Space Allocation ............................................................................................ 115
  2.5 Input Data ............................................................................................................................. 9
    2.5.1 File Formats ..................................................................................................................... 115
    2.5.2 SQL Record Format .......................................................................................................... 115
    2.5.3 Text Normalization .......................................................................................................... 115
    2.5.4 User-defined Source Normalization ................................................................................... 115
  2.6 Output Structures ................................................................................................................... 10
  2.7 Constants ................................................................................................................................ 10
  2.8 Error Codes .......................................................................................................................... 11
  2.9 Upgrading iKnow Data ......................................................................................................... 11

3 Setting Up the iKnow Environment ........................................................................................... 13
  3.1 iKnow Domains .................................................................................................................... 13
    3.1.1 Defining a Domain ........................................................................................................... 115
    3.1.2 Setting Domain Parameters ............................................................................................ 14
    3.1.3 Assigning to a Domain ..................................................................................................... 16
    3.1.4 Deleting All Data from a Domain ..................................................................................... 17
    3.1.5 Listing All Domains ....................................................................................................... 17
    3.1.6 Copying a Domain .......................................................................................................... 115
  3.2 iKnow Configurations .......................................................................................................... 19
    3.2.1 Defining a Configuration ................................................................................................. 115
    3.2.2 Setting Configuration Properties ...................................................................................... 115
    3.2.3 Defining a UserDictionary ............................................................................................... 20
    3.2.4 Using a Configuration ...................................................................................................... 115
    3.2.5 Listing All Configurations .............................................................................................. 22

4 Loading Text Data into an iKnow Domain .................................................................................. 23
  4.1 Loader .................................................................................................................................. 23
    4.1.1 Loader Error Logging ....................................................................................................... 115
    4.1.2 Loader Reset() .................................................................................................................. 115
  4.2 Lister .................................................................................................................................... 24
    4.2.1 Initializing a Lister ............................................................................................................ 115

Using iKnow
7 Filtering Sources ............................................................................................................................... 55
  7.1 Supported Filters .................................................................................................................... 115
    7.1.1 Filter Operators ....................................................................................................... 115
  7.2 Filtering by the ID of the Source .......................................................................................... 56
    7.2.1 By External Id ....................................................................................................... 115
    7.2.2 By Source Id ....................................................................................................... 115
  7.3 Filtering a Random Selection of Sources ................................................................................. 57
  7.4 Filtering by Number of Sentences ............................................................................................ 58
  7.5 Filtering by Indexing Date ....................................................................................................... 59
List of Tables

Table I–1: Basic Domain Parameters .................................................................................................. 118
Table I–2: Advanced Domain Parameters ........................................................................................... 122
About This Book

This book describes how to use the iKnow semantic analysis engine to access and analyze unstructured data. Commonly, unstructured data consists of a large number of text sources, such as a collection of newspaper articles or a collection of doctors’ notes. You load this text data into iKnow, and then use iKnow to retrieve meaningful information. iKnow operates on data loaded from source texts; it does not modify source texts. iKnow can perform semantic analysis on texts in Dutch (nl), English (en), French (fr), German (de), Portuguese (pt), and Spanish (es).

The book addresses a number of topics:

• A Conceptual Overview, which describes the iKnow approach to unstructured data and iKnow architecture. It describes both what iKnow is and what iKnow is not, so that users can determine whether iKnow is the best fit for their text access application.

• iKnow Implementation, which describes the implementation of iKnow software in the Caché ObjectScript environment and describes data source considerations. Data sources can be text files, SQL records, globals, RSS feeds, or any other type of text source.

• Setting Up an iKnow Environment and Loading Text Data into an iKnow Domain which describe how to write programs to get unstructured data into iKnow.

• iKnow Queries describes how to write programs that can be run against data loaded into iKnow. Filtering Sources describes how to use various filters to limit those queries to a subset of the data sources loaded into iKnow. Equivalence Sets can be applied when pairs of entities should be considered to be equivalent.

• The two Smart Matching chapters describe how to create a dictionary, and then how to use a dictionary to match against data sources loaded into iKnow.

• The User Interfaces chapter describes several sample GUI interfaces for retrieving information. These interfaces use the queries, filters, and dictionaries described in the prior chapters.

• The iKnow KPIs and DeepSee Dashboards chapter describes how to use iKnow ObjectScript queries as data sources for KPIs (key performance indicators) and how to display these KPIs on DeepSee dashboards.

• The Web Services chapter describes using iKnow with Internet data.

• The final two chapters describe advanced topics. Customizing iKnow describes how to create additional iKnow text processing facilities to supplement those supplied with iKnow. Language Identification describes how to work with source texts in more than one language and with texts in languages that need special processing.

• The Domain Parameters appendix provides a comprehensive list of available domain parameters. You can set these parameters to customize a domain, or set them for all domains systemwide.

• The DeepSee Cube Integration (Deprecated Form) appendix describes the deprecated form of integration between iKnow and DeepSee cubes.

For a detailed outline, see the Table of Contents.

For information on using the newer form of iKnow/DeepSee cube integration, see “Using Unstructured Data in Cubes” in Defining DeepSee Models.

For general information, see Using InterSystems Documentation.
1 Conceptual Overview

The iKnow semantic analysis engine is used to analyze unstructured data, data that is written as text in a human language such as English or French. By providing the ability to rapidly access and analyze this type of data, iKnow allows you to work with all of your data. iKnow does not require you to have any prior knowledge of the contents of this data, or even know what language it is written in, so long as it is one of the languages that iKnow supports.

Commonly, unstructured data consists of multiple source texts, often a very large number of texts. A source text is commonly running text divided by punctuation into sentences. A source text can be a file, a record in a SQL result set, or a web source such as a series of blog entries.

1.1 A Simple Use Case

To see how iKnow handles unstructured data, consider the following sentence:

General Motors builds their Chevrolet Volt in the Detroit-Hamtramck assembly plant.

iKnow first divides a text into sentences; it then analyzes each sentence semantically. It does not need to “understand” or look up the words in the sentence. iKnow then indexes the resulting semantic entities from this sentence (normalizing letters to lower case):

[general motors] {builds} their [chevrolet volt] {in} the [detroit-hamtramck assembly plant]

From the initial sentence iKnow has identified the following entities:

- 3 Concepts: [general motors] [chevrolet volt] [detroit-hamtramck assembly plant].
- 2 Relations: {builds} {in}.
- 2 Non-relevants: their the. (Non-relevants are discarded from further iKnow analysis.)

Note: For the purpose of illustration, this example shows each Concept delimited by square brackets and each Relation delimited by curly braces; within iKnow such delimiter characters are not used.

iKnow assigns each entity a unique Id. iKnow also identifies sequences of these entities that follow the pattern Concept-Relation-Concept (CRC). In this example there are two CRCs:

[general motors] {builds} [chevrolet volt] [detroit-hamtramck assembly plant]

iKnow assigns each CRC a unique Id. iKnow also recognizes that this sentence contains a continuous sequence of CRCs (in this case, CRCRC); this is known as a Path. iKnow assigns each Path a unique Id.
iKnow has now identified the sentences in the text, the relevant Entities in each sentence, which Entities are Concepts and which are Relations, which of these form CRC sequences, and which CRC sequences form a Path. By using these semantic units, iKnow can return many types of meaningful information about the contents of the source texts.

## 1.2 What is iKnow?

iKnow provides access to unstructured data by dividing up text into relational and associated entities and producing an index of these entities. It divides a text into sentences, then divides each sentence into a sequence of Concepts and Relations. It performs this operation by identifying the language of the text (for example, English), then applying the corresponding iKnow language model.

- A Relation is a word or group of words that join two Concepts by specifying a relationship between them. iKnow contains a compact language model that is able to identify the Relations in a sentence.
- A Concept is a word or group of words that is associated by a Relation. By determining what is a Relation, iKnow can identify associated Concepts. Thus the iKnow analysis engine can identify Concepts semantically without “understanding” their content.

**Note:** For the purpose of explanation, verbs are commonly Relations and nouns with their associated adjectives are commonly Concepts. However, the linguistic model of Relations and Concepts is significantly more inclusive and more sophisticated than the distinction between verbs and nouns.

Thus iKnow divides a sentence into Concepts (C) and Relations (R). The language model uses a relatively small and fixed dictionary of relationship words and a set of context rules to identify Relations. Anything not identified as a Relation is considered a Concept. (iKnow also identifies non-relevant words, such as “the” and “a”, and discards them from further analysis.)

Relations and Concepts are collectively known as Entities. However, a Relation is almost never meaningful without an associated Concept. For this reason, iKnow entity analysis emphasizes Concepts and sequences that contain Concepts associated by a Relation.

Because iKnow analyzes text using a small and stable language model focused on identifying Relations, iKnow can rapidly index texts containing any subject matter. iKnow does not need to use a dictionary or ontology to identify Concepts.

Once iKnow has identified the Concepts and Relations in each sentence in a text, or (more commonly) in many texts, this information can be used to perform the following types of operations:

- **Smart Indexing:** provides insight into what’s relevant, what’s related, and what’s representative from a large body of unstructured text.
- **Smart Matching:** provides a means to associate entities in the source texts with external items such as lists or dictionaries. These lists can contain words, phrases, or sentences for full (identical) matching and partial matching, and can contain templates for matching by format.

### 1.2.1 What iKnow Isn’t

iKnow is not a search tool. Search tools enable you to locate only those things that you already believe are in the text. iKnow enables you to use the entire contents of the text data, including texts whose content is wholly unknown to you.

iKnow is not a dictionary-based content analyzer. Unlike dictionary-based tools, it does not break up sentences into individual words then attempt to “understand” those words and reconstruct context. iKnow simply identifies Concepts semantically. It does not need to look up these Concepts in a dictionary or ontology. For this reason its language model is compact, stable, and general-purpose; you do not have to specify any general information about the texts being analyzed, or provide
a separate dictionary of nouns. While iKnow can be extended by associating a dictionary or ontology of Concepts, its essential functions do not require one. Thus it does not require the creation, customizing, or periodic updating of a dictionary.

iKnow is not a stemming tool. It does not reduce relations or concepts to stem forms. Instead, it treats each element as a distinct entity, then identifies its degree of similarity to other elements. Caché does provide a set of classes you can use to perform stemming: the %Text package, as described in the InterSystems Class Reference. %Text and %iKnow are wholly independent of each other and are used for different purposes.

1.3 Logical Text Units Identified by iKnow

1.3.1 Entities

An entity is a minimal logical unit of text. It is either a word or a group of words that iKnow logically groups together into either a concept or a relation. Other logical units, such as a telephone number or an email address, are also considered entities (and are treated as concepts).

iKnow normalizes entities so that they may be compared and counted. It removes non-relevant words, such as “the” and “a”. It translates entities into lower case letters. It removes most punctuation and some special characters from entities.

By default, iKnow restricts its analysis of entities to Concepts. By default, Relations are only analyzed because of their role in linking Concepts together. This default can be overridden, as described in the “Limiting by Position” section of the iKnow Queries chapter.

1.3.2 CRCs, CCs, and Paths

Once iKnow divides a sentence into Concepts (C) and Relations (R), it can determine several types of connections between these fundamental entities.

- CRC is a Concept-Relation-Concept sequence. A CRC is handled as a Master Concept - Relation - Slave Concept sequence. Whether an entity is a Master, Relation, or Slave is known as its position. In some cases, a CRC may have an empty string value for one of the sequence members (CR or RC); this can occur, for example, when the Relation of the CRC is an intransitive verb: “Robert slept.”

- CC is a Concept + Concept pair. iKnow retains the position of each Concept, but ignores the Relation between the two Concepts. A CC can either be handled as two associated Concepts, or as a Master Concept/Slave Concept sequence. You can use CC pairs to identify associated Concepts without regard to their master/slave positions or the linking Relation. This is especially useful when determining a network of Concepts — what Concepts have a connection to what other Concepts. You can also use CC pairs as a master/slave sequence.

- A Path is a meaningful sequence of Concepts and Relations through a sentence. For example, in a common path sequence the Slave Concept of one CRC becomes the Master Concept of the next CRC. This results in a path consisting of five entities: C-R-C-R-C. Other meaningful sequences of Concepts and Relations are also treated as paths. A path is always contained within a single sentence. However, a sentence may contain more than one path. This can occur when iKnow identifies a non-continuous Concept and Relation sequence within the sentence. Not all sentences are paths; some very short sentences may not contain the minimum required sequence of Concepts and Relations to qualify as a path. Once identified, the entities that comprise a path sequence are demarcated and normalized, and the path is assigned a unique Id. Often a CRC is not large enough to identify some meaningfully associated entities; paths are especially useful when returning some smaller entity in a wider context.
1.4 Smart Indexing

Smart indexing is the process of translating unstructured text into a relational network of Concepts and Relations. You can index the contents of multiple unstructured texts, then analyze the resulting indexed entities according to user-defined query criteria, such as listing concepts in order of frequency. Each indexed entity can reference its source text, source sentence, and relational entities, such as its position in a CRC sequence. As part of smart indexing, iKnow assigns two values to each indexed concept, specifying the total number of appearances of the concept in the texts (its frequency), and the number of different texts in which the concept appears (its spread).

Once you have performed smart indexing on multiple texts, iKnow can use this information to analyze the source texts. For example, iKnow can perform intelligent content browsing. From any selected iKnow indexed item, you can browse to other items based on the degree of similarity between these items. Intelligent browsing can be performed within a source text or across all indexed source texts.

Once texts are indexed, iKnow can generate summaries of individual texts. The user specifies the length of the summary as a percentage of the original text. iKnow returns a summary text consisting of those sentences of the original text that are most relevant to the whole, based on index statistics. For example, if a text consists of 100 sentences, and the user specifies a 50% summary, iKnow generates a summary text consisting of the 50 most relevant sentences from the original.

1.5 Smart Matching

Once iKnow has indexed a collection of texts, it is possible to match items found in the texts with one or more user-defined match lists and to tag these matches. Smart matching performs high-precision tagging of concepts and phrases based on a semantic understanding of the complete context. Thus matches can occur between similar concepts or phrases, as well as full (identical) matches. Because this tagging is based on finding semantic matches, smart matching does not require any understanding of the text contents.

Once tagged, each appearance of a matched phrase in the texts remains associated with the tag text. These phrases can be matched as a single entity, a CRC, or a path. For example, the user could supply a list of the names of countries, so that each appearance of a country name in the texts is tagged for rapid access. You can build a dictionary of company names that you can match against analyst reports, allowing you to quickly find the latest news about the companies you're interested in. You can create a dictionary in which each appearance of specified medical procedure (phrased in various ways) is matched to a medical diagnostic code.

This dictionary matching is not limited to simple entities, but extends to CRCs and/or paths if the terms in the dictionary span more than one entity themselves. Because iKnow indexes dictionary terms in the same way that it indexes source texts, a dictionary entry may be as long as a sentence. It may be useful to match a dictionary entry sentence against sources to locate similar information.
2

iKnow Implementation

The iKnow semantic analysis engine is a fully-integrated component of Caché. No separate installation is required. No configuration changes are needed.

Your ability to use iKnow is governed by the Caché license. Most standard Caché licenses provide access to iKnow. All Caché licenses that support iKnow provide full and unlimited use of all components of iKnow.

iKnow is not supported on OpenVMS systems.

iKnow is provided as a collection of APIs containing class object methods and properties which may be invoked from Caché ObjectScript programs. APIs are provided to invoke iKnow operations from Caché (API classes). Equivalent APIs are provided to invoke iKnow operations from SQL (QAPI classes) and SOAP web services (WSAPI classes). These APIs are described in the %iKnow package in the InterSystems Class Reference. There is no separate user interface. iKnow is a core Caché technology and therefore does not have application-like interfaces. However, iKnow does provide a few generic, sample output interfaces in the %iKnow.Ui package.

To use iKnow you must define an iKnow domain within a Caché namespace. You can create multiple iKnow domains; a Caché namespace can contain multiple iKnow domains. All iKnow processing occurs within a specified domain. A set of iKnow indexed text sources is created within a domain. All iKnow queries and other text processing must specify the domain in which to access this data.

2.1 A Note on Program Examples

Many of the program examples in this manual begin by deleting a domain (or its data) and then loading all data from the original text files into an empty domain. For the purpose of these examples, this guarantees that the iKnow indexed source data is an exact match with the contents of the file(s) or SQL table(s) from which it was loaded.

This delete/reload methodology is not recommended for real-world applications processing large numbers of text sources. Instead, you should perform the time-consuming load of all sources for a domain once. You can then add or delete individual sources to keep the indexed source data current with the contents of the original text files or SQL tables.

2.2 A Note on %Persistent Object Methods

iKnow support standard %Persistent object methods for creating and deleting object instances such as domains, configurations, and so forth. These %Persistent method names begin with a % character, such as %New(). Use of %Persistent object methods is preferable to using older non-persistent methods, such as Create(). Users are encouraged to use the %Persistent
object methods for new code. Program examples throughout this documentation have been revised to use these preferred %Persistent methods.

Note that the %New() persistent method requires a %Save() method. The older Create() method does not require a separate save operation.

### 2.3 A Note on %iKnow and %SYSTEM.iKnow

Throughout this documentation, all classes referred to are located in the %iKnow package. However, the %SYSTEM.iKnow class also contains a number of iKnow utilities that can be used to simplify coding of common iKnow operations. These utilities are provided as shortcuts; all of the operations performed by %SYSTEM.iKnow class methods can also be performed by %iKnow package APIs. For further details refer to the InterSystems Class Reference.

### 2.4 Space Requirements and iKnow Globals

iKnow globals in a namespace have the following prefix: ^ISC.IK:

- ^ISC.IK.* are the final globals, permanent globals that contain iKnow data. This iKnow data is roughly 20 times the size of the original source texts.
- ^ISC.IKS.* are the staging globals. During data loading these can grow to 16 times the size of the original source texts. Staging globals should be mapped to a non-journaled database. iKnow automatically deletes these staging globals once source loading and processing is completed.
- ^ISC.IKT.* are the temp globals. During data loading these can grow to 4 times the size of the original source texts. Temp globals should be mapped to a non-journaled database. iKnow automatically deletes these temp globals once source loading and processing is completed.
- ^ISC.IKL.* are logging globals. These are optional and their size is negligible.

**CAUTION:** These globals are for internal use only. Under no circumstances should iKnow users attempt to directly interact with iKnow globals.

For example, if you are loading 30Gb of source documents, you will need 600Gb of permanent iKnow data storage. During data loading you will need 1.17Tb of available space, 600Gb of which will be automatically released once iKnow indexing completes.

In addition, the cachetemp subdirectory in the Mgr directory may grow to 4 times the size of the original source texts for the duration of file loading and indexing.

These space requirements apply when you create a domain in Caché 2012.2 and load iKnow data, or when you upgrade a domain created in Caché 2012.1 to support the new Caché 2012.2 features. They do not apply to existing domains created and loaded with iKnow data in Caché 2012.1, or to any data added to a 2012.1 domain in 2012.2. Caché 2012.1 domains have smaller space requirements (and support fewer features), as described in “Upgrading iKnow Data”.

You should increase the size of the Caché global buffer, based on the size of the original source texts. Refer to the “Performance Considerations when Loading Texts” chapter in this manual.
**2.4.1 Batch Load Space Allocation**

Caché allocates 256MB of additional memory for each iKnow job to handle batch loading of source texts. By default, iKnow allocates one job for each processor core on your system. The $$IKPJOBS domain parameter establishes the number of iKnow jobs; generally the default setting gives optimal results. However, it is recommended that the maximum number of iKnow jobs should be either 16 or the number of processor cores, whichever is smaller.

**2.5 Input Data**

iKnow is used to analyze unstructured data. Commonly, this data consists of multiple text sources, often a large number of texts. A text source can be of any type, including the following:

- A file on disk that contains unstructured text data. For example, a .txt file.
- A record in an SQL result set with one or more fields that contain unstructured text data.
- An RSS web feed containing unstructured text data.
- A Caché global containing unstructured text data.

iKnow does not modify the original text sources, nor does it create a copy of these text sources. Instead, iKnow stores its analysis of the original text source as normalized and indexed items, assigning an Id to each item that permits iKnow to reference its source. Separate Ids are assigned to items at each level: source, sentence, path, CRC, and entity.

iKnow supports texts in the following languages: Dutch (nl), English (en), French (fr), German (de), Portuguese (pt), and Spanish (es). You do not have to specify what language your texts contain, nor must all of your texts or all of the sentences in an individual text be in the same language. iKnow can automatically identify the language of each sentence of each text and applies the appropriate language model to that sentence. You can define an iKnow configuration that specifies the language(s) that your texts contain, and whether or not to perform automatic language identification. Use of an iKnow configuration can significantly improve iKnow performance.

You do not have to specify a genera for text content (such as medical notes or newspaper articles); iKnow automatically handles texts of any content type.

**2.5.1 File Formats**

iKnow accepts source files of any format and with any extension (suffix). By default, iKnow assumes that a source text file consists of unformatted text (for example, a .txt file). It will process source files with other formatting (for example, .rtf, .doc) but may treat some formatting elements as text. To avoid this, you can either convert your source files to .txt files and load these .txt files, or you can create an iKnow converter to remove formatting from source text during iKnow loading.

You specify the list of file extensions as a Lister parameter. Only files with these extensions will be loaded. For example, this list of file extensions can be specified as an AddListToBatch() method parameter.

**2.5.2 SQL Record Format**

iKnow accepts records from an SQL result set as sources. iKnow generates a unique integer value for each record as the iKnow source Id. iKnow allows you to specify an SQL field containing unique values which iKnow uses to construct the external Id for the source records. Note that the iKnow source Id is assigned by iKnow, it is not the external Id, though frequently both are unique integers. Commonly, the iKnow source text is taken from only some of the fields of the result set record, often from a single field containing unstructured text data. It can ignore the other fields in the record, or use their values as metadata to filter (include or exclude) or to annotate the source.
2.5.3 Text Normalization

iKnow maintains links to the original source text. This enables it to return a sentence with its original capitalization, punctuation, and so forth. Within iKnow, normalization operations are performed on entities to facilitate matching:

- Capitalization is ignored. iKnow matching is not case-sensitive. Entity values are returned in all lowercase letters.
- Extra spaces are ignored. iKnow treats all words as being separated by a single space.
- Multiple periods (…) are reduced to a single period, which iKnow treats as a sentence termination character.
- Most punctuation is used by the language model to identify sentences, concepts and relations, then discarded from further analysis. Punctuation is generally not preserved within entities. Most punctuation is only preserved in an entity when there are no spaces before or after the punctuation mark. However, the slash (/), and at sign (@) are preserved in an entity with or without surrounding spaces.

2.5.4 User-defined Source Normalization

The user can define several types of tools for source normalization:

- **Converters** process source text to remove formatting tags and other non-text content during loading.
- **UserDictionary** enables the user to specify how to rewrite or use specific input text content elements during loading. For example, UserDictionary can specify substitutions for known abbreviations and acronyms. It is commonly used to standardize text by eliminating variants and synonyms. It can also be used to specify text-specific exceptions to standard iKnow processing of punctuation.

2.6 Output Structures

iKnow creates global structures to store the results of its operations. These global structures are intended for use by iKnow class APIs only. They are not user-visible and should not be modified by the user.

iKnow indexed data is stored as Caché list structures. Each iKnow list structure contains a generated ID for that item, a unique integer value. iKnow entities can be accessed either by value or by integer ID.

iKnow preserves the relationships amongst indexed entities, so that each entity can reference the entities related to it, the path of that sequence of entities, the original sentence that contains that path, and the location of that sentence within its source text. The original source text is always available for access from iKnow. iKnow operations do not ever change the original source text.

2.7 Constants

iKnow defines constant values in the %IKPublic.inc file. After specifying this include file, you can invoke these constants using the $$ macro invocation, as shown in the following example:

```
#Include %IKPublic
WRITE "The $$FILTERONLY constant=", $$FILTERONLY
```

These constants include domain parameter names, query parameter values, and other constants.
2.8 Error Codes

The General Error Codes 8000-8099 are reserved for use by iKnow. For further details, refer to General Error Messages in the Caché Error Reference.

2.9 Upgrading iKnow Data

Each version of the iKnow data structures is assigned a system version number. Each iKnow domain is assigned a Version property value. All new domains are created with the same Version property as the current system version. Therefore, these two integer values are usually the same.

However, when you upgrade to a newer version of Caché, the iKnow data structures system version may (or may not) increment. If the iKnow data structures system version has incremented, it does not match the Version property of existing domains.

You can use the GetCurrentSystemVersion() method of the %iKnow.Domain class to determine the iKnow data structures system version for the current Caché instance. You can use the GetAllDomains query to list all domains with their domain Version numbers, as shown in the Listing All Domains section of the “Setting Up the iKnow Environment” chapter.

If the iKnow data structures system version does not match the domain Version, these older iKnow domains cannot take advantage of the new iKnow features and performance improvements introduced with this new system version. Older domains will remain operational, but cannot take advantage of new iKnow data structure features until you upgrade the domain. Upgrading a domain increments its Version property. This upgrade operation requires the automatic re-indexing of the domain data. It does not require access to the original source texts. Each domain must be upgraded individually.

Note: Upgrading a domain from Caché 2012.1 (iKnow data structures version 1) to Caché 2012.2 (iKnow data structures version 2) substantially increases the space requirements for permanent and temporary globals. The space requirements for permanent globals increase from 8.7 times to 20 times the size of the original source texts. The space requirements for temporary globals increase from 10.7 times to 20 times the size of the original source texts. The space requirements for the cachetemp directory increase from 1.25 times to 4 times the size of the original source texts.

For this reason, you should only upgrade a large iKnow domain from iKnow data structures version 1 to version 2 if you have the available space and need the new iKnow data structure features (such as semantic dominance and semantic proximity).

To upgrade a domain, use the UpgradeDomain() method of the %iKnow.Utils.UpgradeUtils class. Further details are provided in the InterSystems Class Reference documentation.

Note that the re-indexing that occurs when you upgrade a domain changes the domain Id, but does not change the domain name. Thus upgrading a domain may, in some cases, require changes to programs that reference the domain by a specific domain Id integer. For this reason, coding practices that reference a domain by a literal domain Id number (rather than the domain’s Id property value) should be avoided.
3

Setting Up the iKnow Environment

Before using iKnow on source data, you must create an instance of the objects that define the iKnow environment:

- **Domain**: establishes the logical space for iKnow operations. Specifying a domain is mandatory.
- **Configuration**: establishes the language environment for source document content. Specifying a configuration is optional. If not specified, iKnow provides a default.

You can create multiple instances of domains and configurations. These environment objects are independent of one another, and are independent of any specified set of source data.

3.1 iKnow Domains

All iKnow operations occur within a Domain. A domain is an iKnow defined unit within a Caché namespace. All source data to be used by iKnow is listed and loaded into a domain. A Caché namespace can contain multiple iKnow domains.

3.1.1 Defining a Domain

To define a new domain using class methods, invoke the `%iKnow.Domain.%New()` persistent method, supplying the domain name as the method parameter. A domain name can be any valid string; domain names are not case-sensitive. The name you assign to this domain must be unique for the current namespace. This method returns a domain object reference (oref) which is unique for all namespaces of the Caché instance. You must then save this instance using the `%Save()` method to make it persistent. The domain Id property (an integer value) is not defined until you save the instance as a persistent object, as shown in the following example:

CreateDomain
SET domOref=##class(%iKnow.Domain).%New("FirstExampleDomain")
WRITE "Id before save: ",domOref.Id,!
DO domOref.%Save()
WRITE "Id after save: ",domOref.Id,!

CleanUp
DO ##class(%iKnow.Domain).%DeleteId(domOref.Id)
WRITE "All done"

There are two ways to create a domain if it doesn’t exist, or open the domain if it does exist:

- Use **Exists()** to determine if the domain already exists. If the domain exists, use **Open()** to open it. If the domain doesn’t exist, use **%New()** to create it.
- Use **GetOrCreateId()** to create the domain if it doesn’t exist, or open the domain if it does exist.
The following example checks whether a domain exists. If the domain doesn’t exist, the program creates it. If the domain does exist, the program opens it. For the purpose of demonstration, this program then randomly either deletes or doesn’t delete the domain.

```
DomainCreateOrOpen
SET domn="mydomain"
IF (##class(%iKnow.Domain).Exists(domn))
  { WRITE "The ",domn," domain already exists!
    SET domo=##class(%iKnow.Domain).Open(domn)
    SET domId=domo.Id
  }
ELSE {
  SET domo=##class(%iKnow.Domain).%New(domn)
  DO domo.%Save()
  SET domId=domo.Id
  WRITE "Created the ",domn," domain!
  WRITE "with domain ID ",domId,!
}
ContainsData
SET x=domo.IsEmpty()
IF x=1 {WRITE "Domain ",domn," contains no data",!}
ELSE {WRITE "Domain ",domn," contains data",!}
CleanupForNextTime
SET rnd=$RANDOM(2)
IF rnd {
  SET stat=##class(%iKnow.Domain).%DeleteId(domId)
  IF stat {WRITE "Deleted the ",domn," domain" }
  ELSE { WRITE "Domain delete error:",stat }
}
ELSE {WRITE "No delete this time" }
```

The following example uses the `GetOrCreateId()` shorthand method to create a domain if it doesn’t exist, or open the domain if it does exist. For the purpose of demonstration, this program then randomly either deletes or doesn’t delete the domain.

```
DomainGetOrCreate
SET domn="mydomain"
SET domId=##class(%iKnow.Domain).GetOrCreateId(domn)
SET domoref=##class(%iKnow.Domain).%OpenId(domId)
WRITE "The ",domn," domain with domain ID ",domId,!
ContainsData
SET x=domoref.IsEmpty()
IF x=1 {WRITE "Domain ",domn," contains no data",!}
ELSE {WRITE "Domain ",domn," contains data",!}
CleanupForNextTime
SET rnd=$RANDOM(2)
IF rnd {
  SET stat=##class(%iKnow.Domain).%DeleteId(domId)
  IF stat {WRITE "Deleted the ",domn," domain" }
  ELSE { WRITE "Domain delete error:",stat }
}
ELSE {WRITE "No delete this time" }
```

The `%iKnow.Domain` class methods that create or open a domain are provided with an output `%Status` parameter. This parameter is set when the current system does not have license access to iKnow, and thus cannot create or open an iKnow domain.

### 3.1.2 Setting Domain Parameters

Domain parameters govern the behavior of a wide variety of iKnow operations. The specific parameters are described where applicable throughout this manual. For a list of available domain parameters, refer to the Appendix “Domain Parameters”.

**Note:** In the examples that follow, domain parameters are referenced by their macro equivalent (for example, `$$$IKP-FULLMATCHONLY`), not their parameter name (For example, FullMatchOnly). The recommended programming practice is to use these %IKPublic macros rather than the parameter names.

All domain parameters take a default value. Commonly, iKnow will give optimal results without specifically setting any domain parameters. iKnow determines the value for each parameter as follows:
1. If you have specified a parameter value for the current domain, that value is used. Note that some parameters can only be set before loading data into a domain, while others can be set at any time.

2. If you have specified a systemwide parameter value, that value is used as a default for all domains, except for a domain where a domain-specific value has been set.

3. If you have not specified a value for a parameter at either the domain level or the system level, iKnow uses its default value for that parameter.

### 3.1.2.1 Setting Parameters for the Current Domain

Once you have created a domain, you can set domain parameters for this specific domain using the `SetParameter()` instance method. `SetParameter()` returns a status indicating whether the parameter specified is valid and was set. `GetParameter()` returns the parameter value and the level at which the parameter was set (DEFAULT, DOMAIN, or SYSTEM). Note that `GetParameter()` does not check the validity of a parameter name; it returns DEFAULT for any parameter name it cannot identify as being set at the domain or system level.

The following example gets the default for the SortField domain parameter, sets this parameter for the current domain, then gets the value you set and the level at which it was set (DOMAIN):

```plaintext
#Include %IKPublic
DomainCreate  
SET domn="paramdomain"  
SET domo=##class(%iKnow.Domain).%New(domn)  
WRITE "Created the ",domn," domain",!  
DO domo.%Save()  
DomainParameters  
SET sfval=domo.GetParameter($$$IKPSORTFIELD,.sf)  
WRITE "SortField before SET=",sfval," ",sf,!  
IF sfval=0 {WRITE "changing SortByFrequency to SortBySpread",!  
SET stat=domo.SetParameter($$$IKPSORTFIELD,1)  
IF stat=0 {WRITE "SetParameter failed" QUIT} }  
WRITE "SortField after SET=",domo.GetParameter($$$IKPSORTFIELD,.str)," ",str,!!
```

### 3.1.2.2 Setting Parameters Systemwide

You can set domain parameters for all domains systemwide using the `SetSystemParameter()` method. A parameter set using this method immediately becomes the default parameter value for all existing and subsequently created domains in all namespaces. This systemwide default is overriden for an individual domain using the `SetParameter()` instance method.

Note: The SortField and Jobs domain parameters are exceptions. Setting these parameters at the system level has no effect on the domain settings.

You can determine if a domain parameter has been established as the system default using the `GetSystemParameter()` method. The initial value for a systemwide parameter is always the null string (no default).

If you wish to remove a systemwide default setting for a domain parameter, use the `UnsetSystemParameter()` method. Once a systemwide parameter setting has been established, you must unset it before you can set it to a new value.

The following example establishes a FullMatchOnly systemwide parameter value. If no systemwide default has been established, the program sets this systemwide parameter. If a systemwide default has been established, the program unsets this systemwide parameter, then sets it.

```plaintext
#Include %IKPublic
SystemwideParameterSet  
/* Initial set */  
SET stat=##class(%iKnow.Domain).SetSystemParameter($$$IKPFULLMATCHONLY,1)  
IF stat=1 {  
WRITE "FullMatchOnly set systemwide to: "  
WRITE ##class(%iKnow.Domain).GetSystemParameter($$$IKPFULLMATCHONLY),!  
QUIT }
```
ELSE {
   /* Unset and Reset */
   SET stat=##class(%iKnow.Domain).UnsetSystemParameter($$$IKPFULLMATCHONLY)
   IF stat=1 {
      SET stat=##class(%iKnow.Domain).SetSystemParameter($$$IKPFULLMATCHONLY,1)
      IF stat=1 {
         WRITE "FullMatchOnly was unset systemwide",!," then set to: "
         WRITE ##class(%iKnow.Domain).GetSystemParameter($$$IKPFULLMATCHONLY),!!
         GOTO CleanUpForNextTime }
      ELSE {WRITE "System Parameter set error",stat,!!}
   } ELSE {WRITE "System Parameter set error",stat,!!}
   CleanUpForNextTime
   SET stat=##class(%iKnow.Domain).UnsetSystemParameter($$$IKPFULLMATCHONLY)
   IF stat '=1 {WRITE "Unset error status:",stat}
}
CleanUpForNextTime

The following example shows that setting a systemwide parameter value immediately sets the parameter value for all domains. After setting a systemwide parameter value, you can override this value for individual domains:

#include %IKPublic
SystemwideParameterUnset
SET stat=##class(%iKnow.Domain).UnsetSystemParameter($$$IKPFULLMATCHONLY)
WRITE "Systemwide setting
FullMatchOnly=",##class(%iKnow.Domain).GetSystemParameter($$$IKPFULLMATCHONLY),!!

Domain1Create
SET domn1="mysysdomain1"
SET domo1=##class(%iKnow.Domain).%New(domn1)
DO domo1.%Save()
   WRITE "Created the ",domn1," domain ",domo1.Id,
   WRITE "FullMatchOnly=",domo1.GetParameter($$$IKPFULLMATCHONLY,.str)," ",str,!!
SystemwideParameterSet
SET stat=##class(%iKnow.Domain).SetSystemParameter($$$IKPFULLMATCHONLY,1)
IF stat=0 {WRITE "SetSystemParameter failed" QUIT}
WRITE "Set systemwide FullMatchOnly=",##class(%iKnow.Domain).GetSystemParameter($$$IKPFULLMATCHONLY),!!

Domain2Create
SET domn2="mysysdomain2"
SET domo2=##class(%iKnow.Domain).%New(domn2)
DO domo2.%Save()
   WRITE "Created the ",domn2," domain ",domo2.Id,
   WRITE "FullMatchOnly=",domo2.GetParameter($$$IKPFULLMATCHONLY,.str)," ",str,!!
DomainParameters
WRITE "New domain ",domo2.Id," FullMatchOnly=",domo2.GetParameter($$$IKPFULLMATCHONLY,.str)," ",str,!
WRITE "Existing domain ",domo1.Id," FullMatchOnly=",domo1.GetParameter($$$IKPFULLMATCHONLY,.str)," ",str,!!

OverrideForOneDomain
SET stat=domo1.SetParameter($$$IKPFULLMATCHONLY,0)
IF stat=0 {WRITE "SetParameter failed" QUIT}
WRITE "Domain override FullMatchOnly=",domo1.GetParameter($$$IKPFULLMATCHONLY,.str)," ",str,!

CleanupForNextTime
SET stat=##class(%iKnow.Domain).%DeleteId(dom1Id)
SET stat=##class(%iKnow.Domain).%DeleteId(dom2Id)
SET stat=##class(%iKnow.Domain).UnsetSystemParameter($$$IKPFULLMATCHONLY)

3.1.3 Assigning to a Domain

Once you have created a domain and (optionally) specified its domain parameters, you can assign various components to that domain:

• **Source Data:** After creating a domain, you commonly will load a number (usually a large number) of text sources into a domain; this generates iKnow indexed data within that domain. Loading text sources is a required precondition for most iKnow operations. A variety of text sources are supported, including files, SQL fields, and text strings. After iKnow has indexed a data source, the original data source can be removed without affecting iKnow processing. Changing a data source has no effect on iKnow processing, unless you re-load that data source to update the iKnow indexed data in the domain.

• **Filters:** After creating a domain, you can optionally create one or more filters for that domain. A filter specifies criteria used to exclude some of the loaded sources from a query. Thus a filter allows you to perform iKnow operations on a subset of the data loaded in the domain.
• **Metadata:** After creating a domain, you can optionally specify one or more metadata fields that you can use as criteria for filtering sources. A metadata field is data associated with a source that is not iKnow indexed data. For example, the date and time that a text source was loaded is a metadata field for that source. Metadata fields must be defined *before* loading text sources into a domain.

• **Black Lists:** After creating a domain, you can optionally create one or more black lists for that domain. A black list is a list of entities (such as words or phrases) that you do *not* want a query to return. Thus a black list allows you to perform iKnow operations that ignore specific data entities in data sources loaded in the domain.

• **Dictionaries:** After creating a domain, you can optionally create one or more dictionaries for that domain. A dictionary contains entities that are used to match the iKnow indexed data.

These components are defined using various iKnow classes and methods. Metadata fields must be defined before loading sources. Filters, black lists, and dictionaries can be defined or modified at any time.

### 3.1.4 Deleting All Data from a Domain

Once a domain has been used to list and load data, deleting or changing an original source file has no effect on the source data already listed and loaded in iKnow. You must explicitly add or delete a text file to the set of indexed sources.

The `%DeleteId()` persistent method deletes a domain and all source data that has been listed and loaded in that domain. You can use the `DropData()` method to delete all source data that has been listed and loaded in a specified domain without deleting the domain itself. Either method deletes all indexed source data, allowing you to start over with a new set of data sources.

The following example demonstrates deleting the data from a domain. If the named domain doesn’t exist, the program creates the domain. If the named domain does exist, the program opens the domain and deletes any pre-existing data in the domain.

```plaintext
DomainCreateOrOpen
  SET dname="mytestdomain"
  IF _##class(%iKnow.Domain).Exists(dname)
    { WRITE "The ",dname," domain already exists",!
      SET domoref=_##class(%iKnow.Domain).Open(dname)
      GOTO DeleteData }
  ELSE
    { WRITE "The ",dname," domain does not exist",!
      SET domoref=_##class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      WRITE "Created the ",dname," domain with domain ID ",domoref.Id,!
      GOTO RestOfProgram }
DeleteData
  SET stat=domoref.DropData()
  IF stat { WRITE "Deleted the data from the ",dname," domain",!
    GOTO RestOfProgram } ELSE { WRITE "DropData error",!
    QUIT}
RestOfProgram
  WRITE "The ",dname," domain contains no data"
```

### 3.1.5 Listing All Domains

You can use the `GetAllDomains` query to list all current domains. This is shown in the following example:

```plaintext
DO ##class(%ResultSet).RunQuery("%iKnow.Domain","GetAllDomains")
WRITE !,"Domains in all namespaces"
```

Each domain is listed on a separate line, using the following format: `domainId:domainName:namespace:version`.

The `Version` property is an integer that shows what version of iKnow data structure was used when the domain was created. The data structure Version property changes when a release contains a change to the iKnow data structures. Therefore, a new version of Caché or the introduction of new iKnow features may not change the version number. If the `Version` property
value is not the current iKnow system version, you should upgrade the domain to take advantage of the latest features of iKnow. See Upgrading iKnow Data in the “iKnow Implementation” chapter.

By default, GetAllDomains lists all the current domains for all namespaces. You can specify a boolean argument to limit the listing of domains to the current namespace, as shown in the following example:

```plaintext
ZNSPACE "USER"
DO ##class(%ResultSet).RunQuery("%iKnow.Domain","GetAllDomains",1)
WRITE !,"Domains in the USER namespace",!!
ZNSPACE "SAMPLES"
DO ##class(%ResultSet).RunQuery("%iKnow.Domain","GetAllDomains",1)
WRITE !,"Domains in the SAMPLES namespace",!!
```

A boolean value of 1 limits listing to domains in the current namespace. A boolean value of 0 (the default) lists all domains in all namespaces. (Note: listed Version property values may not be correct for domains other than the current domain.)

You can also list all domains in the current namespace using:

```plaintext
DO ##class(%SYSTEM.iKnow).ListDomains()
```

This method lists the domain Ids, domain names, number of sources, and the domain version number.

### 3.1.6 Copying a Domain

You can copy an existing domain to a new domain in the current namespace by using the CopyDomain() method of the %iKnow.Utils.CopyUtils class. The CopyDomain() method copies a domain definition to a new domain, assigning a unique domain name and domain Id; the existing domain is unchanged. If the new domain does not exist, this method creates a new domain. By default, this method copies the domain parameter settings and assigned domain components from the existing domain to the copy, if these components are present.

By default, the CopyDomain() method copies the source data from the existing domain to the copy. However, if source data copying is requested and no source data is present in the existing domain, the CopyDomain() operation fails.

The following example copies a the domain named “mydomain” and its parameter settings and source data to a new domain named “mydupdomain”. Because “mydomain” contains no source data, the 3rd argument (which specifies whether to copy source data) is set to 0:

```plaintext
DomainMustExistToBeCopied
SET olddom="mydomain"_PIECE($H,:,2)
SET domo=##class(%iKnow.Domain).%New(olddom)
DO domo.%Save()
  IF (!##class(%iKnow.Domain).Exists(olddom))
    {WRITE "Old domain exists, proceed with copy",!!}
  ELSE {WRITE "Old domain does not exist" QUIT}
CopyDomain
  SET newdom="mydupdomain"
  IF (!##class(%iKnow.Domain).Exists(newdom))
    {WRITE "Domain copy overwriting domain ",newdom,!!}
  ELSE {WRITE "Domain copy creating domain ",newdom,!!}
  SET stat=##class(%iKnow.Utils.CopyUtils).CopyDomain(olddom,newdom,0)
  IF stat=1 {WRITE ","olddom", to ""newdom," copying all assignments",!!}
  ELSE {WRITE "Domain copy failed with status ",stat,!!}
Cleanup
  SET stat=##class(%iKnow.Domain).%DeleteId(domo.Id)
  WRITE "Deleted the old domain",!
  IF $RANDOM(2) {
    SET newId=##class(%iKnow.Domain).GetOrCreateId("mydupdomain")
    SET stat=##class(%iKnow.Domain).%DeleteId(newId)
    WRITE "Deleted the new domain" }
  ELSE {WRITE "No new domain delete this time" }
```

The CopyDomain() method allows you to quickly copy all of the domain settings, source data, and assigned components of an existing domain to a new domain. It provides boolean options for all-or-nothing copying of assigned components. Other methods in the %iKnow.Utils.CopyUtils class provide greater control in specifying which assigned components to copy from one existing domain to another.
3.2 iKnow Configurations

An iKnow configuration specifies behavior for handling source documents. It is only used during the source data loading operation. A configuration is specific to its namespace; you can create multiple configurations within a namespace. iKnow assigns each configuration in a namespace a configuration Id, a unique integer. Configuration Id values are not reused. You can apply the same configuration to different domains and source text loads. Defining or using an iKnow configuration is optional; if you don’t specify a configuration, iKnow uses the property defaults.

3.2.1 Defining a Configuration

You can create a configuration using the %New() persistent method.

You can determine if an iKnow configuration with that name already exists by invoking the Exists() method. If the configuration exists, you can open it using the Open() method, as shown in the following example:

```
IF ##class(%iKnow.Configuration).Exists("EnFr") {
    SET cfg=##class(%iKnow.Configuration).Open("EnFr")
} ELSE {
    SET cfg=##class(%iKnow.Configuration).%New("EnFr",1,$LB("en","fr"))
    DO cfg.%Save()
}
```

3.2.2 Setting Configuration Properties

A configuration defines the following properties:

- **Name**: A configuration name can be any valid string; configuration names are not case-sensitive. The name you assign to this configuration must be unique for the current namespace.

- **DetectLanguage**: A boolean value that specifies whether to use automatic language identification if more than one language is specified in the Languages property. Because this option may have a significant effect on performance it should not be set unless needed. The default is 0 (do not use automatic language identification).

- **Languages**: What language(s) the source documents contain, and therefore which languages to test for and which language models to apply. The available options are Dutch (nl), English (en), French (fr), German (de), Portuguese (pt), and Spanish (es). The default is English (en). This property is specified as a Cache list of strings (using $LISTBUILD).

- **UserDictionary**: The file path location of a user-defined dictionary that iKnow applies to the source text entities during the load operation. This property is optional; the default is the null string.

- **Summarize**: A boolean value that specifies whether to store summary information when loading source texts. If set to 1, source information is generated that iKnow requires to generate summaries of the loaded source texts. If set to 0, no summaries can be generated for the sources processed with this Configuration object. Setting this option to 1 is generally recommended. The default is 1.

- **MaxConceptLength**: An integer configuring the largest number of words in a concept. Multiple words joined by hyphens are counted as a single word. This property is optional and should only be changed from the default in exceptional circumstances. The default is 0, which sets MaxConceptLength to the default for the current language: maximum of 6 words in a concept for all languages except French; maximum of 8 words in a concept for French. When multiple languages are specified, the MaxConceptLength default is the default for the first language specified.

All configuration properties (except the Name) are assigned default values. You can get or set a configuration property by using property dispatch:

```
IF cfgOref.DetectLanguage=0 {
    SET cfgOref.DetectLanguage=1
    DO cfgOref.%Save()
}
```
Note that you must first %Save() the newly created configuration before you can change its properties using property dispatch, and then you must %Save() the configuration after changing the property values.

The following example creates a configuration that supports English and French with automatic language identification. It then changes the configuration to support English and Spanish:

```
OpenOrCreateConfiguration
SET myconfig="Bilingual"
IF ##class(%iKnow.Configuration).Exists(myconfig) {
  SET cfg=##class(%iKnow.Configuration).Open(myconfig)
  WRITE "Opened existing configuration ",myconfig,! }
ELSE { SET cfg=##class(%iKnow.Configuration).%New(myconfig,1,$LB("en","fr"))
  DO cfg.%Save()
  WRITE "Created new configuration ",myconfig,! }
GetLanguages
WRITE "that supports ",$LISTTOSTRING(cfg.Languages),!
SetConfigParameters
SET cfg.Languages=$LISTBUILD("en","sp")
DO cfg.%Save()
WRITE "changed ",myconfig," to support ",$LISTTOSTRING(cfg.Languages),!
CleanUpForNextTime
SET rnd=$RANDOM(2)
IF rnd {
  SET stat=##class(%iKnow.Configuration).%DeleteId(cfg.Id)
  IF stat {WRITE "Deleted the ",myconfig," configuration" }
}
ELSE {WRITE "No delete this time",! }
```

For a description of using multiple languages and automatic language identification, refer to the “Language Identification” chapter of this manual.

### 3.2.3 Defining a UserDictionary

The UserDictionary property specifies a user-defined file applied to the source texts during listing. For example, if UserDictionary contains the substitution of “Doctor” for the abbreviation “Dr.” every occurrence of “Dr.” is replaced by the word “Doctor” in the data indexed by iKnow. The original source file is not changed, but all representations of the source text within iKnow contain this substitution. Unlike all other components of iKnow, UserDictionary changes the source content before listing and loading.

You can use the UserDictionary to substitute one term for another, to expand acronyms and abbreviations, or to avoid or cause a sentence break.

To specify a UserDictionary, you supply the full pathname as the 4th argument in the configuration %New() method:

```
SET cfg=##class(%iKnow.Configuration).%New(myconfig,0,$LISTBUILD("en"),"C:\temp\udict.txt",1)
```

Specifying a UserDictionary is optional. A UserDictionary is specified for a configuration, but exists independent of any specific configuration or domain. The same UserDictionary can be specified for multiple configurations.

**Note:** You cannot modify an existing configuration; a %New() does not delete/replace an existing configuration.

Therefore, to add a UserDictionary to an existing configuration you must explicitly delete then re-create the named configuration. Alternatively, you can create a new configuration with a new configuration name.

The UserDictionary is applied to sources when the sources are listed; already indexed sources are not affected by changes to UserDictionary.

You specify substitution pairs in the UserDictionary file (in this case, udict.txt). Each substitution pair is a separate line with the following format: oldstring,newstring. Note that substitution is string substitution, and that pairs are case sensitive. A UserDictionary file must be in UTF-8 format encoding. The following is a sample UserDictionary file:

```
Mr.,Mister
Dr.,Doctor
Fr.,Fr
UK.,United Kingdom
```

The slash character provides additional formatting options:
These are shown in the following sample UserDictionary:

\UK,United Kingdom
\+,plus
Fr.,noend
\STOP,end

### 3.2.4 Using a Configuration

You can apply a defined configuration in any of the following ways:

- Defining the **DefaultConfig** domain parameter.
- Specifying the configuration as the first argument of the **Init()** instance method to initialize the Lister instance and override the configuration default.
- Invoking **%iKnow.Source.Lister.SetConfig()**.
- Specifying the configuration as an argument of the **loader.ProcessBatch()** method.

The following example demonstrates a configuration that sets MaxConceptLength=2. The program defines the configuration, then specifies it in the Lister **Init()** instance method. The example uses the **GetTop()** query to show the application of MaxConceptLength. **GetTop()** returns the top concepts, none of which are more than two words in length. (To shorten the query data set, this example sets the `$$IKPMINTOPCONCEPTLENGTH` domain parameter to 13, this prevents **GetTop()** from listing concepts of less that 14 characters.)

```plaintext
#include %IKPublic
OpenOrCreateConfiguration
SET myconfig="ShortConcepts"
SET stat=##class(%iKnow.Configuration).Exists(myconfig)
IF stat {
    SET cfg=##class(%iKnow.Configuration).Open(myconfig)
    WRITE "Opened existing configuration ",myconfig,! 
    SET cfgId=cfg.Id
} 
ELSE {
    SET cfg=##class(%iKnow.Configuration).%New(myconfig,0,$LISTBUILD("en"),",",1,2)
    IF ##class(%iKnow.Configuration).Exists(myconfig)
    {WRITE "Configuration ",myconfig," now exists",!
    DO cfg.%Save()
    SET cfgId=cfg.Id
    WRITE "with configuration ID ",cfgId,! }
    DomainCreateOrOpen
    SET dname="xyzdomain"
    IF (##class(%iKnow.Domain).Exists(dname))
    { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DomainSetup }
    ELSE {
    SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    DO DomainSetup }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO DomainSetup }
ELSE { WRITE "DropData error ",##class(%System.Status).DisplayError(stat) QUIT}
DomainSetup
SET domId=domoref.Id
SET stat=domoref.setParameter(##$IKPMINTOPCONCEPTLENGTH,13)
IF stat=0 { WRITE "setParameter failed" QUIT}
ListerAndLoader
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init(myconfig)
```

Using iKnow 21
3.2.5 Listing All Configurations

You can use the `GetAllConfigurations` query to list all defined configurations in the current namespace. This is shown in the following example:

```
WRITE "The current namespace is: ", $NAMESPACE, !
WRITE "It contains the following configurations: ", !
DO ##class(%ResultSet).RunQuery("%iKnow.Configuration","GetAllConfigurations")
```

Each configuration is listed on a separate line, listing the configuration Id followed by the configuration parameter values. Listed values are separated by colons. If the configuration is defined with a list of supported languages, `GetAllConfigurations` displays these language abbreviations separated by commas.
Loading Text Data into an iKnow Domain

To make text data available for iKnow processing, you invoke a Loader and a Lister. The Loader supervises iKnow processing of text sources, using the Lister and a Processor. The Lister identifies the text sources to be used by the Loader. iKnow provides a variety of Listers for different types of source text data. Each Lister, by default, automatically invokes the corresponding Processor with default parameters. There is one Loader used for data sources of all types.

Note that the Loader and Lister objects can be created in any order, but both must have been created before you invoke the Lister `AddListToBatch()` instance method and then the Loader `ProcessBatch()` instance method (or other equivalent Lister and Loader methods).

4.1 Loader

The Loader (％iKnow.Source.Loader) is the main class coordinating the loading process. You must create a new loader object for the domain. To create a loader object:

```plaintext
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
```

After creating a loader and a lister, you issue instance methods to list and process the sources. For example, when performing a batch load you issue the Lister `AddListToBatch()` instance method to list the text sources. You then issue the Loader `ProcessBatch()` instance method to process the listed sources. This Loader method calls the Lister to scan the locations marked by `AddListToBatch()`, then calls the Processor to read those documents and push them to the iKnow engine and finally, it invokes the `^%iKnow.BuildGlobals` routine to process the staging globals loaded by the iKnow engine.

4.1.1 Loader Error Logging

If a load operation completes, but encounters errors in loading one or more sources, these errors are recorded in an error log. Errors of varying severity can be retrieved using the `GetErrors()`, `GetWarnings()`, and `GetFailed()` methods. For example, a failed load error (`GetFailed()`) occurs if you attempt to load a source file that has no contents. A warning load error (`GetWarnings()`) occurs if there is an error in the source metadata.

You can use the `ClearLogs()` method to clear the error log of error messages at any or all of these severity levels.

4.1.2 Loader Reset()

If a load operation didn't complete in an expected fashion and you want to start from scratch, you should invoke the `Reset()` method for the loader instance, as follows:

```plaintext
DO myloader.Reset()
```
4.2 Lister

The Lister identifies text files, records, or other sources of unstructured data you wish iKnow to index. That is, all text that will eventually end up as a Source in the domain. The unit of content in iKnow is a Source, which can represent any unit of text you wish to analyze, such as a text file, a record in a SQL table, an RSS posting, or other text source.

Usually a Source is a text containing multiple sentences. However, a source can contain content of any type. For example, a file containing the number 123 is treated as a Source containing one sentence. A file with no contents is not listed as a Source.

All listers are found in class %iKnow.Source.Lister and have their own specific type of sources they can scan. For example, the subclass %iKnow.Source.File.Lister scans a file system and the subclass %iKnow.Source.RSS.Lister scans RSS web feeds, such as blog postings, in XML file format. iKnow provides seven listers for different types of sources. You can also create your own custom lister.

Most text sources require a Lister. However, text that is directly specified as a string does not require a Lister.

Through the AddListToBatch() method you can instruct the Lister to look into a specific directory, SQL table, or RSS feed for Sources. The lister parameters depend on the actual Lister class.

4.2.1 Initializing a Lister

You can create a Lister instance for a domain using the %New() method for that type of lister, supplying the domain Id. The following example creates two listers within the specified domain:

```
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SET flister=##class(%iKnow.Source.File.Lister).%New(domId)
WRITE flister,
SET rlister=##class(%iKnow.Source.RSS.Lister).%New(domId)
WRITE rlister
```

Each lister automatically invokes the corresponding processor, as follows:

- The Domain.Lister invokes the Domain.Processor.
- All other Listers invokes the Temp.Processor. The %iKnow.Source.Temp.Processor has that name because it processes temporary globals that are automatically created and deleted by iKnow during the loading process.

Each processor has default processor parameters, which are appropriate for most iKnow sources. Therefore, in most cases, you do not need to specify a processor or processor parameters. If you do not specify a processor, iKnow uses the default processor, as shown by the DefaultProcessor() method.

4.2.2 Overriding Lister Instance Defaults

In most cases, the lister instance defaults are appropriate for the processing of your iKnow sources.

If you wish to overriding lister instance defaults for Configuration, Processor, or Converter objects, you can, optionally, use the Init() instance method to initialize the Lister instance. If you omit Init() the defaults are used.

The complete Lister initialization is as follows:

```
Init(config,processor,processorparams,converter,converterparams)
```

To specify the default for any of these items, specify the empty string ("") as the Init() parameter value.
You can also initialize these objects separately using the `SetConfig()`, `SetProcessor()`, and `SetConverter()` methods.

- **Configuration (Config):** If you do not specify a configuration, iKnow uses the default configuration. A configuration specifies what language(s) the text documents contain, and whether or not automatic language identification should be used. A configuration object is not domain-specific; you can use the same configuration for multiple domains. While not required, explicitly specifying a configuration is recommended.

- **Processor:** Using `lister.Init()` you can specify a processor and processor parameters. A processor reads the texts into iKnow. Specifying a processor is optional. If you do not specify a processor, iKnow uses the default processor and its parameter defaults. If you specify a processor, you can specify the processor parameter values, as shown in the following example:

```plaintext
SET flister=#class(%iKnow.Source.File.Lister).%New(domId)
SET processor="%iKnow.Source.File.Processor"
SET pparams=$LB("Latin1")
DO flister.Init("",processor,pparams,"","")
```

If explicitly specified, the processor subclass should be either of the same type as the Lister subclass (for example, `%iKnow.Source.File.Lister` takes `%iKnow.Source.File.Processor`) or `%iKnow.Source.Temp.Processor` if the Lister subclass has no corresponding Processor subclass. You can also create your own custom processor.

Processor parameters are specified as a Caché list. For `%iKnow.Source.File.Processor` the first list element is the name of the character set used (for example "Latin1"). The `%iKnow.Source.Temp.Processor` does not take any processor parameters.

- **Converter:** Using `lister.Init()` you can specify a user-defined converter and converter parameters. A Converter converts formatted source documents to plain text, removing HTML or XML tags, PDF formatting, or other non-text contents. Usually separate converters are used for each source document formatting type. Specifying a converter is optional. The default is to use no converter. If no converter is used, iKnow indexes formatting contents as well as text contents.

### 4.2.3 Lister Assigns IDs to Sources

The lister assigns two unique IDs to each source:

- **Source ID (internal ID):** a unique integer assigned by iKnow that is used for iKnow internal processing.
- **External ID:** a unique identifying string or number. The External ID is used as the link for any user-specified application that wishes to use iKnow. The External ID has the following structure:

```
| ListerReference: FullReference |
```

The Lister Reference is either the full class name of the Lister class used to load this source, or a short alias defined by the Lister class itself, prefixed with a colon. The Full Reference is a string for which the format is defined by the Lister class. It contains a Group Name and a Local Reference. It is up to the Lister to provide the implementation to derive the Group Name and Local Reference from this Full Reference, and to rebuild the Full Reference from the Group Name and Local Reference.

For example, the text file external ID :FILE:c:\mytextfiles\mydoc.txt consists of:

- **ListerReference:** the Lister class alias :FILE
- **FullReference:** c:\mytextfiles\mydoc.txt, which consists of the Group Name c:\mytextfiles\ and the Local Reference mydoc.txt.

For data in an SQL table, the ListerReference is :SQL. The Group Name is the groupfield, a field in the record that contains a unique value, and the Local Reference is the row ID.

For data in a string or global variable, the ListerReference is :TEMP.
The external ID format described here is the default; external ID format is configurable using the SimpleExtIds domain parameter.

You can access a source using either ID. The %iKnow.Queries.SourceAPI class contains methods for accessing these IDs. The GetByDomain() method returns both IDs for each source. Given the source ID, the GetExternalId() method returns the external ID. Given the external ID, the GetSourceId() method returns the source ID.

You can determine the lister class alias using the GetAlias() method of the %iKnow.Source.File.Lister class. If no alias exists, the External ID contains the full Lister class name.

### 4.2.4 Lister Defaults Example

The following is a minimal Lister and Loader example, taking all defaults. It establishes a domain, then creates Lister and Loader instance objects for that domain. It does not invoke lister.Init(), but takes the defaults for configuration, processor, and converter:

```
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SetListerAndLoader
    SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
    SET myloader=##class(%iKnow.Source.Loader).%New(domId)
UseListerAndLoader
    SET install=$SYSTEM.Util.InstallDirectory()
    SET dirpath=install_"dev\tutorials\iknow\data\reviews"
    SET stat=mylister.AddListToBatch(dirpath,$LB("txt","log"),0,"")
    WRITE "The lister status is ",$System.Status.DisplayError(stat),!
    SET stat=myloader.ProcessBatch()
    WRITE "The loader status is ",$System.Status.DisplayError(stat),!
```

Most examples in this book delete old data before using the Lister and Loader; this old data deletion is for demonstration purposes to allow these examples to be run repeatedly. Most examples in this book do not specify the processor and processor parameters, taking the defaults. Many examples in this book specify values for configuration rather than taking the defaults.

### 4.2.5 Lister Parameters

When you invoke a method to specify sources, you specify Lister parameters. You specify the same Lister parameters for the AddListToBatch() Lister instance method (for large batch loads of sources) and the ProcessList() Loader instance method (for adding a small number of sources to an existing batch of sources).

There are four Lister parameters that cumulatively define which sources are to be listed for iKnow indexing:

- **Path**: the location where the sources are located, specified as a string. This parameter is mandatory.

- **Extensions**: one or more file extension suffixes that identify which sources are to be listed. Specified as a Caché list data structure, each element of which is a string (refer to $LISTBUILD for details on Caché list data structures). By default the Lister selects all files in the Path directory that contain data, regardless of their file extension suffix. This includes files with no file extension suffix or with a file extension suffix indicating a non-text (such as .jpg). Empty files are not selected. Directories are not selected. When an extension suffix parameter is specified, the Lister selects only those files in the Path directory with that file extension suffix (or with no file extension suffix) that contain data.

- **Recursive**: a boolean value that specifies whether to search subdirectories of the path for sources. If selected, multiple levels of subdirectories are searched for sources. 1 = include subdirectories. 0 = do not include subdirectories. The default is 0.

- **Filter**: a string specifying a filter used to limit which sources are to be listed for iKnow indexing. For example, a user-designed filter could limit the Lister to only those files that have a specified substring in their file names. The default is to use no filter. (Note that this use of the word “filter” is completely separate from the filters in the %iKnow.Filters class that are used to include or exclude already-indexed sources supplied to an iKnow query.)
4.2.6 Batch or List?

iKnow provides two ways to load sources of all types, batch loading (ProcessBatch()) or list loading (ProcessList()). Both perform the same processing, they differ in their speed of execution. Which one you use depends primarily on how many sources you are loading. As a general rule, when loading ten or fewer sources, use ProcessList(); when loading one hundred or more sources, use ProcessBatch(). Which to use on intermediate numbers of sources depends on the nature of the specific sources.

4.3 Listing and Loading Examples

The examples in this section show the different ways to load sources:

- lister.AddListToBatch() and loader.ProcessBatch() to batch load a large number of sources.
- loader.SetLister() and loader.ProcessList() to load a small number of sources, or to add sources to an existing batch load.
- loader.BufferSource() and loader.ProcessBuffer() to load a string as a source. You can, of course, specify a local or global variable that contains the string.

You can also load sources as virtual sources using loader.ProcessVirtualList() or loader.ProcessVirtualBuffer(), as described in Loading a Virtual Source.

4.3.1 Loading Files

The following executable example performs a batch load of the source files in the Windows directory dirpath that have the extensions .txt or .log.

```
DomainCreateOrOpen
SET dname="mydomain"
IF (##class(%iKnow.Domain).Exists(dname))
  { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat
  { GOTO SetEnvironment }
ELSE
  WRITE "DropData error ",$System.Status.DisplayError(stat)
QUIT
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig")
  { SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
ELSE
  { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),","",1)
    DO cfg.%Save() }
CreateListerAndLoader
SET flister=##class(%iKnow.Source.File.Lister).%New(domId)
DO flister.Init("myconfig","","","")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
UseListerAndLoader
SET dirpath=$SYSTEM.Util.InstallDirectory()
SET dirpath=install\_dev\tutorials\iknow\data\reviews
SET stat=flister.AddListToBatch(dirpath,$LB("txt","log"),0,""
WRITE "The lister status is ",$System.Status.DisplayError(stat),!
SET stat=myloader.ProcessBatch()
WRITE "The loader status is ",$System.Status.DisplayError(stat),!
QueryLoadedSources
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources"
```

Using iKnow
This example performs a batch load, appropriate for loading a large number of files. To load a small number of files use the **SetLister()** and **ProcessList()** methods.

### 4.3.2 Loading SQL Records

The following executable example performs a batch load of the records of the Cinema.Review table. It loads as a source text the ReviewText field value for each record. If there is an error in the SQL query, the Loader returns an error status.

**iKnow** programs that load SQL data must use the `%iKnow.Source.SQL.Lister`. This lister always invokes the `%iKnow.Source.Temp.Processor`, which takes no parameters. There is, therefore, no reason to specify the processor, unless you have created your own custom processor.

```plaintext
ZNSPACE "Samples"
DomainCreateOrOpen
SET dname="mydomain"
IF (##class(%iKnow.Domain).Exists(dname))
  { WRITE "The ",dname," domain already exists",!
    SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { WRITE "The ",dname," domain does not exist",!
    SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    WRITE "Created the ",dname," domain with domain ID ",domoref.Id,!
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { WRITE "Deleted the data from the ",dname," domain",!!
  GOTO SetEnvironment }
ELSE    { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT}
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
  SET cfg=##class(%iKnow.Configuration).Open("myconfig")
} ELSE {
  SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
  DO cfg.%Save() }
CreateListerAndLoader
SET flister=##class(%iKnow.Source.SQL.Lister).%New(domId)
DO flister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
QueryBuild
SET myquery="SELECT ID AS UniqueVal,Film,ReviewText FROM Cinema.Review"
SET idfld="UniqueVal"
SET grpfld="Film"
SET dataflds=$LB("ReviewText")
SET metaflds=$LB("Film")
SET mkeys=$LB("Film")
UseLister
SET stat=flister.AddListToBatch(myquery,idfld,grpfld,dataflds,metaflds)
UseLoader
SET stat=myloader.ProcessBatch()
QueryLoadedSources
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources"
```

A metafield is the name (or alias) assigned to the field in the original source. A metakey is the name that you assign to the corresponding iKnow display field. If you do not specify metakeys, the metafield names are used for iKnow display.

This example performs a batch load, appropriate for loading a large number of SQL records. To load a small number of SQL records use the **SetLister()** and **ProcessList()** methods.

### 4.3.3 Loading Elements of a Subscripted Global

The following executable example loads the elements of a subscripted global. It uses the `%iKnow.Source.Global.Lister` and specifies the following Lister parameters to the **ProcessList()** method: global name, first subscript (inclusive), and last subscript (inclusive). This example uses the ^Aviation.AircraftD global, found in the Samples namespace. Because this is a sparse array, only a few of the subscripts between 1 and 50,000 contain data:
DomainCreateOrOpen
ZYSPACE "Samples"
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
{ SET domoref=#class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ SET domoref=#class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",#$System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF #class(%iKnow.Configuration).Exists("myconfig")
{ SET cfg=#class(%iKnow.Configuration).Open("myconfig")
  GOTO ListerAndLoader }
ELSE
{ SET cfg=#class(%iKnow.Configuration).%New("myconfig",0,##LISTBUILD("en"),",",1)
  DO cfg.%Save() }
ListerAndLoader
SET mylister=#class(%iKnow.Source.Global.Lister).%New(domId)
DO mylister.Init("myconfig","","","")
SET myloader=#class(%iKnow.Source.Loader).%New(domId)
IF stat='1 { WRITE "SetLister error ",#$System.Status.DisplayError(stat)
  QUIT }
SET gbl="^Aviation.AircraftD"
SET stat=myloader.ProcessList(gbl,1,50000)
IF stat='1 { WRITE "ProcessList error ",#$System.Status.DisplayError(stat)
  QUIT }
SourceSentenceQueries
SET numSrcD=#class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The domain contains ",numSrcD," sources",
SET numSentD=#class(%iKnow.Queries.SentenceAPI).GetCountByDomain(domId)
WRITE "These sources contain ",numSentD," sentences"

The ProcessList() method can specify only one subscript level at a time. In order to iterate through multiple subscript levels, you must write code to invoke this method at the desired subscript level. For example, to load the second level subscripts 1 and 2, you would write code such as the following:

FOR i=1:1:90000 {
  SET gbl="^Aviation.NarrativeS("_i_")"
  SET stat=myloader.ProcessList(gbl,1,2) }

This loads globals such as ^Aviation.NarrativeS(85879,1) and ^Aviation.NarrativeS(85879,2).

4.3.4 Loading a String

The following executable example loads a single global (or a string literal) as a source file. Note that no Lister is required when loading a string.

DomainCreateOrOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
{ WRITE "The ",dname," domain already exists",
  SET domoref=#class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ WRITE "The ",dname," domain does not exist",
  SET domoref=#class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  SET domId=domoref.Id
  WRITE "Created the ",dname," domain with domain ID ",domId,
  GOTO CreateLoader }
DeleteOldData
SET stat=domoref.DropData()
IF stat { WRITE "Deleted the data from the ",dname," domain",
  SET domId=domoref.Id
  GOTO CreateLoader }
ELSE
{ WRITE "DropData error ",#$System.Status.DisplayError(stat)
  QUIT }
CreateLoader
SET myloader=#class(%iKnow.Source.Loader).%New(domId)
UseLoader
SET ^a="I drove at 70mph then sped up to 100mph when the light changed."
DO myloader.BufferSource("ref","^a")
DO myloader.ProcessBuffer()
QuerySources
WRITE "number of sources: ",##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)

The first argument of the BufferSource() method specifies a unique external source Id. The following example creates a separate source for each global subscript:

SET i=1
WHILE $DATA(^a(i)) {
  DO myloader.BufferSource("ref"_i,^a(i))
  DO myloader.ProcessBuffer()
  SET i=i+1 }
WRITE "end of data"

### 4.4 Updating the Domain Contents

After you have performed an initial load of sources to a domain, you can change this list of sources by adding sources or by deleting sources. Updating a domain refers to responding to changes in the set of source texts. This should not be confused with upgrading a domain, which refers to responding to changes in the iKnow software, commonly after installing a significant new version of Caché.

#### 4.4.1 Adding Sources

After you have performed an initial load of sources to a domain (using the AddListToBatch() and ProcessBatch() methods) you may want to add more files to the list of sources. This is done using the SetLister() and ProcessList() methods. The ProcessList() method takes the same parameters as the AddListToBatch() method.

- To add a one source at a time: SET stat=myloader.ProcessList("C:\mytestfiles\newfile.txt")
- To add a directory of sources: SET stat=myloader.ProcessList("C:\mytestfiles\logfiles",$LB("log"),0,"")

Adding more sources to a batch load is shown in the following example:

```
DomainCreateOrOpen
  SET dname="mydomain"
  IF ##class(%iKnow.Domain).Exists(dname) {
    SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
  ELSE { SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save() }
  GOTO SetEnvironment }
DeleteOldData
  SET stat=domoref.DropData()
  IF stat { GOTO SetEnvironment }
  ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig") {
    SET cfg=##class(%iKnow.Configuration).Open("myconfig")
  } ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
    DO cfg.%Save() }
ListerAndLoader
  SET flister=##class(%iKnow.Source.File.Lister).%New(domId)
  DO flister.Init("myconfig","","","")
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(flister)
SourceBatchLoad
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install_"dev\tutorials\iknow\data\reviews"
  SET stat=flister.AddListToBatch(dirpath,$LB("txt"),0,"")
  SET stat=myloader.ProcessBatch()
  IF stat != 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
    QUIT }
QueryLoadedSources
  WRITE "Source count is ",##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId),!
ExpandListofSources
```
4.4.2 Deleting Sources

You can remove a source that has been loaded to a domain using the **DeleteSource()** method of the **%iKnow.Source.Loader** class. This method cannot be used to delete a virtual source; a separate **DeleteVirtualSource()** method is provided for this purpose.

### 4.5 Loading a Virtual Source

A virtual source is a source that is not static. You might, for example, use a virtual source for a file that is being frequently modified. The srcId of a virtual source is a negative integer. The external Id of a virtual source begins with the ListerReference (the Lister class alias), commonly :TEMP.

Adding a virtual source does not update iKnow statistics. For this reason, using a virtual source may be desirable when you wish to temporarily add sources for a specific purpose without incurring the overhead of revising the domain statistics. You should use a virtual source when adding a source that is being continuously modified, such as a source in the process of being written. Because the virtual source Id is a negative number, it is easy to distinguish virtual sources from regular sources. Different methods are used to delete virtual sources and regular sources.

You can load virtual sources using **loader.SetLister()** and **loader.ProcessVirtualList()** or **loader.BufferSource()** and **loader.ProcessVirtualBuffer()**. The following program loads a virtual source using **ProcessVirtualBuffer()**.

---

**Loading a Virtual Source**

A virtual source is a source that is not static. You might, for example, use a virtual source for a file that is being frequently modified. The srcId of a virtual source is a negative integer. The external Id of a virtual source begins with the ListerReference (the Lister class alias), commonly :TEMP.

Adding a virtual source does not update iKnow statistics. For this reason, using a virtual source may be desirable when you wish to temporarily add sources for a specific purpose without incurring the overhead of revising the domain statistics. You should use a virtual source when adding a source that is being continuously modified, such as a source in the process of being written. Because the virtual source Id is a negative number, it is easy to distinguish virtual sources from regular sources. Different methods are used to delete virtual sources and regular sources.

You can load virtual sources using **loader.SetLister()** and **loader.ProcessVirtualList()** or **loader.BufferSource()** and **loader.ProcessVirtualBuffer()**. The following program loads a virtual source using **ProcessVirtualBuffer()**.
SET status=myloader.ProcessVirtualBuffer(config)

SET vsrclist=myloader.GetSourceIds()
FOR i=1:1:$LL(vsrclist) {
  SET srcid=-$LIST(vsrclist,i)
  WRITE "External Id=",##class(%iKnow.Queries.SourceAPI).GetExternalId(domId,srcid)
  WRITE "  Source Id=",srcid,!
  WRITE "  Sentence Count=",##class(%iKnow.Queries.SentenceAPI).GetCountBySource(domId,$lb(srcid)),!
}

Note that the %iKnow.Queries.SourceAPI.GetCountByDomain() method does not count virtual sources. You can determine if a virtual source has been loaded by invoking %iKnow.Queries.SourceAPI.GetExternalId(domId,-1). Here -1 is the srcId of the first virtual source loaded.

By default, many iKnow queries process only ordinary sources and ignore virtual sources. To use these queries to process a virtual source you must specify a vSrcId parameter value for the query method.

### 4.5.1 Deleting a Virtual Source

The %iKnow.Source.Loader class provides two methods for deleting virtual sources.

- **DeleteVirtualSource()** deletes a single virtual source indexed for a domain. You specify the domain Id (a positive integer) and the virtual source Id (a negative integer). This deletes all iKnow entities generated for this source text.

- **DeleteAllVirtualSources()** deletes all of the virtual sources indexed for a specified domain. This deletes all iKnow entities generated for these source texts.

### 4.6 Copying and Re-indexing Loaded Source Data

After you have successfully loaded sources into a domain, you may wish to copy some or all of these sources to another domain. When iKnow copies these loaded sources it also re-indexes them. The copied sources therefore have different source Ids and entity Ids; the external Ids are not changed.

Some reasons you might want to copy/re-index from one domain to another:

- To create a copy of a domain. You may wish to make a backup copy, or to create a copy to serve as a snapshot of the domain at a particular time. For example, when indexing RSS feeds you may wish to create a snapshot because these feeds change over time; at a future date you might no longer have access to the original source data.

- To create a domain containing a subset of the original set of sources. The new domain can be smaller, more efficient, and easier to work with. You can specify this copied subset of sources by a list of source Ids to copy, or by a filter that limits which sources to copy. For example, you could create a domain consisting of only the newest sources, which you could then query without having to filter by date for each query.

- To create a domain containing the merged sets of sources from two domains, or to add sources from one domain into a domain that already contains sources.

- To re-index the sources in a domain after extreme modification of the set of sources. For example, if you very frequently add or delete multiple sources in a domain, the indexing may no longer be optimal. (Normal adding and deleting of sources does not degrade index performance.) By copying the domain, you re-index the current sources that you are copying, making the indexing in the new domain optimal.

- To apply iKnow language model revisions. Release versions of iKnow commonly contain improvements to its language models. These may include introduction of support for new languages and improvements to already-supported languages. Copying the set of sources in a domain re-indexes these sources, and therefore applies the most current iKnow language models to the copied sources.
You use the `%iKnow.Source.Domain.Lister` class to copy/re-index from one domain to another. The new domain must already be defined before you can create a Lister instance for this class using the `%New()` method. Both domains must be in the same namespace.

The following example populates the mydomain domain, then copies the contents of mydomain to an empty domain named newdomain, automatically re-indexing the newdomain contents:

```plaintext
EstablishAndPopulateFirstDomain
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET dirpath=install_"dev\tutorial\iknow\data\reviews"
SET stat=mylister.AddListToBatch(dirpath,$LB("txt","log"),0,"")
SET stat=myloader.ProcessBatch()

TestQueryFirstDomain
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources in the from domain",!

CreateSecondDomain
SET domNewId=domOref.Id
DO domOref.%Save()
SET domNewId=domOref.Id

CopyAndReindexFromFirstDomainToSecondDomain
SET newlister=##class(%iKnow.Source.Domain.Lister).%New(domNewId)
SET newloader=##class(%iKnow.Source.Loader).%New(domNewId)
SET stat=newlister.AddListToBatch(domId)
SET stat=newloader.ProcessBatch()

TestQuerySecondDomain
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domNewId)," sources in the to domain"

CleanUpForNextTime
SET stat=##class(%iKnow.Domain).%DeleteId(domNewId)
IF stat '=' 1 {WRITE "Domain delete error:",stat }
```

The `AddListToBatch()` method can take a second parameter to specify which sources are to be copied. It can either specify a list of sources (a comma-separated list of source Id integers) or by specifying a filter.

The following example is identical to the previous example, except that it limits which sources are to be copied by specifying a comma-separated list of source Ids.

```plaintext
EstablishAndPopulateFirstDomain
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET dirpath=install_"dev\tutorial\iknow\data\reviews"
SET stat=mylister.AddListToBatch(dirpath,$LB("txt","log"),0,"")
SET stat=myloader.ProcessBatch()

TestQueryFirstDomain
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources in the from domain",!

CreateSecondDomain
SET domNewId=domOref.Id
DO domOref.%Save()
SET domNewId=domOref.Id

SubsetOfSourcesToCopy
SET subset="1,3,5"

CopyAndReindexFromFirstDomainToSecondDomain
SET newlister=##class(%iKnow.Source.Domain.Lister).%New(domNewId)
SET newloader=##class(%iKnow.Source.Loader).%New(domNewId)
SET stat=newlister.AddListToBatch(domId,subset)
SET stat=newloader.ProcessBatch()

TestQuerySecondDomain
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domNewId)," sources in the to domain"

CleanUpForNextTime
SET stat=##class(%iKnow.Domain).%DeleteId(domNewId)
IF stat '=' 1 {WRITE "Domain delete error:",stat }
```

### 4.6.1 UserDictionary and Copied Sources

A UserDictionary is applied when a source is listed. Therefore, any UserDictionary modifications made to the initial loaded sources will appear in the copied sources. However, because the copy operation is also a list operation, you can also apply a new UserDictionary to modify the sources as they are copied.

For example, the UserDictionary used when the sources were originally listed substitutes “Doctor” for the abbreviation “Dr.”; this substitution will be present in the copied sources. Later you modified the UserDictionary to also substitute
“Doctor” for “Physician”. This change to your UserDictionary had no effect on the already-loaded sources. When you copy the sources, you apply this revised UserDictionary. The “Dr.” to “Doctor” substitution is performed 0 times, because that substitution is already present in the initial loaded sources; the “Physician” to “Doctor” substitution is performed on the copied sources.
Performance Considerations when Loading Texts

Because iKnow typically handles large amounts of text data, the following performance considerations should be heeded when loading source text:

• Before starting a batch load of a significant number of sources, stop database journaling. Once the batch load completes, make sure to restart journaling. Refer to the “Journaling” chapter of the Caché Data Integrity Guide for information on stopping and restarting journaling.

• Before starting a batch load of a significant number of sources (or a small number of very large sources), set the global buffer pool to a size large enough to handle this operation. iKnow indexing creates a large number of temporary globals. If the global buffer pool is not large enough to handle these temporary globals in memory, they are written to disk. These disk I/O operations can significantly affect iKnow performance. Refer to “Memory and Startup Settings” in the “Configuring Caché” chapter of the Caché System Administration Guide.

• iKnow indexing requires substantially more disk space than the space occupied by the source texts. The approximate space requirements for temporary and permanent globals are described in “Globals and Space Requirements” section of the “Implementation” chapter of this manual.

• Do not configure more language support than is required for your sources. Your iKnow Configuration should specify only those languages that are actually found in your sources. If all of your sources are in one language, do not specify automatic language identification. Unless n-grams are required for the language, do not set the EnableNgrams domain parameter.
6
iKnow Queries

The iKnow semantic analysis engine supplies a large number of query APIs which are used to return text entities and statistics about these text entities. For example, the `%iKnow.Queries.CrcAPI.GetTop()` method returns the most frequently occurring CRCs in a specified domain. The `%iKnow.Queries.CrcAPI.GetCountBySource()` returns the total number of unique CRCs that appear in the specified sources.

6.1 Types of Queries

There are three types of queries provided. They are distinguished by their name suffixes:

- API: Caché ObjectScript queries
- QAPI: Caché SQL queries
- WSAPI: SOAP-accessible Web Services queries

For each of these types, iKnow provides queries for:

- **Entities**: return all entities in a source or multiple sources; the most frequently occurring entities; entities similar to a supplied string, etc.
- **CCs**: return concept-concept pairs.
- **CRCs**: return concept-relation-concept (master-relation-slave) sequences.
- **Paths**: return chain of concept-relation-concept sequences within a sentence. A path contains a minimum of two CRCs (CRCRC).
- **Sentences**: return sentences that contain a specified CRC, entity, etc.
- **Sources**: return sources that contain a specified CRC, entity, etc.

6.2 Queries Described in this Chapter

This chapter describes and provides examples of many commonly-used iKnow queries:

- Counting sources in a domain
- Counting sentences in a source
6.3 Query Method Parameters

The following parameters are common to many query methods:

- **domainid**: The domain ID is an integer that identifies the domain.

- **result**: If a query returns an array of values rather than just a single result value, the result set is passed by reference (using the dot prefix operator, for example, `.result`). You can then use ZWRITE to display the whole result set in raw Caché list format, or use a loop structure and list-to-string conversion to return one row at a time, as shown in the examples in this chapter.

- **page and pagesize (optional)**: To prevent methods from retrieving and returning thousands of records, the Query API uses a paging mechanism to allow the user to limit the number of results returned. It divides the results into equal-length pages, with the length of each page specified as the pagesize. For example, if you want the first ten results, you specify page 1 and a pagesize of 10. If you want the next page of results, you specify page 2 and pagesize 10. The default values are page=1 and pagesize=10.

- **setop (optional)**: If a query applies more than one selection criteria, the Setop logical operator specifies whether the query should return the union or the intersection of the result sets. 1 (SUCCESSINTERSECT) returns results that match any of the supplied selection criteria. 2 (SUCCESSUNION) returns results that match all of the supplied selection criteria. The default is SUCCESSUNION.

- **entitylist**: In queries that return matches to an entity (for example, GetByEntities(), GetRelated(), GetSimilar()) a Caché list of entities. You can specify entitylist entities in any mix of uppercase and lowercase letters; iKnow matches them against indexed entities normalized to lowercase.

- **vSrcId**: The source Id of a virtual source, specified as a negative integer. If specified, only entities in that virtual source are processed by the query. If omitted or specified as 0, only ordinary sources are considered by the query and virtual sources are ignored. The default is 0.

6.4 Counting Sources and Sentences

To count the number of sources loaded, you can use the GetCountByDomain() method of the %iKnow.Queries.SourceAPI class.
To count the sentences in all of the sources loaded, you can use the `GetCountByDomain()` method of the `%iKnow.Queries.SentenceAPI` class. To count the sentences in a single source, you can use the `GetCountBySource()` method.

The following example demonstrates these sentence count methods:

```
DomainCreateOrOpen
SET dname="mydomain"
IF (##class(%iKnow.Domain).Exists(dname))
  { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") { 
  SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),","1) 
  DO cfg.%Save() }
ListerAndLoader
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_"dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,""
IF stat = 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
  QUIT }
SourceSentenceQueries
SET numSrcD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The domain contains ",numSrcD," sources",!
SET numSentD=##class(%iKnow.Queries.SentenceAPI).GetCountByDomain(domId)
WRITE "These sources contain ",numSentD," sentences",!!
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId,1,20)
SET i=1
WHILE $DATA(result(i)) {
  SET extId = $LISTGET(result(i),2)
  SET fullref = $PIECE(extId,"\",1,4)
  SET fname = $PIECE(fullref,"\",0,$LENGTH(extId,"\"))
  SET numSentS = ##class(%iKnow.Queries.SentenceAPI).GetCountBySource(domId,result(i))
  WRITE fname," has ",numSentS," sentences",!
  SET i=i+1 }
```

### 6.4.1 What is a Sentence?

iKnow uses a language model to divide the source text into sentences. If a file contains any content at all (other than blank spaces), the initial content is treated as the first sentence. A period followed by a blank space usually indicates a sentence break, though iKnow language models recognize exceptions to this rule. For example, the English language model recognizes common abbreviations, such as “Dr.” and “Mr.”; in these cases iKnow removes the period rather than performing a sentence break.

iKnow also breaks sentences when a line return occurs. Therefore, the title of a text and the section headings within the text are considered to be sentences.

You can use the `UserDictionary` option of your Configuration to cause or avoid sentence endings in specific cases. For example, the abbreviation “Fr.” (Father or Friar) is not recognized by the English language model. It is treated as a sentence break. You can use UserDictionary to either remove the period or to specify that this use of a period should not cause a sentence break. UserDictionary is applied as a source is loaded; already loaded sources are not affected.
6.5 Counting Entities

To count the number of sources that contain one or more occurrences of a specified entity, you can use the `GetCountByEntities()` method of the `%iKnow.Queries.SourceAPI` class. In this method you can specify a list on one or more entities to search for in the loaded sources.

Note that here, and throughout iKnow, the concept of “entity” differs significantly from the familiar notion of a search term. For example, the entity “dog” does not occur in the sentence “The quick brown fox jumped over the lazy dog.” The entity “lazy dog” does occur in this sentence. An entity can be a concept or a relation; you could, for example, count the number of sources that contain the entity “is” or the entity “jumped over”. However, in these examples and in most real-world cases, iKnow matches concepts or concepts associated by a relation.

The following example demonstrates these query count methods:

```iKnow
#include %IKPublic
DomainCreateOrOpen
  SET dname="mydomain"
  IF (%#class(%iKnow.Domain).Exists(dname))
    { SET domoref=%#class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData } 
  ELSE
    { SET domoref=%#class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment } 
DeleteOldData
  SET stat=domoref.DropData()
  IF stat { WRITE "DropData error ",%System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF %#class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=%#class(%iKnow.Configuration).Open("myconfig") }
  ELSE
    { SET cfg=%#class(%iKnow.Configuration).%New("myconfig",0,%LISTBUILD("en"),","1)
      DO cfg.%Save() }
ListerAndLoader
  mylister=%#class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  myloader=%#class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET install=%SYSTEM.Util.InstallDirectory()
  SET dirpath=install_"dev\tutorials\iknow\data\reviews"
  SET stat=myloader.ProcessList(dirpath,%LB("txt"),0,""
  IF stat = -1 { WRITE "Loader error ",%System.Status.DisplayError(stat)
    QUIT }
SourceCount
  WRITE %#class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources",
SingleEntityCounts
  SET ent=%LB("movie","movies","film","films","road movies","screen")
  SET ptr=0
  FOR x=1:1:100 {
    SET stat=%LISTNEXT(ent,ptr,val)
    IF stat=1 
      { WRITE %#class(%iKnow.Queries.SourceAPI).GetCountByEntities(domId,%LB(val))," contain ",val,!
      }
    ELSE
      { GOTO MultipleEntityCounts }
  }
MultipleEntityCounts
  WRITE %#class(%iKnow.Queries.SourceAPI).GetCountByEntities(domId,%LB("film","movie"),"",$$SYSTEMINTERSECT)
  WRITE " sources contain 'film' AND 'movie'
```

The last two queries in this example apply the Setop logical operator to the result set, returning a count of the sources that contain any of the listed entities ($$$UNION), or all of the listed entities ($$$INTERSECT). The default is $$$UNION.
6.6 Listing Most-Frequently-Occurring Entities

An iKnow query can return the most frequently occurring entities in the source documents in descending order of frequency. Each entity is returned as a separate record in Caché list format.

The entity record format is as follows:

• The entity ID, a unique integer assigned by iKnow.
• The entity value, specified as a string.
• Frequency: an integer count of how many times the entity occurs in the source documents.
• Spread: an integer count of how many source documents contain the entity.

The following query returns the most frequent (top) entities in the sources loaded by this program. It sets the page (1) and pagesize (12) parameters to specify how many entities to return. It returns (at most) the top 12 entities:

```cachê
DomainCreateOrOpen
  SET dname="mydomain"
  IF (!$class(%iKnow.Domain).Exists(dname))
    { SET domoref=$class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData }
  ELSE
    { SET domoref=$class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment }
DeleteOldData
  SET stat=domoref.DropData()
  IF stat
    { GOTO SetEnvironment }
  ELSE
    { WRITE "DropData error ",$System.Status.DisplayError(stat)
      QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF (!$class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=$class(%iKnow.Configuration).Open("myconfig") }
  ELSE
    { SET cfg=$class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
      DO cfg.%Save() }
ListerAndLoader
  SET mylister=$class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  SET myloader=$class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install_"dev\tutorials\iknow\data\reviews"
  SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
  IF stat
    { WRITE "Loader error ",$System.Status.DisplayError(stat)
      QUIT }
RunQueries
  SET i=1
  DO $$class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources",!
    DO $$class(%iKnow.Queries.EntityAPI).GetTop(.result,domId,1,12)
DisplayGetTopResults
  SET i=1
  WHILE $DATA(result(i))
    { WRITE $LISTTOSTRING(result(i),",",1),!
      SET i=i+1 }
  WRITE "Printed the top ",i-1," entities!
```

By default, `GetTop()` returns the top concepts ($$$ENTTYPECONCEPT); it does not return relations. To return the top relations, specify a value of 1 ($$$ENTTYPEREATION) for the 8th parameter:

`GetTop(.result,domId,1,nument,,,,,1)`

6.7 Creating a Black List

A black list is a list of entities that you do not want a query to return. For example, if your source texts include greetings and salutations, you might want to put “many thanks”, “best regards”, and other stock phrases with no real information
content in your black list. A black list might also be used to suppress top concepts that are too general and widespread to be of interest when analyzing query results.

**Note:** When displaying results of a query that applied a black list, it is important to note that use of a black list silently changes the query results by suppressing some information. Make sure the black lists used are relevant for the data contents, for the user looking at the query results, and for the context in which query results are displayed.

A black list is assigned to a specific domain. You can define a black list by using the %iKnow.Utils.MaintenanceAPI class methods to define, populate, and maintain black lists. This class provides methods for populating a black list either by specifying entities as strings or by specifying entities by entity Id. You can use the CopyBlackLists() method of the %iKnow.Utils.CopyUtils class to copy all defined black lists in a domain to another domain.

The Knowledge Portal and the Basic Portal user interfaces support the use of black lists.

The following example suppresses movie terms that are too general to be of interest. It uses CreateBlackList() to create a black list, uses AddStringToBlackList() to add entities to the black list, then supplies the black list to the GetTop() method:

```plaintext
DomainCreateOrOpen
SET dname="mydomain"
IF (%class(%iKnow.Domain).Exists(dname))
{ SET domoref=%class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ SET domoref=%class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",%System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF %class(%iKnow.Configuration).Exists("myconfig")
{ SET cfg=%class(%iKnow.Configuration).Open("myconfig") }
ELSE
{ SET cfg=%class(%iKnow.Configuration).%New("myconfig",0,%LISTBUILD("en"),"",1)
  DO cfg.%Save() }
ListerAndLoader
SET mylister=%class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=%class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_"dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat = -1 { WRITE "Loader error ",%System.Status.DisplayError(stat)
  QUIT }
CreateBlackList
SET blname="MoviesBlackout"
SET blId=%class(%iKnow.Utils.MaintenanceAPI).CreateBlackList(domId,blname,
  "The movie reviews black list")
PopulateBlackList
SET black=$LB("movie","movies","film","films","cinema","cinemas","screen","screens")
SET ptr=0
FOR x=0:1:100 {
  SET moredata=$LISTNEXT(black,ptr,val)
  IF moredata=1 {
    SET stat=%class(%iKnow.Utils.MaintenanceAPI).AddStringToBlackList(domId,blId,val)
  }
  ELSE { WRITE x," entities in black list",!
    GOTO RunQueries }
}
RunQueries
WRITE %class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources",!,!
DO %class(%iKnow.Queries.EntityAPI).GetTop(.result,domId,1,12,"",0,0,0,0,$LB(blId))
DisplayGetTopResults
WRITE "NOTE: the ",blname," black list",!,
  "has been applied to this list of top entities",!
SET i=1
WHILE $DATA(result(i)) {
  WRITE $LISTTOSTRING(result(i),","),!
  SET i=i+1 }
WRITE "Printed the top ",i-1," entities",!
```
The CreateBlackList() method allows you to specify both a name and a description for your black list. A black list name can be any valid string of any length; black list names are case-sensitive. The name you assign to a black list must be unique for the domain. Specifying a duplicate black list name generates ERROR #8091. The black list description is optional; it can be a string of any length.

6.7.1 Queries that Support Black Lists

The following query methods provide a parameter to specify black lists. You can specify multiple black lists to any of these methods by specifying the black list Ids as elements of a %List structure, using the SLISTBUILD function.

Entity Queries:
- %iKnow.Queries.EntityAPI.GetByFilter()
- %iKnow.Queries.EntityAPI.GetBySource()
- %iKnow.Queries.EntityAPI.GetCountByDomain()
- %iKnow.Queries.EntityAPI.GetCountBySource()
- %iKnow.Queries.EntityAPI.GetNewBySource()
- %iKnow.Queries.EntityAPI.GetRelated()
- %iKnow.Queries.EntityAPI.GetRelatedById()
- %iKnow.Queries.EntityAPI.GetSimilar()
- %iKnow.Queries.EntityAPI.GetSimilarCounts()
- %iKnow.Queries.EntityAPI.GetTop()

Sentence Queries:
- %iKnow.Queries.SentenceAPI.GetNewBySource()

Source Queries:
- %iKnow.Queries.SourceAPI.GetSimilar(): iKnow ignores blacklisted entities both when selecting similar entities and when calculating similarity scores.

6.8 CRC Queries

An iKnow query that returns a CRC (Concept-Relation-Concept sequence) returns it in the following format:

- The CRC ID, a unique integer assigned by iKnow.
- The Master Concept, specified as a string.
- The Relation, specified as a string.
- The Slave Concept, specified as a string.
- Frequency: an integer count of how many times the CRC occurs in the source documents.
- Spread: an integer count of how many source documents contain the CRC.
6.8.1 Listing CRCs that Contain Entities

One common use of CRCs is to specify an entity (usually a Concept) and return the CRCs that contain that entity. This provides the various contexts in which an entity appears in a source (or sources). Because iKnow normalizes all text to lowercase letters, you must specify these matching entities in lowercase.

The following query returns all of the CRCs that contain the specified Concepts ("movie", "movies", "road movies") as either the master concept or the slave concept of a CRC:

```plaintext
DomainCreateOrOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
{ SET domoref=#class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ SET domoref=#class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF #class(%iKnow.Configuration).Exists("myconfig")
{ SET cfg=#class(%iKnow.Configuration).Open("myconfig")
 ELSE { SET cfg=#class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
  DO cfg.%Save() }
 ListerAndLoader
SET mlister=#class(%iKnow.Source.File.Lister).%New(domId)
DO mlister.Init("myconfig")
SET mloader=#class(%iKnow.Source.Loader).%New(domId)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install\_dev\_tutorials\iknow\data\reviews
SET stat=mloader.SetLister(mlister)
SET stat=mloader.ProcessList(dirpath,$LB("txt"),0,""
IF stat "= 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
  QUIT }
CRCQuery
SET myconcepts=$LB("movie","movies","road movies")
DO #class(%iKnow.Queries.CrcAPI).GetByEntities(.result,domId,myconcepts)
SET i=1
WHILE $DATA(result(i)) {
  WRITE $LISTTOSTRING(result(i),",",1),!
  SET i=i+1 }
WRITE !,"That's all folks!"
```

6.8.2 Counting Sources that Contain a CRC

The following program example returns the count of sources that contain the specified CRCs. To specify CRCs to the GetCountByCrcs() method, you must specify each CRC as a %List (using $LB), and then group these CRCs together as a %List. This is shown in the following example:

```plaintext
DomainCreateOrOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
{ SET domoref=#class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ SET domoref=#class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF #class(%iKnow.Configuration).Exists("myconfig")
{ SET cfg=#class(%iKnow.Configuration).Open("myconfig")
 ELSE { SET cfg=#class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
  DO cfg.%Save() }
 ListerAndLoader
```
The `GetCountByCrcs()` method returns the count of sources that contain any of the specified CRCs.

### 6.8.3 Listing Sources or Sentences that Fulfill a CRC Mask

You can use a CRC mask to specify an entity value for a specific CRC position. Each CRC has three positions: master, relation, and slave. With a CRC mask you can specify either an entity value or a wildcard for each position. A CRC mask enables you to list sources or sentences that contain CRCs that match one or more positional values. Because it specifies both position and entity value, the `GetByCrcMask()` partial CRC match is a more restrictive match than `GetByEntities()`, but a less restrictive match than `GetByCrcs()`.

The following example uses a CRC mask that matches the entity “movie” in master position, while using wildcards to permit any value in the CRC relation and slave positions. The `GetByCrcMask()` method matches this mask against every sentence in each source, and returns the sentence Id and the sentence text of those sentences that contain a CRC with “movie” in the master position.
6.9 Listing Similar Entities

You can list the unique entities that are similar to a specified string. An entity is similar if one of the following applies:

- The string is identical to the entity.
- The string is one of the words of the entity.
- The string is the first letters of one of the words of the entity.

Similarity returns each unique entity (Master Concept or Slave Concept) with integer counts of its frequency and spread, in descending sort order of these integer counts. Similarity does not match Relations. As is true throughout iKnow, matching ignores letter case; all entities are returned in lowercase letters. Similarity does not use stemming logic; “cat” returns both “cats” and “category”.

The following example lists the entities that are similar to the string “movie”:

```plaintext
DomainCreateOrOpen
SET dname="mydomain"
IF (**class(%iKnow.Domain).Exists(dname)) { SET domoref==**class(%iKnow.Domain).Open(dname) GOTO DeleteOldData }
ELSE { SET domoref==**class(%iKnow.Domain).%New(dname) DO domoref.%Save() GOTO SetEnvironment } DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment } ELSE { WRITE "DropData error ",System.Status.DisplayError(stat) QUIT }
SetEnvironment
SET domId=domoref.Id
IF **class(%iKnow.Configuration).Exists("myconfig") { SET cfg==**class(%iKnow.Configuration).Open("myconfig") } ELSE { SET cfg==**class(%iKnow.Configuration).%New("myconfig",0,%LISTBUILD("en"),",",1) DO cfg.%Save() } ListerAndLoader
SET mylister==**class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader==**class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.ProcessList(dirpath,%LB("txt"),0,"")
IF stat != 1 { WRITE "Loader error ",System.Status.DisplayError(stat) QUIT }
SourceCountQueries
WRITE **class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," total sources",!
WRITE **class(%iKnow.Queries.SourceAPI).GetCountByEntities(domId,%LB("movie")), " sources contain 'movie'",!
SimilarEntityQuery
WRITE ","Movie Similarity:!
DO **class(%iKnow.Queries.EntityAPI).GetSimilar(.simresult,domId,"movie",1,20)
SET j=1
WHILE $DATA(simresult(j)) { WRITE %LISTTOSTRING(simresult(j),",",1),!
    SET j=j+1 }
```

The default **domain parameter setting governing entity similarity is EnableNgrams, a boolean value.
6.9.1 Parts and N-grams

The GetSimilar() and GetSimilarCounts() methods have a mode parameter that specifies where to search for similarity. There are two available values:

- $$\text{USEPARTS}$$ causes iKnow to match the beginning of each part (word) for similarity. For texts in English and most other languages this is generally the preferred setting. $$\text{USEPARTS}$$ is the default.

- $$\text{USENGRAMS}$$ causes iKnow to match words and linguistic units within words (n-grams) for similarity. This mode is used when the source text language compounds words. For example, $$\text{USENGRAMS}$$ would commonly be used with German, a language which regularly forms compound words. $$\text{USENGRAMS}$$ would not be used with English, a language which does not compound words. $$\text{USENGRAMS}$$ can only be used in a domain that has the EnableNgrams domain parameter set.

6.10 Listing Related Entities

An entity is related to another entity if both occur in a CRC. By default, the related entity can be either a master concept or a slave concept. (Refer to “Limiting by Position” (below) to override this default.)

The following example shows how iKnow returns related entities. It first determines how many CRCs contain the entity “road movies” and lists these CRCs. (In this small example, you can simply read all the CRCs to see what is related to “road movies”; in a much larger collection of sources this would not be practical.) The program example then lists all of the entities that are related to “road movies” as either slave or master (you can confirm these relations by matching these entities against the CRCs listed earlier):

```iKnow
DomainCreateOrOpen
  SET dname="mydomain"
  IF (##class(%iKnow.Domain).Exists(dname))
    { SET domoref=##class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData }
  ELSE
    { SET domoref=##class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment }
DeleteOldData
  SET stat=domoref.DropData()
  IF stat { GOTO SetEnvironment }
  ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
  ELSE
    { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),",",1) }
    DO cfg.%Save() }
ListerAndLoader
  SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install\_dev\_tutorials\_iknow\_data\_reviews"
  SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
  IF stat != 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
    QUIT }
SourceCountQueries
  WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," total sources",!
  WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByEntities(domId,$LB("road movies")),
  " sources contain 'road movies'",!!
ContainCRCQuery
  WRITE ##class(%iKnow.Queries.CrcAPI).GetCountByEntities(domId,$LB("road movies")),
  " CRCs contain 'road movies'",!
  DO ##class(%iKnow.Queries.CrcAPI).GetByEntities(.result,domId,$LB("road movies"))
    SET i=1
    WHILE $DATA(result(i))
      { WRITE $LISTTOSTRING(result(i),",",1),!
        SET i=i+1 }
```

Using iKnow

47
6.10.1 Limiting by Position

The position of an entity can be Master Concept, Relation, or Slave Concept. By default, the `GetRelated()` method returns all related concepts regardless of position and does not return relations. You can change this default by specifying a macro constant for the 8th parameter (positiontomatch). The available constants are as follows:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$$USEPOSM</td>
<td>1</td>
<td>Master Concepts</td>
</tr>
<tr>
<td>$$$USEPOSR</td>
<td>2</td>
<td>Relations</td>
</tr>
<tr>
<td>$$$USEPOSMR</td>
<td>3</td>
<td>Master Concepts and Relations</td>
</tr>
<tr>
<td>$$$USEPOSS</td>
<td>4</td>
<td>Slave Concepts</td>
</tr>
<tr>
<td>$$$USEPOSSM (the default)</td>
<td>5</td>
<td>Master Concepts and Slave Concepts</td>
</tr>
<tr>
<td>$$$USEPOSSRS</td>
<td>6</td>
<td>Relations and Slave Concepts</td>
</tr>
<tr>
<td>$$$USEPOSSALL</td>
<td>7</td>
<td>Master Concepts, Relations, and Slave Concepts</td>
</tr>
</tbody>
</table>

The following example separates the related master concepts and the related slave concepts. (Note that $$$USEPOSM means that the supplied string is the master concept in the CRC, and the related entities are the slave concepts.)

```plaintext
#include %IKPublic
DomainCreateOrOpen
  SET dname="mydomain"
  IF ##class(%iKnow.Domain).Exists(dname)
  { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
  ELSE
    { SET domoref=##class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment }
  DeleteOldData
  SET stat=domoref.DropData()
  IF stat { GOTO SetEnvironment }
  ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig")
  { SET cfg=##class(%iKnow.Configuration).Open("myconfig")
  }
  ELSE
    SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
    DO cfg.%Save()
  SetListerAndLoader
  SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install."\dev\tutorials\iknow\data\reviews"
  SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,""
  IF stat "= 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
    QUIT }
CountRelatedQuery
  SET road=",\road movies"
  WRITE !,##class(%iKnow.Queries.EntityAPI).GetRelatedCount(domId,$LB(road))
  WRITE " entities are related to ",road,"",!
ListRelatedMastersQuery
```

48 Using iKnow
6.11 Counting Paths

The following example shows the count of paths and the count of sentences for 20 sources. Note that there may be more sentences than paths in a source, or more paths than sentences in a source.

6.12 Listing Similar Sources

The iKnow semantic analysis engine can list which sources are similar to a specified source. Similarity between sources is determined by the number of entities that appear in both sources (the overlap), and the percentage of the source contents that contain overlap.
The **GetSimilar** method can calculate similarity of sources to a specified source. Because of the potentially large number of similar sources, this method is commonly used with a filter to limit the set of sources considered. **GetSimilar** can use your choice of three algorithms, each of which takes an algorithm parameter:

- Basic similarity of items ($$$SIMSRCSIMPLE, the default). Available algorithm parameters are “ent” (entity similarity, the default), “crc” (Concept-Relation-Concept sequence), or “cc” (Concept + Concept pair).
- Using an equivalence set ($$$SIMSRCEQUIVS), with the algorithm parameter being the Id of the equivalence set.
- Using semantic dominance calculations ($$$SIMSRCDOMENTS). The algorithm parameter is a boolean flag that specifies limiting similarity to sources that contain a dominant entity that is also a dominant entity in the specified source.

For each similar source, iKnow returns a list of elements with the following format:

<table>
<thead>
<tr>
<th>srcId</th>
<th>extId</th>
<th>%Overlap</th>
<th>%New</th>
<th>#EntitiesMatchSrc</th>
<th>#OverlappingEntities</th>
<th>#EntitiesThisSrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source ID, an integer assigned by iKnow.</td>
<td>The external ID for the source.</td>
<td>The percentage of the contents of the source that is the same as the match source.</td>
<td>The percentage of the contents of the source that is new. New contents are those that do not match with the match source.</td>
<td>The number of unique entities in the source being matched against this source.</td>
<td>The number of unique entities that are found in both sources.</td>
<td>The number of unique entities in this source.</td>
</tr>
</tbody>
</table>

The following example demonstrates the listing of similar sources. It takes the default similarity algorithm ($$$SIMSRCSIMPLE) and its default algorithm parameter (“ENT”). The similarity display omits the source external IDs:

```plaintext
#Include %IKPublic
DomainCreateOrOpen
   SET dname="mydomain"
   IF (##class(%iKnow.Domain).Exists(dname))
   {  SET domoref=##class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData }
   ELSE
   {  SET domoref=##class(%iKnow.Domain).%New(dname)
       DO domoref.%Save()
      GOTO ListerAndLoader }
DeleteOldData
   SET stat=domoref.DropData()
   IF stat { GOTO ListerAndLoader }
   ELSE    { WRITE "DropData error ",$System.Status.DisplayError(stat)
      QUIT}
ListerAndLoader
   SET domId=domoref.Id
   SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
   SET myloader=##class(%iKnow.Source.Loader).%New(domId)
   SET stat=myloader.SetLister(mylister)
   SET install=$SYSTEM.Util.InstallDirectory()
   SET dirpath=install_"dev	utorials\iknow\data\reviews"
   SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
   IF stat <> 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
      QUIT }
SourceCountQueries
   WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," total sources",!
SimilarEntityQuery
   DO ##class(%iKnow.Queries.SourceAPI).GetByEntities(.result,domId,$LB("movie"))
   SET i=1
   WHILE $DATA(result[i]) {
      SET src = $LISTTOSTRING(result[i],",",1)
      SET srcId = $PIECE(src,"",1)
```
6.13 Summarizing a Source

The iKnow semantic analysis engine can summarize a source text by returning the most relevant sentences. It returns a user-specified number of sentences in the original sentence order, selecting those sentences that have the highest similarity to the overall content of the source text. iKnow determines relevance by calculating an internal relevancy score for each sentence. Sentences that contain concepts that appear many times in the source text are more likely to be included in the summary than those that contain concepts that only appear once in the source text. iKnow considers the overall frequency of each concept, the similarity of each concept to the most frequent concepts in the source, and other factors.

The accuracy of a summary therefore depends on two factors:

- The source text must be large enough to permit meaningful frequency analysis, but not too large. iKnow summarization works best on texts the length of a chapter or article. A book-length text should be summarized chapter-by-chapter.
- The number of sentences in the summary should be a large enough subset of the original for the returned sentences to form a readable summary text. The minimum recommended summary percentage is between 25% and 33%, depending on the contents of the text.

iKnow provides two summary methods:

- **GetSummary()** which returns each sentence of the summary text as a separate result. The sentence Id is returned as the first element of each returned sentence.
- **GetSummaryDirect()** which returns the summary text as a single string. By default, the sentences within this string are separated by an ellipsis: a space followed by three periods, followed by a space. For example, “This is sentence one. ... This is sentence two.” You may specify a different sentence separator, if desired. Because this method concatenates multiple sentences into a single string, it may attempt to create a string longer than the Caché maximum string length. When the maximum string length is reached, Caché sets this method’s isTruncated boolean output parameter to 1, and truncates the remaining text.

The following example goes through the source texts in a domain until it finds one that contains more than 30 sentences. It then uses **GetSummary()** to summarize that source to half of its original sentences:
DO mylister.Init("myconfig")
SET myloader=%class(%iKnow.Source.Loader).%New(domId)
SET stat=my loader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install\"\dev\\tutorials\\iknow\\data\\reviews\"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat = -1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
QUIT }
SourceSentenceTotals
SET numSrcD=%class(%iKnow. Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The domain contains ",numSrcD," sources",!!
SET numSentD=%class(%iKnow. Queries.SentenceAPI).GetCountByDomain(domId)
WRITE "These sources contain ",numSentD," sentences",!!
DO %class(%iKnow. Queries.SourceAPI).GetByDomain(.result,domId)
SentenceCounts
FOR i=1:1:numSrcD {
  SET srcId = $LISTGET(result(i),1)
  SET extId = $LISTGET(result(i),2)
  SET fullref = $PIECE(extId,":",3,4)
  SET fname = $PIECE(fullref,\"",$LENGTH(extId,\")\")
  SET numSentS = %class(%iKnow. Queries.SentenceAPI).GetCountBySource(domId,result(i))
  WRITE fname," has ",numSentS," sentences",!!,
  IF numSentS > 30 {GOTO SummarizeASource }
}
QUIT
SummarizeASource
SET sumlen=$NUMBER(numSentS/2,0)
WRITE "total sentences=",numSentS," summary=",sumlen," sentences",!!
DO %class(%iKnow. Queries.SentenceAPI).GetSummary(.sumresult,domId,srcId,sumlen)
FOR j=1:1:sumlen { WRITE $LISTGET(sumresult(j),2),!! }
WRITE "$EXTRACT END OF ",fname," SUMMARY",!!
QUIT

DomainCreateOrOpen
SET dname="mydomain"
IF %class(%iKnow. Domain).Exists(dname)
  { SET domoref=%class(%iKnow. Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=%class(%iKnow. Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE
  { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
SET domId=domoref.Id
IF %class(%iKnow. Configuration).Exists("myconfig")
  { SET cfg=%class(%iKnow. Configuration).Open("myconfig") }
ELSE
  { SET cfg=%class(%iKnow. Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
    DO cfg.%Save() }
ListerAndLoader
SET mylister=%class(%iKnow. Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=%class(%iKnow. Source.Loader).%New(domId)
SET stat=my loader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install\"\dev\\tutorials\\iknow\\data\\reviews\"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat = -1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
QUIT }
SourceSentenceCounts
SET numSr cD=%class(%iKnow. Queries.SourceAPI).GetCountByDomain(domId)
DO %class(%iKnow. Queries.SourceAPI).GetByDomain(.result,domId)
FOR i=1:1:numSrcD {
  SET srcId = $LISTGET(result(i),1)
  SET extId = $LISTGET(result(i),2)
  SET fullref = $PIECE(extId,":",3,4)
  SET fname = $PIECE(fullref,\"",$LENGTH(extId,\")\")
  SET numSentS = %class(%iKnow. Queries.SentenceAPI).GetCountBySource(domId,result(i))
  IF numSentS > 30 {GOTO SummarizeASource }
}
QUIT
SummarizeASource

Note that $NUMBER is used to assure that the specified summary sentence count is an integer. $LISTGET is used to remove the sentence Id and return just the sentence text.

The following example uses GetSummaryDirect() to return the same summary as a single concatenated string. It then uses $EXTRACT to divide the string into 36-character lines for display purposes:

DomainCreateOrOpen
SET dname="mydomain"
IF %class(%iKnow. Domain).Exists(dname)
  { SET domoref=%class(%iKnow. Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=%class(%iKnow. Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE
  { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
SET domId=domoref.Id
IF %class(%iKnow. Configuration).Exists("myconfig")
  { SET cfg=%class(%iKnow. Configuration).Open("myconfig") }
ELSE
  { SET cfg=%class(%iKnow. Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
    DO cfg.%Save() }
ListerAndLoader
SET mylister=%class(%iKnow. Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=%class(%iKnow. Source.Loader).%New(domId)
SET stat=my loader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install\"\dev\\tutorials\\iknow\\data\\reviews\"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat = -1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
QUIT }
SourceSentenceCounts
SET numSr cD=%class(%iKnow. Queries.SourceAPI).GetCountByDomain(domId)
DO %class(%iKnow. Queries.SourceAPI).GetByDomain(.result,domId)
FOR i=1:1:numSrcD {
  SET srcId = $LISTGET(result(i),1)
  SET extId = $LISTGET(result(i),2)
  SET fullref = $PIECE(extId,":",3,4)
  SET fname = $PIECE(fullref,\"",$LENGTH(extId,\")\")
  SET numSentS = %class(%iKnow. Queries.SentenceAPI).GetCountBySource(domId,result(i))
  IF numSentS > 30 {GOTO SummarizeASource }
}
QUIT
SummarizeASource

Using iKnow
6.14 Querying a Subset of the Sources

iKnow provides filters that allow you to include or exclude sources from a query. You can include or exclude sources based on the source Id, sentence count, the source’s indexed date (the date the source was loaded into iKnow), user-defined metadata characteristics, or the number of dictionary matches.

iKnow supports the combining of multiple filters through logical AND and logical OR operators. For further details, refer to the Filtering chapter of this manual.
Filtering Sources

You can use filters to include or exclude sources supplied to an iKnow query. You can perform a query on all of the loaded sources, or you can filter the list of sources to perform the query on only those sources that meet the criteria of the filter. A filter selects sources based on information associated with the source (metadata), not the actual contents of the source.

7.1 Supported Filters

iKnow supplies a small number of predefined filters, and provides facilities to allow users to easily define their own filters.

- **Source Id**: the SourceIdFilter and ExternalIdFilter allow you to select sources based on the Id of the source. iKnow assigns these values as part of the source indexing process.
- **Random Sources**: the RandomFilter allows you to select a random sampling of the sources in a domain. You specify a percentage of the total sources to include in your filter sample.
- **Sentence Count**: the SentenceCountFilter allows you to select sources based on the minimum and/or maximum number of sentences in the source. iKnow counts the sentences in a source as part of the indexing process.
- **Indexed Date**: the DateIndexed field allows you to select sources based on the date and time when they were indexed by iKnow. iKnow assigns this field value as part of the source indexing process.
- **Metadata**: iKnow allows you to define filters that select sources based on data values that you have associated with a source.
- **Match Count**: the SimpleMatchFilter allows you to select sources based on the minimum and/or maximum number of dictionary matches to the contents of the source.
- **SQL Query**: the SqlFilter allows you to select sources based on the results of an SQL query.

7.1.1 Filter Operators

You assign to each filter one or more equality operators. If the filter is matching against a string value, use the “=” equality operator. If the filter is matching against a numeric value, you can use one or more of the following operators: “=”, “<”, “<=”, “>”, “>=”. Equality operators are always specified as quoted string elements in a list structure. Equality operators are matched against a single value. This is shown in the following example:

```sql
SET filt=##class(%iKnow.Filters.SimpleMetadataFilter).%New(domId,metaId,
"="",today)
```
The BETWEEN operator is matched against a parameter string containing a pair of values that are separated by $$$MDVALSEPARATOR (the semicolon character). This is shown in the following example:

```
SET filt=##class(%iKnow.Filters.SimpleMetadataFilter).%New(domId,metafldId,
        "BETWEEN","yesterday;tomorrow")
```

To associate two or more filters, you can create a GroupFilter and assign it multiple filters (subfilters). In this way you can combine multiple filters to select which sources are supplied to an iKnow query. You assign each GroupFilter either the AND or the OR logical operator. Therefore, to create a compound filter involving both AND and OR logic, you must create a GroupFilter with $$$GROUPFILTERAND logic and a GroupFilter with $$$GROUPFILTEROR logic.

### 7.2 Filtering by the ID of the Source

The most basic source filter is used to limit the sources supplied to a query by providing the Source Id or the External Id of each source that you wish to include in the filtered result set.

#### 7.2.1 By External Id

The ExternalIdFilter includes those sources whose external Ids are listed in a %List structure. Any element in this list that is not a valid External Id, or is a duplicate external Id is silently passed over.

The following example takes .txt files as sources and filters in two sources by External Id. It then lists the details of these two sources:

```plaintext
DomainCreateOrOpen
  SET dname="mydomain"
  IF ##class(%iKnow.Domain).Exists(dname) {
    SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData
  } ELSE {
    SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment
  }
DeleteOldData
  SET stat=domoref.DropData()
  IF stat {
    GOTO SetEnvironment
  } ELSE {
    WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT
  }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig") {
    SET cfg=##class(%iKnow.Configuration).Open("myconfig")
  } ELSE {
    SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en","",1))
    DO cfg.%Save()
  }
ListerAndLoader
  SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET dirpath=$SYSTEM.Util.InstallDirectory()
  SET dirpath=dirpath_"dev\tutorialsknow\data\reviews"
  SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
  IF stat != 1 {
    QUIT
  }
DefineExtIdFilter
  SET txt1=":FILE:"_dirpath_"\review1.txt"
  SET txt3=":FILE:"_dirpath_"\review3.txt"
SourceCountQueries
  SET numSrcD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
ApplyExtIdFilter
  SET numSrcFD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filt)
  WRITE "The external Id filter included ",numSrcFD," of these sources!
  DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId,1,20,filt)
    FOR i=1:1:numlist {
      IF $DATA(result(i)) {
        SET extId = $LISTGET(result(i),2)
```
7.2.2 By Source Id

The SourceIdFilter includes those sources whose source Ids are listed in a %List structure. Source Ids are integers. They can be listed in any order. Any element in this list that is not a valid source Id, or is a duplicate source Id is silently passed over.

The following example takes SQL records as sources and filters in several sources by Source Id. In this example, five source Ids are specified, but only three of these correspond to records in the table. Therefore, the total source count is 3:

```
7.3 Filtering a Random Selection of Sources

You can use the %iKnow.Filters.RandomFilter to select a sample of your source. You specify a percentage (as a fractional number between 0 and 1), and this filter returns the corresponding percentage of the indexed sources in the specified domain. For example, a value of ".5" means that 50% of the sources in the domain will be included in the filtered result. Halves are rounded up, so 50% of 5 sources is 3 sources. This filter selects the requisite number of sources randomly to provide a random sample of the whole set of indexed sources.

The following example randomly selects 50% of the sources. You can run this example repeatedly to demonstrate that different sources are sampled:
```
DomainCreateOrOpen
  SET dname="mydomain"
  IF (##class(%iKnow.Domain).Exists(dname))
    { SET domoref=##class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData }
  ELSE { SET domoref=##class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment } 
DeleteOldData
  SET stat=domoref.DropData()
  IF stat { GOTO SetEnvironment }
  ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
      QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
  ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
      DO cfg.%Save() }
ListerAndLoader
  SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET stat=myloader.SetLister(mylister)
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install_"dev\tutorials\iknow\data\reviews"
  SET lst=myloader.ProcessList(dirpath,$LB("txt"),0,"")
  IF stat = 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
      QUIT }
DefineAFilter
  SET filt=##class(%iKnow.Filters.RandomFilter).%New(domId,.5)
SampledSourceQueries
  SET numSrcD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
  SET numSrcFD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filt)
  WRITE "Of these ",numSrcD," sources ",numSrcFD," were sampled:",
  DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId,1,20,filt)
  SET i=1
  WHILE $DATA(result(i))
    SET extId = $LISTGET(result(i),2)
    SET fname = $PIECE($PIECE(extId,":\",3,4),"\",$LENGTH(extId,"\")
    WRITE "sample #",i," is ",fname,
    SET i=i+1 }

7.4 Filtering by Number of Sentences

The following example filters out sources that contain less than 7 sentences. It returns the total number of sources, then uses the filter to return the total number of sources containing 7 or more sentences, then uses the filter again to return the number of sentences in each of these sources:

DomainCreateOrOpen
  SET dname="mydomain"
  IF (##class(%iKnow.Domain).Exists(dname))
    { SET domoref=##class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData }
  ELSE { SET domoref=##class(%iKnow.Domain).%New(dname)
      DO domoref.%Save()
      GOTO SetEnvironment } 
DeleteOldData
  SET stat=domoref.DropData()
  IF stat { GOTO SetEnvironment }
  ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat)
      QUIT }
SetEnvironment
  SET domId=domoref.Id
  IF ##class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
  ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
      DO cfg.%Save() }
ListerAndLoader
  SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
  DO mylister.Init("myconfig")
  SET myloader=##class(%iKnow.Source.Loader).%New(domId)
  SET install=$SYSTEM.Util.InstallDirectory()
  SET dirpath=install_"dev\tutorials\iknow\data\reviews"
  SET lst=myloader.ProcessList(dirpath,$LB("txt"),0,"")
  IF stat = 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
      QUIT }
IF stat '= 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)  
QUIT }

DefineAFilter
SET filt=#class(%iKnow.Filters.SentenceCountFilter).%New(domId)
SET nsent=7
DO filt.MinSentenceCountSet(nsent)
SourceSentenceQueries
SET numSrcD=#class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The domain contains ",numSrcD," sources!
SET numSrcFD=#class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filt)
WRITE "Of these sources ",numSrcFD," contain ",nsent," or more sentences!
SentencesPerSource
DO #class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId,1,20,filt)
SET i=1
WHILE $DATA(result(i)) {  
SET extId = $LISTGET(result(i),2)
SET fname = $PIECE($PIECE(extId,":",3,4),"\"",$LENGTH(extId,"\")
SET numSentS = #class(%iKnow.Queries.SentenceAPI).GetCountBySource(domId,result(i))
WRITE fname," has ",numSentS," sentences!
SET i=i+1 }

To filter using both minimum and maximum number of sentences, invoke both instance methods. The following filter selects those sources containing between 10 and 25 sentences (inclusive):

MinMaxFilter
SET min=10
SET filt=#class(%iKnow.Filters.SentenceCountFilter).%New(domId)
DO filt.MinSentenceCountSet(min)
DO filt.MaxSentenceCountSet(min+15)

7.5 Filtering by Indexing Date

Every iKnow source is assigned the DateIndexed metadata field. The value of this field is the date and time that a source was indexed by iKnow, in Coordinated Universal Time format (UTC) represented in $HOROLOG format. This is the same as the $ZTIMESTAMP time, except that DateIndexed does not include fractional seconds.

You can create a filter using DateIndexed to include or exclude sources based on when iKnow loaded the source. You can filter using a specific date and time, or filter for a specific date, which encompass all time values within that date. You can use BETWEEN logic to filter for a range of dates.

The following example filters for sources loaded today:

DomainCreateOrOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname)) {  
SET domoref=#class(%iKnow.Domain).Open(dname)  
GOTO DeleteOldData }
ELSE {  
SET domoref=#class(%iKnow.Domain).%New(dname)  
DO domoref.%Save()  
GOTO SetEnvironment }
DeleteOldData
SET oldstat=domoref.DropData()
IF oldstat ( GOTO SetEnvironment )
ELSE { WRITE "DropData error ",$System.Status.DisplayError(oldstat)  
QUIT }
SetEnvironment
SET domId=domoref.Id
IF #class(%iKnow.Configuration).Exists("myconfig") {  
SET cfg=#class(%iKnow.Configuration).Open("myconfig")  
ELSE {  
SET cfg=#class(%iKnow.Configuration).%New("myconfig"),0,$LISTBUILD("en"),","1)  
DO cfg.%Save() }
ListerAndLoader
SET mylister=#class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=#class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_\"dev\"\"tutorials\\"iKnow\\data\\reviews"  
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat '= 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)  
QUIT }
DefineAFilter
SET tday = $PIECE($ZTIMESTAMP,",",1)
7.6 Filtering by User-defined Metadata

In iKnow, data is the contents of a source that iKnow processes and indexes. In iKnow, metadata can be any data associated with a source that is not iKnow indexed data. You use iKnow metadata to identify iKnow data. A metadata filter uses the value of a metadata field to determine which sources to supply to a query.

Note: The iKnow definition of “metadata” describes how data is used, not the intrinsic nature of the data. This concept differs somewhat from the way this word is used elsewhere in Caché software.

iKnow provides a default metadata management system that is independent of the query APIs. The %iKnow.Queries.MetadataAPI class and accompanying %iKnow.Filters.SimpleMetadataFilter provide implementations for basic metadata filtering. If you wish to implement a custom Metadata API, you should implement (at least) the %iKnow.Queries.MetadataI interface and register your class as the “MetadataAPI” domain parameter: DO domain.SetParameter(“MetadataAPI”, “Your.Metadata.Class”). The example that follows uses the %iKnow.Filters.SimpleMetadataFilter class.

In Caché SQL, each record of an SQL table constitutes an iKnow source. Through the ProcessList() method (for small numbers of records) or AddListToBatch() method (for large numbers of records), you define the Lister parameters:

- You define the RowID field as a component of the the iKnow external Id. iKnow also generates a source Id for each row as a unique integer; this iKnow source Id is completely independent of the RowId or other SQL identifier values.
- You define a field (or fields) that contain a string of text as a data field to be indexed as iKnow data.
- You define a field (or fields) as an iKnow metadata field. iKnow can use the values of this metadata field to select sources for an iKnow query.

Note that it is possible to specify the same field as both one of the data fields and as a metadata field. You can optionally also define metakey fields that correspond to the metadata fields.

This is shown in the following example. The Cinema.Review table contains fields for a film number, a review score (from 1 to 5), and a review text. In this example, the ReviewScore is used as a metadata field. This metadata field is used in three filters: two equality filters, which filter for ReviewScore=4 and ReviewScore=5, and a BETWEEN filter that filters for ReviewScores between 4 and 5 (inclusive). This example uses the DropData(1) method, because DropData() with no argument does not delete metadata. Also note that the AddField() method must be invoked before listing and loading the data.
ELSE { SET domoref=##class(%iKnow.Domain).%New(dname) DO domoref.%Save() GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",$$System.Status.DisplayError(stat) QUIT }
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
   SET cfg=##class(%iKnow.Configuration).Open("myconfig")
ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),","1)
   DO cfg.%Save() }
CreateListerAndLoader
SET mylister=##class(%iKnow.Source.SQL.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
QueryBuild
SET myquery="SELECT ID AS UniqueVal,Film,ReviewScore,ReviewText FROM Cinema.Review"
SET idfld="UniqueVal"
SET grpfld="Film"
SET dataflds=$LB("ReviewText")
SET metaflds=$LB("ReviewScore")
AddMetaField
SET val=##class(%iKnow.Queries.MetadataAPI).AddField(domId,"ReviewScore","=","<",">","BETWEEN"),$$MDDTNUMBER
UseListerAndLoader
SET stat=myloader.SetLister(mylister)
SET stat=myloader.ProcessList(myquery,idfld,grpfld,dataflds,metaflds)
IF stat '= 1 { WRITE "Loader error ",$$System.Status.DisplayError(stat) QUIT }
QueryLoadedSources
SET numsrc = ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "number of sources=",numsrc;
ApplyFilter
SET filt4=##class(%iKnow.Filters.SimpleMetadataFilter).%New(domId,"ReviewScore", "=".4)
SET numSrcF4=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filt4)
WRITE "Of these ",numsrc," sources ",numSrcF4," had a four-star review","!
SET filt5=##class(%iKnow.Filters.SimpleMetadataFilter).%New(domId,"ReviewScore", "=".5)
SET numSrcF5=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filt5)
WRITE "Of these ",numsrc," sources ",numSrcF5," had a five-star review","!
SET filtB=##class(%iKnow.Filters.SimpleMetadataFilter).%New(domId,"ReviewScore", "BETWEEN","4;5")
SET numSrcFb=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId,filtb)
WRITE "Of these ",numsrc," sources ",numSrcFb," had between 4 and 5 stars"!

7.7 Filtering by SQL Query

The %iKnow.Filters.SqlFilter class allows you to select SQL sources based on the results of an SQL query. This query can select on any of the following fields:

- **SourceId**: the (internal) Source ID of the sources to be selected.
- **ExternalId**: the full External ID of the sources to be selected.
- **IdField** and **GroupField**: the two columns used together as identifiers when adding the sources to the domain: Local Reference (IdField) and Group Name (GroupField). See also %iKnow.Source.SQL.Lister.

Note that these result column names are case sensitive.

For example, the following filter selects the source with the specified SourceId:

```plaintext
ZNSPACE "Samples"
DomainCreateOrOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
   { SET domoref=#class(%iKnow.Domain).Open(dname) DO DeleteOldData } ELSE
   { SET domoref=#class(%iKnow.Domain).%New(dname) DO domoref.%Save() GOTO SetEnvironment }
```
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",$System.Status.DisplayError(stat) QUIT }

SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") { 
SET cfg##class(%iKnow.Configuration).Open("myconfig") 
ELSE { SET cfg##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1) 
DO cfg.%Save() }

CreateListerAndLoader
SET filister##class(%iKnow.Source.SQL.Lister).%New(domId)
DO filister.Init("myconfig")
SET myloader##class(%iKnow.Source.Loader).%New(domId)

QueryBuild
SET myquery="SELECT ID AS UniqueVal,Film,ReviewText FROM Cinema.Review"
SET idfld="UniqueVal"
SET grpfld="Film"
SET dataflds=$LB("ReviewText")
SET metaflds=$LB("Film")
SET mkeys=$LB("Film")

UseLister
SET stat=filister.AddListToBatch(myquery,idfld,grpfld,dataflds,metaflds)

UseLoader
SET stat=myloader.ProcessBatch()

QueryLoadedSources
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," sources",!!
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId)
WRITE "Unfiltered results:"!
SET i=1
WHILE $DATA(result(i)) {
WRITE $LISTTOSTRING(result(i)),!
SET i=i+1 }

FilterSources
SET filter##class(%iKnow.Filters.SqlFilter).%New(domId,"SELECT 6 AS SourceId")
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.fresult,domId,0,0,filter)
WRITE !,"Filtered results:"!
SET j=1
WHILE $DATA(fresult(j)) {
WRITE $LISTTOSTRING(fresult(j)),!
SET j=j+1 }

The following filter selects sources by ExternalId:

SET filter##class(%iKnow.Filters.SqlFilter).%New(domId,"SELECT ",_LIST(result(1),2)_" AS ExternalId")
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(domId,0,0,filter)
WRITE result

7.8 Filter Modes

The FilterMode argument specifies what statistical reprocessing should be performed after applying a filter. If no reprocessing is performed, the filter is applied but the frequency and spread statistics are the values calculated for the item before filtering and the sort sequence remains unchanged. The following are the available filter modes:
The default is $$$FILTERONLY.

Refer to Constants in the “iKnow Implementation” chapter for use of $$$ macros.

### 7.9 Using Multiple Filters

The following example corresponds to the Boolean expression "(filter1 AND !(filter2 OR filter3))":

```plaintext
#Include %IKPublic
SET domoref=##class(%iKnow.Domain).%New("MyDomain")
DO domoref.%Save()
SET domId=domoref.Id
/* . . . */
Create3Filters
/* . . . */
GroupFilters
SET group1=##class(%iKnow.Filters.GroupFilter).%New(domId,$$$GROUPFILTERAND,0)
SET group2=##class(%iKnow.Filters.GroupFilter).%New(domId,$$$GROUPFILTEROR,1)
DO group1.AddSubFilter(filter1)
DO group2.AddSubFilter(filter2)
DO group2.AddSubFilter(filter3)
DO group1.AddSubFilter(group2)
```
Equivalence Sets

An equivalence set is a set of paired indexed terms (Concepts or Relations) that you wish to define as equivalent. To define an equivalence set, you first create the set, assigning it an internal Id (an integer value). You then populate the set with pairs of indexed terms. The equivalence set is a separate entity that holds these pairings. Defining an equivalence set does not in any way change the entities themselves; it creates a table of paired entity Ids for future use. You can create multiple equivalence sets for use in different contexts.

For example, you are querying iKnow texts that contain medical notes to determine the accuracy of patients’ self-diagnosis. In this context, the terms ‘asleep’, ‘comatose’, ‘unconscious’, ‘unresponsive’, ‘uncommunicative’, and ‘silent’ are a meaningful equivalence set. In other medical contexts, these words are not at all equivalent.

Equivalence sets are defined using the methods of the %iKnow.Queries.EquivAPI class (and the corresponding EquivQAPI and EquivWSAPI classes). Use the CreateSet() method, then populate the equivalence set using the DefineTerms() method. These methods are described in the InterSystems Class Reference.

The following example creates an equivalence set with an Id value of 7, and populates it with a list of equivalence value Ids:

```plaintext
DomainCreateOrOpen
  SET dname="mydomain"
  IF (##class(%iKnow.Domain).Exists(dname))
  { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat
  { WRITE "DropData error ",System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
SET domId=domoref.Id
ListerAndLoader
  DO myloader.SetLister(mylister)
  SET install=SYSTEM.Util.InstallDirectory()
  SET dirpath=install_dev\tutorial\iknow\data\reviews
  IF stat
    { WRITE "Loader error ",System.Status.DisplayError(stat)
      QUIT }
CreateEquivSet
  WRITE "CreateSet status: ",%iKnow.Queries.EquivAPI.CreateSet(domId,7),!!
PopulateEquivSet
  SET equlvs = LISTBUILD(1,65,202,264,267,278)
  SET ptr=0
  WHILE $LISTNEXT(equvs,ptr,val)
    DO %iKnow.Queries.EquivAPI.DefineTerms(domId,7,65,val)
  QUIT
QueryEquivSet
  DO %iKnow.Queries.EquivAPI.GetTerms(.result,domId,7,65)
```

Using iKnow
You can query either side of an equivalence pair. Equivalence pairs are logically independent. In other words, if A=B and B=C, if you query B you will get A and C, but if you query A you will get B but not C.

You can delete an equivalence pair from an equivalence set using the RemoveTerms() method, as shown in the following example:

```
DomainCreateOrOpen
SET dname="mydomain"
IF (##class(%iKnow.Domain).Exists(dname))
  { SET domoref=##class(%iKnow.Domain).Open(dname)
    GOTO DeleteOldData }
ELSE
  { SET domoref=##class(%iKnow.Domain).%New(dname)
    DO domoref.%Save()
    GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE
  { WRITE "DropData error ",$System.Status.DisplayError(stat)
    QUIT }
SetEnvironment
SET domId=domoref.Id
ListerAndLoader
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_"dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat '=' 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
    QUIT }
CreateEquivSet
WRITE "CreateSet status: ",##class(%iKnow.Queries.EquivAPI).CreateSet(domId,7),!!
PopulateEquivSet
SET equivs = $LISTBUILD(1,65,202,264,267,278)
SET ptr=0
WHILE $LISTNEXT(equivs,ptr,val) {
  SET err = ##class(%iKnow.Queries.EquivAPI).DefineTerms(domId,7,65,val)
  IF err '=' 1 { WRITE "equivalence populate error" QUIT}
}
SET onetime=0
QueryEquivSet
DO ##class(%iKnow.Queries.EquivAPI).GetTerms(.result,domId,7,65)
SET i=1
WHILE $DATA(result(i),list) {
  WRITE "$LISTGET(list)
  WRITE ",
  WRITE "$LISTGET(list,2),"'
  WRITE " freq: ",$LISTGET(list,3)
  WRITE " spread: ",$LISTGET(list,4),!
  SET i=i+1 }
WRITE "End of equivalence set"
```

Using iKnow
9

Dominance and Proximity

The iKnow semantics package consists of two classes: %iKnow.Semantics.DominanceAPI and %iKnow.Semantics.ProximityAPI. These classes with their parameters and methods are described in the InterSystems Class Reference.

This chapter describes:

- Semantic Dominance
- Semantic Proximity

9.1 Semantic Dominance

Semantic dominance is the overall importance of an entity within a source. iKnow determines semantic dominance by performing the following tests and obtaining statistical results:

- The number of times that an entity appears in a source (the frequency).
- The number of times that each component word of an entity appears in a source.
- The number of words in the entity.
- The type of entity (concept or relation).
- The diversity of entities in the source.
- The diversity of component words in the source.

iKnow generates these values when the sources are loaded as part of iKnow indexing. It combines these values to produce a dominance score for each entity. The higher the dominance score, the more dominant the entity is within the source.

Note: iKnow assigns the entities in each source a dominance score. These scores are specific to that source. Dominance score values can be used for comparing the relative dominance of entities within that source. Dominance score values cannot be used for comparing the relative dominance of an entity in different sources.

For example, for the concept “cardiovascular surgery” to be semantically dominant in a document, a statistical analysis would be performed. It determines that this concept appears 20 times in the source. A dominant concept is not the same as a top concept. In this source the concepts “doctor” (60 times), “surgery” (50 times), “operating room” (40 times), and “surgical procedure” (30 times) are far more common. However, the component words of “cardiovascular surgery” appear over twice as many times as the concept itself: “cardiovascular” (50 times) and “surgery” (80 times), which lends support to this being a dominant concept. In contrast, the concept “operating room” appears 40 times, but its component words
appear barely more often than the concept: “operating” 60 times and “room” only 45 times. This indicates that this source is much more concerned with cardiovascular matters and surgery than it is with rooms.

iKnow gives these frequency counts greater or lesser weight based on the number of words in the original entity and whether that entity is a concept or a relation. (There is a much smaller number of commonly-occurring relations than concepts.) However, to determine how dominant a concept really is, iKnow has to compare it to the total number of concepts in the source. If 5% of the concepts in the source contain the words “cardiovascular” and “surgery”, and these words do not combine in other concepts nearly as frequently as they do together, we know that these words not only appear frequently in the source, but that the source does not have a wide range of subject matter. If, however, the source contains a nearly equal occurrence of the word “surgery” in concepts with “hand”, “kidney” and “brain” and the word “cardiovascular” appears nearly as often with the words “exercise” and “diet” it is apparent that the source contains a wide range of subject matter. The concept “cardiovascular surgery” and its component words may appear more frequently than others, but may not significantly dominate the subject matter of the source.

By performing these statistical calculations, iKnow can determine the dominant concepts in a source — the subjects that are of greatest interest to you. iKnow performs this analysis without using an external reference corpus (such as pre-existing table of the relative frequency of words in a “typical” medical text). iKnow determines dominance using only the contents of the actual source text, and thus can be used on sources on any topic without any prior knowledge of the subject matter.

9.1.1 Concepts of Semantic Dominance

The following are the key elements of semantic dominance:

- **Profile**: the counts of elements that are used to calculate a dominance score.
- **Typical**: a typical source is a source in which the dominant entities in that source are most similar to the dominant elements of the group of sources. This is the opposite of Breaking.
- **Breaking**: a breaking source is a source in which the dominant entities in that source are least similar to the dominant elements of the group of sources. This is the opposite of Typical. For example, a breaking news story would likely be least similar to the dominant entities in all of the news stories from the previous month.
- **Overlap**: the number of occurrences of an entity in different sources.
- **Correlation**: a comparison of the entities in a source with a list of entities, returning a correlation percentage for each source.

9.1.2 Semantic Dominance Examples

This chapter describes and provides examples for the following semantic dominance queries:

- Dominance profile counts for the domain
- Dominance scores for the domain
- Dominance score for a specified entity
- Overlap counts between sources
- Typical sources
- Breaking sources
- Sources by correlation
9.1.2.1 Dominance Profile Counts

The following example uses the `GetProfileCountByDomain()` method to return the total count of unique values for an entity type in the specified domain. The available entity types are: 0=concept ($$$SDCONCEPT or $$$SDENTITY); 1=relation ($$$SDRELATION); 2=CRC ($$$SDCRC); 4=aggregate ($$$SDAGGREGATE, the default) which is the total of all concepts, relations, and CRCs.

```plaintext
#Include %IKPublic
ZNSPACE "Samples"
DomainCreateOrOpen
   SET dname="mydomain"
   IF (##class(%iKnow.Domain).Exists(dname))
      SET domoref=##class(%iKnow.Domain).Open(dname)
      GOTO DeleteOldData
   ELSE
      SET domoref=##class(%iKnow.Domain).%New(dname)
      GOTO SetEnvironment
   ENDIF
DeleteOldData
   SET stat=domoref.DropData()
   IF stat { GOTO SetEnvironment }
   ELSE
      WRITE "DropData error ",$System.Status.DisplayError(stat)
      QUIT
   ENDIF
SetEnvironment
   SET domId=domoref.Id
   IF ##class(%iKnow.Configuration).Exists("myconfig")
      SET cfg=##class(%iKnow.Configuration).Open("myconfig")
   ELSE
      SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
      DO cfg.%Save()
   ENDIF
ListerAndLoader
   SET mylister=##class(%iKnow.Source.Global.Lister).%New(domId)
   DO mylister.Init("myconfig")
   SET myloader=##class(%iKnow.Source.Loader).%New(domId)
   SET stat=myloader.SetLister(mylister)
LoadGlobalsTwoSubscriptLevels
   FOR i=1:1:90000
      SET gbl="^Aviation.NarrativeS("i_")"
      SET stat=myloader.ProcessList(gbl,1,1)
      QUIT
   ENDIF
ProfileCounts
   WRITE ##class(%iKnow.Semantics.DominanceAPI).GetProfileCountByDomain(domId,$$$SDCONCEPT)," total concepts",
   WRITE ##class(%iKnow.Semantics.DominanceAPI).GetProfileCountByDomain(domId,$$$SDRELATION)," total relations",
   WRITE ##class(%iKnow.Semantics.DominanceAPI).GetProfileCountByDomain(domId,$$$SDCRC)," total CRCs",
   WRITE ##class(%iKnow.Semantics.DominanceAPI).GetProfileCountByDomain(domId)," aggregate total"
```

9.1.2.2 Dominance Scores

The `GetProfileByDomain()` method returns the dominant profile entities. For each entity returns the following:

- The entity source Id, a unique integer.
- The entity value (the external Id). Because this method can return not only single concepts, but CRCs and paths made up of multiple concepts and relations, this value is returned as a list structure.
- The entity type, an integer code: 0=concept ($$$SDCONCEPT or $$$SDENTITY, the default); 1=relation ($$$SDRELATION); 2=CRC ($$$SDCRC); 3=path ($$$SDPATH); 4=aggregate ($$$SDAGGREGATE) returns first all concepts, then all relations, then all CRCs.
- The calculated dominance value. Dominance values for concepts and relations are always integers. Dominance values for CRCs may be integers, or fractional numbers, such as 3480.5, that are half of an integer.

The following example uses the `GetProfileByDomain()` method to return the dominant profile entities for the `^Aviation.NarrativeS` globals when the second subscript is 1. For example `^Aviation.NarrativeS(42,1)`. For each entity it returns the value, the source Id, and the dominance score. In this example, all returned entities are type 0 (concepts), so displaying the type is omitted.
9.1.2.3 Dominance Score for a Specified Entity

iKnow uses the GetDomainValue() method to return the dominance value for a specified entity. You specify the entity by its entity Id (a unique integer), and specify the entity type by a numeric code. The default entity type is 0 (concept).

A single set of unique entity Ids is used for concepts and relations; therefore, no concept has the same entity Id as a relation. A separate set of unique entity Ids is used for CRCs; therefore, a CRC may have the same entity Id as a concept or a relation. This numbering is entirely coincidental; there is no connection between the entities.

The following example takes the top 12 entities and determines the dominance score for each entity. As one can see from the results of this example, the top (most frequently occurring) entities do not necessarily correspond to the entities with the highest dominance scores:
{ WRITE "DropData error ",System.Status.DisplayError(stat)
QUIT }

SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
SET cfg=##class(%iKnow.Configuration).Open("myconfig")
} ELSE {
SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
DO cfg.%Save()
}

ListerAndLoader
SET mylister=##class(%iKnow.Source.Global.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)

LoadGlobalsTwoSubscriptLevels
FOR i=1:1:90000
SET gbl="^Aviation.NarrativeS("_i_")"
SET stat=myloader.ProcessList(gbl,1,1)
QUIT }

TopEntitiesDominanceScores
DO ##class(%iKnow.Queries.EntityAPI).GetTop(.result,domId,1,12)
SET i=1
WHILE $DATA(result(i)) {
SET topstr=$LISTTOSTRING(result(i),",",1)
SET topid=$PIECE(topstr,",",1)
SET val=$PIECE(topstr,",",2)
SET spc=25-$LENGTH(val)
WRITE val
WRITE $JUSTIFY("top=",spc),i
WRITE " dominance=
WRITE ##class(%iKnow.Semantics.DominanceAPI).GetDomainValue(domId,topid,0),!
SET i=i+1
WRITE "Top ",i-1," entities and their dominance scores"

9.1.2.4 Overlap Between Sources

The following example uses the GetOverlap() method to return the overlap score for each entity. This score is the count of overlapping occurrences in all of sources in the domain for that entity. Note that before you can invoke GetOverlap(), you must first invoke BuildOverlap(). When displaying the results, type=0 (concepts) for all of the returned values in this example and is therefore omitted from the display; the $JUSTIFY function is used in this example to align display of the overlap counts:

#include %IKPublic
ZNSPACE "Samples"
DomainCreateOrOpen
SET dname="mydomain"
IF ##class(%iKnow.Domain).Exists(dname)
{ SET domoref=##class(%iKnow.Domain).Open(dname)
GOTO DeleteOldData }
ELSE
{ SET domoref=##class(%iKnow.Domain).%New(dname)
DO domoref.%Save()
GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE { WRITE "DropData error ",System.Status.DisplayError(stat)
QUIT }

SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
SET cfg=##class(%iKnow.Configuration).Open("myconfig")
} ELSE {
SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),"",1)
DO cfg.%Save()
}

ListerAndLoader
SET mylister=##class(%iKnow.Source.Global.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
QUIT }
FOR i=1:1:90000
SET gbl="^Aviation.NarrativeS("_i_")"
SET stat=myloader.ProcessList(gbl,1,2)
QUIT }
SourceSentenceQueries
SET numSrcD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
9.1.2.5 Typical Sources

The following example lists the ten most typical sources with their dominance scores. In order to retrieve typical sources you must issue the GetOverlap() method, followed by the FindMostTypicalSources() and GetTypicalSources() methods.

9.1.2.6 Breaking Sources

Breaking sources are those sources that differ significantly from the other sources in the domain (or filtered subset of the domain). In order to retrieve breaking sources you must issue the GetOverlap() method, followed by the FindBreakingSources() and GetBreakingSources() methods.
In the following program, after identifying each breaking source, the program uses `GetBySource()` to return the dominant entities for each breaking source. For each entity it returns:

- the entity Id, which is a unique integer for that entity type. However, because `GetBySource()` returns multiple entity types (concepts, CRCs, paths), this entity Id is only unique and meaningful within the specified type.
- the entity value, which can be a single concept, or a list of concepts and relations. This list can be a CRC or a path. *(To simplify the display, the entity value is here commented out.)*
- the entity type: 0 for a concept, 1 for a relation, 3 for a CRC, or an odd number > 3 for a path. The path value corresponds to the number of elements in the path. If the number of elements in the path is an even number, the entity type is the next larger odd number.
- the dominance value, which is a calculated positive number greater than 1. The higher the dominance value, the more dominant the entity is within that source. The dominance value for a concept or relation is an integer; the dominance value for a CRC or path may contain a decimal fraction.

```plaintext
#Include %IKPublic
ZNSPACE "Samples"
DomainCreateOrCreateOpen
SET dname="mydomain"
IF (#class(%iKnow.Domain).Exists(dname))
{ SET domoref=#class(%iKnow.Domain).Open(dname) 
  GOTO DeleteOldData }
ELSE
{ SET domoref=#class(%iKnow.Domain).%New(dname) 
  DO domoref.%Save() 
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment } ELSE
{ WRITE "DropData error ",$System.Status.DisplayError(stat)
QUIT }
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
  SET cfg=#class(%iKnow.Configuration).Open("myconfig") }
ELSE { SET cfg=#class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en","",1)
  DO cfg.%Save() }
ListerAndLoader
SET mylister=#class(%iKnow.Source.Global.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=#class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
LoadGlobalsTwoSubscriptionLevels
FOR i=1:1:90000 
  SET gbl="^Aviation.NarrativeS(_i_)"
  SET stat=myloader.ProcessList(gbl,1,1)
    QUIT }
SourceCountQueries
WRITE #class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," total sources",!!
DominanceBreakingSources
SET datstat = #class(%iKnow.Semantics.DominanceAPI).BuildOverlap(domId)
IF datstat !=-1 (WRITE "BuildOverlap() error " QUIT)
SET datstat = #class(%iKnow.Semantics.DominanceAPI).FindBreakingSources(domId)
IF datstat !=-1 (WRITE "FindBreakingSources() error " QUIT)
DO #class(%iKnow.Semantics.DominanceAPI).GetBreakingSources(.profresult,domId,1,10)
SET j=1,k=1
WHILE $DATA(profresult(j),srclist) {
  SET src =$LISTGET(srclist)
  WRITE !,"Source id: ",src
  DO #class(%iKnow.Semantics.DominanceAPI).GetBySource(.srcresult,domId,src,1,10)
  WHILE $DATA(srcresult(k),list) {
    WRITE "id:",$LISTGET(list)
    /* WRITE " values:",$LISTTOSTRING($LISTGET(list,2)) */
    WRITE " type:",$LISTGET(list,3)," dominance:",$LISTGET(list,4),!
    SET k=k+1 }
  SET k=1
  SET j=j+1 }
WRITE !!,"Printed ",j-1," breaking sources"
9.1.2.7 Sources By Correlation

GetSourcesByCorrelation() returns those sources that correlate to a list of entities, calculating the a percentage of correlation between the source’s entities and the list entities. A correlation percentage of 1 (100%) means that there are as many correlations in the source as there are listed entities; it does not mean that every listed entity appears in that source. For this reason, it is possible to return a correlation percentage greater that 1. Sources with 0% correlation are not listed.

In order to retrieve sources by correlation you must issue the GetOverlap() method, followed by the GetSourcesByCorrelation() method. You can supply a filter to GetOverlap() to limit the set of sources to be tested for correlation.

The following example tests sources by comparing them to a list of concepts that refer to helicopters. It returns the source ID, the external Id, and the percentage of correlation:

```plaintext
#Include %IKPublic
ZNSPACE "Samples"
DomainCreateOrOpen
SET dname="mydomain"
IF (##class(%iKnow.Domain).Exists(dname))
    { SET domoref=##class(%iKnow.Domain).Open(dname)
        GOTO DeleteOldData }
ELSE
    { SET domoref=##class(%iKnow.Domain).%New(dname)
        domoref.%Save()
        GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE    { WRITE "DropData error ",$System.Status.DisplayError(stat)
            QUIT}
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig")
    { SET cfg=##class(%iKnow.Configuration).Open("myconfig") }
ELSE { SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,$LISTBUILD("en"),",",1)
        DO cfg.%Save() }
ListerAndLoader
SET mylister=##class(%iKnow.Source.Global.Lister).%New(domId)
    DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
IF stat '= 1 { WRITE "SetLister error ",$System.Status.DisplayError(stat)
            QUIT }
FOR i=1:1:90000 {
    SET gbl="^Aviation.NarrativeS(_i_)"
    SET stat=myloader.ProcessList(gbl,1,2)
                QUIT }
SourceSentenceQueries
SET numSrcD=##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The domain contains ",numSrcD," sources","!
SET numSentD=##class(%iKnow.Queries.SentenceAPI).GetCountByDomain(domId)
WRITE "These sources contain ",numSentD," sentences",!!!
SourcesByCorrelation
DO ##class(%iKnow.Semantics.DominanceAPI).BuildOverlap(domId)
SET heliterms=$LB(796,1048,1706,2484,2571,2653,3284,3987,4037,4038,4054,4957)
DO ##class(%iKnow.Semantics.DominanceAPI).GetSourcesByCorrelation(.result,domId,heliterms)
DisplaySourcesByCorrelation
SET i=1
WHILE #$DATA(result(i)) {
    WRITE $LISTTOSTRING(result(i),","),! 
    SET i=i+1 }
```

9.2 Semantic Proximity

Semantic proximity is a calculation of the semantic “distance” between two entities within a sentence. The higher the proximity integer, the closer the entities.

As a demonstration of this semantic distance, given the sentence:

```
Using iKnow
```
The giraffe walked with long legs to the base of the tree, then stretched his long neck up to reach the lowest leaves.

Semantic proximity is calculated for each entity in each sentence, then these generated proximity scores are added together producing an overall proximity score for each entity for the entire set of source texts. For example, given the sentences:

"The giraffe walked with long legs to the base of the tree, then stretched his long neck up to reach the lowest leaves. Having eaten, the giraffe bent his long legs and stretched his long neck down to drink from the pool."

the proximity of the concept “giraffe” might be as follows: long legs=128, long neck=67, base=42, tree=32, pool=32, lowest leaves=21.

Entity proximity is commutative; this means that the proximity of entity1 to entity2 is the same as the proximity of entity2 to entity1. iKnow does not calculate a semantic proximity of an entity to itself. For example, the sentence “The boy told a boy about another boy.” would not generate any proximity scores, but the sentence “The boy told a younger boy about another small boy.” generates the proximity scores younger boy=64, small boy=42. If the same entity appears multiple times in a sentence, the proximity score is additive. For example, the proximity for the concept “girl” in the sentence “The girl told the boy about another boy.” is boy=106, the total the two proximity scores 64 and 42.

The following example uses the GetProfileForEntity() method to return the proximity of the concept “movie” to other concepts in sentences in all of the sources in the domain.
Dominance and Proximity

```plaintext
{ SET domoref=##class(%iKnow.Domain).Open(dname)
  GOTO DeleteOldData }
ELSE
{ SET domoref=##class(%iKnow.Domain).%New(dname)
  DO domoref.%Save()
  GOTO SetEnvironment }
DeleteOldData
SET stat=domoref.DropData()
IF stat { GOTO SetEnvironment }
ELSE
{ WRITE "DropData error ",System.Status.DisplayError(stat)
  QUIT }
SetEnvironment
SET domId=domoref.Id
IF ##class(%iKnow.Configuration).Exists("myconfig") {
  SET cfg=##class(%iKnow.Configuration).Open("myconfig")
ELSE
{ SET cfg=##class(%iKnow.Configuration).%New("myconfig",0,System.ListBuild("en"),"",1)
  DO cfg.%Save()
ListerAndLoader
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
DO mylister.Init("myconfig")
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install":dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat = -1 { WRITE "Loader error ",System.Status.DisplayError(stat)
  QUIT }
SourceCountQuery
WRITE ##class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)," total sources",!!
QueryBySource
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId,1,20)
SET j=1,k=1
WHILE $DATA(result(j),srclist) {
  SET src = $LISTGET(srclist)
  WRITE !,"Source id: ",src,!
  DO ##class(%iKnow.Semantics.ProximityAPI).GetClustersBySource(.srcresult,domId,src,1,20)
    WHILE $DATA(srcresult(k)) {
      WRITE $PIECE(item,"",1)," ^ "
      WRITE $PIECE(item,"",2)," ^ "
      WRITE $PIECE(item,"",3),!
      SET k=k+1 }
  SET k=1
  SET j=j+1 }
WRITE !,"Printed ",j-1," sources"
```

Smart Matching: Creating a Dictionary

Smart Matching means combining the results of the iKnow Indexing process with some external knowledge you have in the form of a dictionary, taxonomy, or ontology. What makes iKnow matching “smart” is that those Indexing results help you judge the quality of a match because they identify which words belong together to form concepts and relations. For example, iKnow can identify if a match for your dictionary term "flu" is actually referring to the concept "flu" or the concept "bird flu" in your indexed text source. In the latter case, which is called a partial match, it is clear the match should or could be treated differently than the full match where the dictionary term corresponds exactly to the entity in the indexed text source.

To perform Smart Matching, you must create or acquire a dictionary. If you are creating a dictionary, you must then populate it with the items and terms that you wish to use for matching. Once you have a populated dictionary, you can perform matching operations using the contents of the dictionary.

10.1 Introducing Dictionary Structure and Matching

To populate an iKnow dictionary you first create an item, then associate one or more terms with that item. Commonly a dictionary consists of multiple items, with each item associated with multiple terms. An item is a word or phrase that is a relevant tag for many entities in the source texts. When an entity in the source texts is determined to be a match, it is tagged with the item. For example, the item "ship" is a relevant tag for "ship", "boat", "sail", "oars", and so forth.

To perform this matching, you populate each item in the dictionary with match terms. A term can be single entity (like "motor boat") or a phrase or sentence (like "boats are rowed with oars or paddles"). iKnow indexes each term in the dictionary using the same language model used for the source texts. iKnow then matches each term with the same content unit in the source texts (a Concept term is matched against a Concept in source text; a CRC term is matched against a CRC in source text). If iKnow identifies a match between a term and a unit of source text, iKnow tags the source text passage with the associated dictionary item. This matching frequently is not identical, but requires iKnow to use a scoring algorithm to determine if the term and source text warrant being tagged as a match.

10.1.1 Terminology

A Dictionary is a way to group different terms that have something to do with one another in a logical way. A dictionary could for example be Cities, ICD10 codes or French wines. As a dictionary is the level of aggregation used within the matching APIs, it is specific to the use case to decide what level of real-world grouping should correspond to a dictionary. Taking a higher level (such as "all ICD10 codes") will yield better performance and use lower disk space, but a lower level (such as "a separate one for all ICD10 categories") might offer more granularly grouped results. Each dictionary has a name and a description.
A Dictionary Item is a uniquely identifiable item in your dictionary. Examples of a dictionary item could be cities, the individual codes in ICD10 or individual chateaux. Each dictionary typically has many dictionary items (lots of small dictionaries with few items can decrease performance). A dictionary item has a URI, which should be unique within the domain and can be used as an external identifier, and an optional description. This URI can be used when building rules to interpret matching results later on.

A Dictionary Term is a string that could appear somewhere in a text and represent the Dictionary Item it belongs to. For example, "Antwerp", "Anvers" and "Antwerpen" could be different terms associated with the same dictionary item representing the city of Antwerp. Dictionary terms are the free text strings on which the actual matching is based when doing string-based matching and could be different spellings, translations or synonyms of what your Dictionary Item stands for. These strings are passed through the engine and, when containing more than just a single entity, will automatically be transformed into a more complex structure to be able to match across the boundaries of a single concept (CRC or Path). A dictionary term should also have a language associated with it, if it needs to be processed by the engine.

When processing a new dictionary term by passing it through the iKnow engine, one or more Dictionary Elements are generated to represent the different entities identified within the term. For example, a dictionary term "failure of the liver" would be translated into the three elements "failure", "of" and "liver", with "the" being discarded as non-relevant. These elements are generated and managed automatically and only figure in some types of output, so you shouldn't worry too much about them.

If you want to identify dates, numbers or other formatted pieces of string, you can use Dictionary Formats to specify them, and these can then be included in a Dictionary Term, either representing the complete term, or just a single element within a more complex one. A format is a meaningful pattern of characters, such as a date format. You could associate the formats “nn/nn/nnnn” and “nnnn-nn-nn” with the item named Date. iKnow tags any occurrence of these formats in the source texts with the Date item.

## 10.2 Creating a Dictionary

%iKnow.Matching.DictionaryAPI has a number of methods to create a new dictionary and to assign it items, terms, and formats:

- **CreateDictionary()** is used to create an iKnow dictionary.
  
  The 1st argument specifies the domain Id as an integer. The 2nd argument allows you to specify a meaningful dictionary name. The remaining arguments are optional. The 3rd argument allows you to provide a description of the dictionary, the 4th allows you to specify the language (default is English), and the 5th a custom matching profile. CreateDictionary() returns the dictId, a unique integer. This dictionary ID is used by subsequent smart matching methods. If a dictionary with the specified name already exists, CreateDictionary() returns -1.

- **CreateDictionaryItem()** is used to create an item within a dictionary. You specify the dictId. CreateDictionaryItem() returns the dictItemId, a unique integer.

- **CreateDictionaryTerm()** is used to associate a term with an existing item. You supply the dictItemId. CreateDictionaryTerm() returns the dictTermId, a unique integer.

- **CreateDictionaryItemAndTerm()** is a shortcut that can be used in a specific case. It can be used to create an item and to create a term associated that item when both the term and the item have the same value. For example the item “flu” might have several associated terms (“influenza”, “le grippe”, bird flu”, “H1N1”): you can use CreateDictionaryItemAndTerm() to create the item “flu” and assign it the associated term “flu”. You could, of course, perform the same operation using two method calls: CreateDictionaryItem() and CreateDictionaryTerm().

- **CreateDictionaryTermFormat()** is used to associate a term that consists of a format with an existing item. You supply the dictItemId. CreateDictionaryTermFormat() returns the dictTermId, a unique integer.
10.2.1 Dictionary Creation Example

The following example creates a dictionary named "MovieTerms" and populates it with two items and their associated terms:

```
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
/* ... */
CreateDictionary
SET dictname="MovieTerms"
SET dictdesc="A dictionary of cinema review terms"
SET dictId=##class(%iKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname,dictdesc)
IF dictId=-1 {WRITE "Dictionary ",dictname," already exists",!
GOTO ResetForNextTime }
ELSE {WRITE "created a dictionary ",dictId,!
}
PopulateDictionaryItem1
SET itemId=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryItem(domId,dictId,"comedy",domId_dictId_"comedy")
SET term1Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId,"romantic comedy")
SET term2Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId,"comedy of manners")
SET term3Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId,"bromance")
PopulateDictionaryItem2
SET itemId2=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,dictId,"adventure",domId_dictId_"adventure")
SET i2term1Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId2,"action movie")
SET i2term2Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId2,"western")
SET i2term3Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId(itemId2,"road movie")
DisplayDictionary
SET stat=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryItemsAndTerms(.result,domId,dictId)
SET i=1
WHILE $DATA(result(i)) {
   WRITE $LISTTOSTRING(result(i),",",1),!
   SET i=i+1 }
WRITE "End of items in dictionary ",dictId,!
/* ... */
ResetForNextTime
IF dictId = -1 {
   SET dictId=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname)}
SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
ELSE { WRITE "DropDictionary error ",dictId,!
}
```

10.2.2 Defining a Format Term

The %iKnow.Matching.Formats package provides three simple format classes:

- %iKnow.Matching.Formats.SimpleDateFormat, which matches dates and times in a variety of formats. For a list of the supported date and time formats, refer to the $ZDATETIMEH function in the Caché ObjectScript Reference.
- %iKnow.Matching.Formats.SimplePrefixFormat, which matches any entity that begins with a specified prefix string.
- %iKnow.Matching.Formats.SimpleSuffixFormat, which matches any entity that ends with a specified suffix string.

You can create additional format classes as needed.

The following example uses %iKnow.Matching.Formats.SimpleSuffixFormat. It first defines a dictionary containing one item: speed. The “speed” item contains two terms: “excessive speed” and the suffix format term “mph” (miles per hour). This suffix format will match any entity that ends with the suffix “mph”, for example “65mph”:

```
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
/* ... */
CreateDictionary
SET dictname="Traffic"
SET dictdesc="A dictionary of traffic enforcement terms"
SET dictId=##class(%iKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname,dictdesc)
IF dictId=-1 {WRITE "Dictionary ",dictname," already exists",!
GOTO ResetForNextTime }
ELSE {WRITE "created a dictionary ",dictId,!
}
CreateDictionaryItemAndTerms

SET item1Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryItem(domId,dictId,"speed",domId_dictId_"speed")

SET term1Id=##class(%iKnow.Matching.DictnaryAPI).CreateDictionaryTerm(domId,item1Id,"excessive speed")
SET term2Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTermFormat(domId,item1Id,"%iKnow.Matching.Formats.SimpleSuffixFormat","$LB("mph",0,3)"")
WRITE "dictionary=",dictId,"item=",item1Id,"terms=",term1Id,"term2Id=",term2Id,
/* ... */
ResetForNextTime
IF dictId = -1 {
  SET dictId=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname)}
SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
IF stat {WRITE "deleted dictionary ",dictId," error ",$System.Status.DisplayError(stat) }

10.2.3 Multiple Formats in a Dictionary Term

You can input dictionary formats directly as part of a dictionary term. This allows you to create a dictionary term containing multiple elements, including one or more format elements, as well as string elements.

To use this feature, you specify a "coded" description of the format as part of the string submitted to the CreateDictionaryTerm() method. This coded description has the following format:

@@@User.MyFormatClass@@@param1@@@param2@@@

This description consists of the full class name of the format class (implementing %iKnow.Matching.Formats.Format), a @@@ separator, and a @@@-delimited list of the format parameters to be passed to the format class. The entire description is delimitied with @@ markers at the beginning and end.

If the format class takes no parameters, or the defaults are to be used, specify the format class name delimited by @@ markers.

When including this format in a dictionary term string, you must make sure that iKnow will recognize it as a single entity. For examples, the term "was born in @@User.MyYearFormat@@" is interpreted as a single entity, but the term "was born in the year @@User.MyYearFormat@@" is not.

If iKnow cannot find the specified format class, the @@ usage is considered intentional and the whole entity is treated as a simple string element.

Using this syntax makes it easier to load dictionaries from files or tables without requiring separate steps or actions for the formats.

10.3 Listing and Copying Dictionaries

The %iKnow.Matching.DictionaryAPI class has a number of methods to count or list existing dictionaries and their items and terms.

The %iKnow.Util.CopyUtils class has a number of methods to copy a dictionary or all dictionaries from one domain to another.

10.3.1 Listing Existing Dictionaries

The following example lists all of the dictionaries in the domain. For the purpose of demonstration, this example first creates two empty dictionaries, one in English (the default language) and one in French:

SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
SET dictname1="Diseases",dictname2="Maladies"
SET dictdesc1="English disease terms",dictdesc2="French disease terms"
CreateFirstDictionary
SET dictId1=##class(%iKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname1,dictdesc1)
IF dictId1 = -1 {
    SET dictId1=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname1)
    SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
        QUIT }
    GOTO CreateFirstDictionary }
ELSE {WRITE "created a dictionary ",dictId1,!}
CreateSecondDictionary
SET dictId2=##class(%iKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname2,dictdesc2,"fr")
IF dictId2 = -1 {
    SET dictId=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname2)
    SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
        QUIT }
    GOTO CreateSecondDictionary }
ELSE {WRITE "created a dictionary ",dictId2,!}
GetDictionaries
SET stat=##class(%iKnow.Matching.DictionaryAPI).GetDictionaries(.dicts,domId)
WRITE "get dictionaries status is:",$System.Status.DisplayError(stat),!!
SET k=1
WHILE $DATA(dicts(k)) {
    WRITE $LISTTOSTRING(dicts(k)),!
    SET k=k+1 }
WRITE "End of list of dictionaries"

GetDictionaries() lists the Id, name, description, and language for each dictionary.

10.3.2 Copying Dictionaries

You can copy dictionaries from one domain to another within the current namespace.

• You can use the CopyDictionaries() method of the %iKnow.Utils.CopyUtils class to copy all defined dictionaries in a domain to another domain. By default this method also copies matching profiles from one domain to another.

• You can use the CopyDictionary() method of the %iKnow.Utils.CopyUtils class to copy a single defined dictionary in a domain to another domain. By default this method also maps matching profile Ids from one domain to another.

• You can use the CopyDomain() method of the %iKnow.Utils.CopyUtils class to copy dictionaries as part of a domain copy operation.

10.4 Extending Dictionary Constructs

Though iKnow only describes simple dictionaries in the Matching API, this does not restrict you from using more advanced tools like ontologies, taxonomies or other more hierarchical constructs. The goal of the Matching API is to provide the hooks for just the matching, rather than yet another generic structure that tries to cover every construct. Therefore, you should just flatten the structure of the ontology or taxonomy you have. By appropriately choosing your dictionary item URIs, you’ll be able to reconstruct or interpret the matching results within the context of your ontology or taxonomy.

In the Matching API, the formatting bits are pluggable in the sense that you can provide your own implementation of a class that does for example regular expression matching by implementing the %iKnow.Matching.Formats.Format interface.
11

Smart Matching: Using a Dictionary

Smart Matching means combining the results of the iKnow Indexing process with some external knowledge you have in the form of a dictionary, taxonomy, or ontology. What makes iKnow matching “smart” is that those Indexing results help you judge the quality of a match because they identify which words belong together to form concepts and relations. For example, iKnow can identify if a match for your dictionary term “flu” is actually referring to the concept “flu” or the concept “bird flu” in your indexed text source. In the latter case, which is called a partial match, it is clear the match should or could be treated differently than a full (exact) match where the dictionary term corresponds exactly to the entity in the indexed text source.

To perform Smart Matching, you must create or acquire a dictionary. Creating and populating a dictionary is described in the previous chapter. Once you have a populated dictionary, you can perform matching operations using the contents of the dictionary.

11.1 How Dictionary Matching Works

iKnow matches each term in the dictionary with the same level construct in the source texts (a Concept term is matched against a Concept in source text; a CRC term is matched against a CRC in source text). These matches can be exact, or can be partial matches. If the match between term and source text is exact, iKnow tags the source text passage with the dictionary item associated with that term. If the match between term and source text is not exact, iKnow scores the degree of match between them. This match scoring involves calculating a match score for each component entity (concept or relation), then, if required, using these entity match scores to calculate the match score for a CRC, path, or sentence. If the scoring of a partial match achieve a configured minimal match score, iKnow tags the source text passage with the dictionary item associated with the term (see the MinimalMatchScore property in %iKnow.Matching.MatchingProfile).

11.1.1 Match Scoring

iKnow generates a match score, a floating point number, which is calculated from all of the entity matches detected between the dictionary term and the unit of source text. For an entity match, this match score can range between 0 (no match) and 1 (an exact match). For CRC or path matches, this match includes this range, but can be greater than 1. The algorithm used to calculate this score is complex, but includes the following considerations:

- A full match of an entity can be exact (same words in same order) or scattered (same words in different order). The match score for a scattered match is determined by multiplying 1 (exact match) by the value of the ScatteredMatchMultiplier property of the matching profile.
- A partial match of an entity (concept or relation) is assigned a percentage based on the percentage of the source text string that matches the dictionary term.
• In a partial match of an entity, a matching relation is assigned only half as much value as a matching concept. You can change this ratio (for example, to an equal evaluation of relation and concept matches) by setting the `RelationshipScoreMultiplier` property of the matching profile.

• When matching a CRC, path, or sentence, the match scores from the entity matches are added, then divided by the length of the dictionary term, multiplied by the number of matching entities, then multiplied by the `DisorderMultiplier` property, a value representing the degree of disorder (difference in the sequence of entities) between the unit of source text and the dictionary term.

The above is not an exact formula for obtaining a match score. It is provided to show the principal considerations used when iKnow calculates a match score.

iKnow provides a matching profile `%IKnow.Matching.MatchingProfile`, which consists of the numeric properties mentioned above, and others. iKnow uses this matching profile when calculating a match score. iKnow provides default property values for the matching profile. Unless otherwise specified, the default matching profile is assigned to each dictionary. This default matching profile provides accurate matching for most applications. Creating and assigning a custom matching profile is described later in this chapter.

## 11.2 Matching Against a Dictionary

You can use the `%IKnow.Matching.MatchingAPI` class to perform matches between text source entities and a populated dictionary (or multiple dictionaries).

### 11.2.1 Matching a String

The `GetDictionaryMatches()` method matches a string against a dictionary and returns the match items. The following example matches a string against the MovieTerms dictionary:

```plaintext
SET domId=##class(%IKnow.Domain).GetOrCreateId("mydomain")
/* ... */
CreateDictionary
SET dictname="MovieTerms"
SET dictId=##class(%IKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname)
IF dictId=-1 {WRITE "Dictionary ",dictname," already exists",!
  GOTO ResetForNextTime }
ELSE {WRITE "created a dictionary ",dictId,!! }
PopulateDictionaryItem
SET itemId=##class(%IKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,itemId,
"adventure",domId_dictId_"adventure")
SET item2Id=##class(%IKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,itemId,
"action movie")
SET item3Id=##class(%IKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,itemId,
"western")
SET item4Id=##class(%IKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,itemId,
"road movie")
DisplayDictionary
SET stat=##class(%IKnow.Matching.DictionaryAPI).GetDictionaryItemsAndTerms(.result,domId,dictId)
WRITE ",items and terms in dictionary ",dictname,!
SET i=1
WHILE $DATA(result(i)) {
  WRITE ",LISTTOSTRING(result(i),",",1),!
  SET i=i+1 }
WRITE ",End of items in dictionary ",dictId,!!
DoMatching
SET mystring="A western action adventure movie script"
SET stat=##class(%IKnow.Matching.MatchingAPI).GetDictionaryMatches(.num,domId,mystring,SLB(dictId))
IF stat'=1 {WRITE ",get matches status is:",$System.Status.DisplayError(stat),!
  QUIT }
WRITE ",The string is: ",mystring,!
WRITE ",The matches are:!
SET j=1
WHILE $DATA(num(j)) {
  WRITE ",match number ",j," is ",$LISTTOSTRING(num(j)),!
  SET j=j+1 }
WRITE ",End of match items for dictionary ",dictId,!!
ResetForNextTime
```

Using iKnow
IF dictId = -1 {
  SET dictId=#class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname)}
SET stat=#class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
IF stat {WRITE "deleted dictionary ",dictId,!
}

The following is a match result (returned in %List format, here shown as a comma-separated string):
5,5,15adventure,18,action movie,18,term,.333333,00101,0,. These element values are explained below:

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>15adventure</th>
<th>18</th>
<th>action movie</th>
<th>18</th>
<th>term</th>
<th>.333333</th>
<th>00101</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary Id</td>
<td>item Id</td>
<td>item URI (domain Id + dictionary Id + item)</td>
<td>term Id</td>
<td>term</td>
<td>element Id</td>
<td>type of match (term, format, or unknown)</td>
<td>match score</td>
<td>matched word bit map</td>
<td>is scattered boolean</td>
</tr>
</tbody>
</table>

The matched word bit map 00101 shows the match of the dictionary term “action movie” to the string “A western action adventure movie / script”. The bit map stops when the match completes, so there is no bit for the word “script”. The isScattered boolean is 0 because the words “action” and “movie” are in the same order in the dictionary term and in the string.

11.2.2 Matching a Source

The GetTotalItemScoresBySource() method matches a source against a dictionary and returns the match scores for each dictionary item.

The following example matches all of the sources in the domain against the MovieTerms dictionary and returns each source’s match scores for each dictionary item:

```plaintext
DomainGetOrCreate
SET dname="mydomain"
SET domId=#class(%iKnow.Domain).GetOrCreateId(dname)
SET domoref=#class(%iKnow.Domain).%OpenId(domId)
IF domoref.IsEmpty() {GOTO ListerAndLoader }
ELSE {IF domoref.DropData() {GOTO ListerAndLoader }
ELSE {WRITE "DropData error ",$System.Status.DisplayError(stat) QUIT} }

ListerAndLoader
SET domId=domoref.Id
SET mylister=#class(%iKnow.Source.File.Lister).%New(domId)
SET myloader=#class(%iKnow.Source.Loader).%New(domId)
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_"dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,""
IF stat = -1 {WRITE "Loader error ",$System.Status.DisplayError(stat) QUIT}
SourceCountQuery
SET numSrcD=#class(%iKnow.Queries.SourceAPI).GetCountByDomain(domId)
WRITE "The ",dname," domain contains ",numSrcD," sources",!
CreateDictionary
SET dictname="MovieTerms"
SET dictId=#class(%iKnow.Matching.DictionaryAPI).CreateDictionary(domId,dictname)
IF dictId=-1 {WRITE "Dictionary ",dictname," already exists",!
GOTO DisplayDictionary } ELSE {WRITE "created a dictionary ",dictId,!
}

PopulateDictionaryItem1
SET itemId=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryItem(domId,dictId, "filmwords",domId_dictId_"filmwords")
SET termId1=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId, "movie")
SET termId2=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId, "movies")
SET termId3=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId, "film")

PopulateDictionaryItem2
SET itemId2=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryItemAndTerm(domId,dictId, "filmtypes",domId_dictId_"filmtypes")
SET i2termId1=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId2, "road movies")
SET i2termId2=#class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId2, "road movies")
```

Using iKnow
"box office successes")
SET i2term3Id=##class(%iKnow.Matching.DictionaryAPI).CreateDictionaryTerm(domId,itemId2,
"romantic thrillers")
DisplayDictionary
SET stat=##class(%iKnow.Matching.DictionaryAPI).GetDictionaryItemsAndTerms(.result,domId,dictId)
WRITE "Items and terms in dictionary ",dictname,!
SET i=1
WHILE $DATA(result(i)) {
WRITE $LISTTOSTRING(result(i),",",1),!  SET i=i+1 }
WRITE "End of items in dictionary ",dictId,!!
DoMatching
SET num=0
FOR j=1:1:10 {SET extId=##class(%iKnow.Queries.SourceAPI).GetExternalId(domId,j)
SET mstat=##class(%iKnow.Matching.MatchingAPI).GetTotalItemScoresBySource(.mresult,domId,extId,$LB(dictId))
IF mstat '=1 {WRITE "End of sources",!
QUIT }
SET k=1
WHILE $DATA(mresult(k)) {
WRITE $PIECE($LISTTOSTRING(mresult(k)),",",2),""
WRITE $PIECE($LISTTOSTRING(mresult(k)),",",4)
WRITE " matches: ",$PIECE($LISTTOSTRING(mresult(k)),",",6)
WRITE " score: ",$PIECE($LISTTOSTRING(mresult(k)),",",7),!
SET k=k+1 }
WRITE "End of ",srcname," match items for dictionary ",dictId,!!
}
ResetForNextTime
IF dictId = -1 {##class(%iKnow.Matching.DictionaryAPI).GetDictionaryId(domId,dictname)}
SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
IF stat (WRITE "deleted dictionary ",dictId,!! }
The following is a match result (returned in %List format, here shown as a comma-separated string):
76,MovieTerms,95,comedy,176comedy,6,5.72. These element values are explained below:

<table>
<thead>
<tr>
<th>76</th>
<th>MovieTerms</th>
<th>95</th>
<th>comedy</th>
<th>176comedy</th>
<th>6</th>
<th>5.72</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary Id</td>
<td>dictionary name</td>
<td>item Id</td>
<td>item</td>
<td>item URI (domain Id + dictionary Id + item)</td>
<td>score</td>
<td></td>
</tr>
</tbody>
</table>

11.3 Defining a Matching Profile

Whether a dictionary term and a unit of source text constitute a match is determined by the matching profile properties found in %iKnow.Matching.MatchingProfile. There are eight properties that together determine whether something is a match or not. All of these properties take an appropriate default value. By default, every dictionary is assigned the default matching profile.

You can create a custom matching profile with one or more properties whose value differs from the default. You can then assign this custom matching profile to a dictionary. Any properties not specified in the custom matching profile take default values. You can create any number of custom matching profiles. The same matching profile can be applied to multiple dictionaries. You can define the matching profile to be specific to a domain, or to be available to all domains in the namespace.

The following example creates three custom matching profiles. The first is specific to a domain and specifies the optional matching profile name; iKnow assigns it a positive integer Id. The second and third are available to all domains in the namespace; iKnow assigns each of them a negative integer Id. Because the third specifies the optional matching profile name, it must specify 0 as a placeholder for the domain Id parameter:
The following example shows how to define a custom matching profile and assign it to a dictionary. It specifies the custom matching profile instance oref (in this case, customprofile) to the CreateDictionary() method to override the default.

```
#Include %IKPublic
SET domId=##class(%iKnow.Domain).GetOrCreateId("mydomain")
CustomizeMatchingProfile
SET customprofile=##class(%iKnow.Matching.MatchingProfile).%New()
WRITE "MinimalMatchScore initial value=",customprofile.MinimalMatchScore,!
SET customprofile.MinimalMatchScore=".4"
WRITE "MinimalMatchScore custom value=",customprofile.MinimalMatchScore,!
CreateDictionary
/* . . . */
CleanUpForNextTime
SET stat=##class(%iKnow.Matching.DictionaryAPI).DropDictionary(domId,dictId)
IF stat {WRITE "Dropped the dictionary"
```

11.3.1 Matching Profile Properties

Whether the terms in a dictionary and a unit of source text constitute a match is determined by the matching profile property values you define in your custom matching profile. These properties determine the match score for each unit of text and the threshold which specifies the minimum match score to report as a match. Because the contents of a dictionary tends to vary depending on the use case, you may wish to customize one or more matching profile properties for your dictionary. Changing any of these properties may dramatically change the number of matches reported. Therefore, you might want to experiment with changes to property values while testing on a small subset of your data before registering the matching profile to match the dictionary to the whole dataset.

The meaning of the properties in %iKnow.Matching.MatchingProfile are explained in the class documentation.

11.3.1.1 MinimalMatchScore

The most common matching profile property to tune is the MinimalMatchScore. This property value is the lower threshold for matches to be saved. You can set it to a fractional value between 1 (only perfect matches saved) and 0 (all potential matches saved); the default is 0.33.

- By increasing this property value you can filter out low-quality matches if you have an excessive number of match candidates. This would be appropriate if your dictionary is fairly generic and contains many common terms. You may only wish to report CRCs and paths that are a very close match to several of these common terms.
- By decreasing this property value you can increase the number of match candidates. This would be appropriate if your dictionary is highly specific, consisting only of critical terms that should be flagged at all times. You may wish to report CRCs and paths that loosely match with a dictionary of technical terms, so as to avoid missing a loose, but significant, match.

Setting MinimalMatchScore to 0 returns all possible match results. If you're starting to work on matching a new dictionary to a small subset of your data, you can start by setting MinimalMatchScore to 0, then gradually increase it, filtering out low-quality matches, until you get a reasonable number of results.
11.3.2 Domain Parameters and Matching

You can specify a different domain-wide default matching profile by setting the MAT:DefaultProfile (IKPMATDEFAULTPROFILE) domain parameter. (The “MAT:” prefix indicates a domain parameter specific to matching operations.) You can also set individual domain parameters that influence matching operations.

11.3.2.1 Matching Single Relations

By default, the iKnow matching algorithm skips dictionary terms that match a single relation entity. For example, if you have a dictionary term “to”, iKnow does not attempt to match the entity “goes to” in the sentence “Pete goes to the cinema”. This default optimizes matching performance when your dictionary primarily contains and targets concepts.

However, if your dictionary deliberately targets single relation elements, you can change this domain parameter default by setting the MAT:SkipRelations (IKPMATSKIPRELS) domain parameter to 0. This causes all entities to be matched against all dictionary terms, regardless of the type of entity.

This option is set as a domain parameter — not a matching profile property — because this skipping step occurs at a time when it is not yet known what the matched terms might be, and hence could not apply any dictionary-specific profiles. Therefore, it is implemented as a domain parameter, ensuring it applies to all dictionaries within the domain.

11.3.2.2 Other Matching Operation Domain Parameters

Several of the settable domain parameters can influence matching operations: MAT:SkipRelations, MatchScoreMargin, FullMatchOnly, and EntityLevelMatchOnly. If you change these domain parameters, any sources matched using a previous domain parameter setting will have to be explicitly re-matched to reflect the new domain parameter value.

For further information on these domain parameters, refer to the Domain Parameters appendix to this manual.
12
User Interfaces

The iKnow technology does not have a default user interface. This chapter describes a few sample user interfaces provided with iKnow. These can provide a convenient starting point when developing a query interface specific to your use of iKnow.

These sample user interfaces are found in the %iKnow.UI package, and are implemented as Zen pages, as described in the InterSystems Class Reference.

12.1 How to Display iKnow User Interfaces

1. from the Caché Cube, access Studio.
2. from the File drop-down menu, select Change Namespace. Select the %SYS namespace. Click OK.
3. from the File drop-down menu, select Open.
4. from the Open window, make sure that Include System Items is checked. Then select %iKnow->UI->and the desired user interface. Click Open. The source code for the user interface appears in the Class Editor (the main Studio window).
5. click the View Web Page icon (the planet Earth icon). The user interface displays in your default browser. (Note that abstract classes cannot be displayed.)

For further details, refer to the Using Caché Studio manual.

12.2 Abstract Portal

This is the superclass for all other portals and pages in the %iKnow.UI package. It groups a lot of reusable materials such as handling of domain ID, the "selected" source ID, metadata filters and paging for iKnow query-driven tables and groups. It is abstract, and cannot be run by itself, but should not impose restrictions on subclasses or component names, as long as the corresponding panes (optDomainPane, txtTermPane, optSourcePane and filterPane) are used.

The %iKnow.UI.AbstractPortal class has five subclasses that fall into two types:

• The Loading Wizard, which provides a source file management interface.
• The query portal interfaces, which provide different displays of source data, showing iKnow indexing and dictionary matching.
12.3 Abstract Source Viewer

This class extends the AbstractPortal superclass. It contains the `ProcessInput()` and `DeleteCurrentVirtualSource()` methods. It is abstract, and cannot be run by itself.

12.4 Loading Wizard

A management and maintenance interface that allows a user to easily choose (or manage) Domain and Configuration objects and then load files from a filesystem directly into a domain using `%iKnow.Source.File.Lister`. It also provides functionality to load metadata values from a CSV (comma-separated values) file with rows corresponding to files previously loaded through the File Lister. This operation automatically creates previously nonexistent metadata fields on the fly.

For further details on Domain and Configuration objects, refer to the “Setting Up an iKnow Environment” chapter. For further details on listing and loading, refer to the “Loading Text Data into iKnow” chapter.

12.5 Knowledge Portal

This is a sample Zen page query display interface with broad application. It shows a wealth of information about the various language elements identified by iKnow, including entities, CRCs, CCs and paths, providing a contextual at-a-glance view of what’s in your data. The generic filters option allows for easily selecting subsets of a domain based on metadata criteria and the summary option provides quick access to the contents of the sources themselves. This interface provides a sample of how iKnow Smart Indexing can be used to quickly overview and navigate a large set of documents.

The Knowledge Portal offers straightforward examples of calling the different iKnow Query APIs from Zen, including top entities, similar entities, and related entities, with the frequency and spread for each.

The Knowledge Portal supports the use of black lists.

12.6 Basic Portal

This sample Zen page query display interface is a simplified version of the Knowledge Portal. It displays entities and sources only. It does not display CRCs, CCs and paths. It provides filtering and summary capabilities, but by default it shows the full text of the source.

The Basic Portal supports the use of black lists.

12.7 Indexing Results

This sample Zen page query display interface looks at the Smart Indexing results for a single document to verify the correctness of the analysis done by the iKnow analysis engine. It shows how iKnow cuts up each sentence into a sequence of concepts (bold and highlighted), relations (underlined), and non-relevants (italic). The page also shows a frequency-sorted list of detected Concepts and CRCs. This page provides an option for loading input manually. This page offers an example of how `%iKnow.Queries.SentenceAPI.GetParts()` can be used for custom highlighting.
For definitions of concepts, relations, CRCs, and non-relevants, refer to the “Conceptual Overview” chapter.

12.8 Matching Results

This sample Zen page query display interface gives for each document an overview of the different matching results in the document. It allows for easy browsing through match results against a dictionary and allows you to display the details of each individual match. It uses %iKnow.Matching.MatchingAPI.GetHighlightedSentences() to display the text with dictionary matches highlighted in color, and %iKnow.Matching.MatchingAPI.GetMatchElements() to display the details of a specific match including the dictionary name, its item and term, the match score, type of match and the entity matched. This interface provides a sample of how iKnow Smart Matching can be used to combine predefined information with Smart Indexing results.

For further details on matching, refer to the “Smart Matching: Using a Dictionary” chapter.
13 iKnow Web Services

The %iKnow package provides web service classes that execute iKnow queries. This chapter provides an overview of these classes and describes how to use them. It discusses the following topics:

• Available iKnow web services
• How to use an iKnow web service
• Example
• Comparison of iKnow web services to the primary APIs
• Additional sources of information

13.1 Available Web Services

iKnow provides the following web service classes:

• %iKnow.Queries.CcWSAPI — provides web methods that you can use to get information about concept-concept (CC) pairs. These web methods are equivalent to the methods in the %iKnow.Queries.CcAPI class.

• %iKnow.Queries.CrcWSAPI — provides web methods that you can use to get information about Concept-Relation-Concept (CRC) triples. These web methods are equivalent to the methods in the %iKnow.Queries.CrcAPI class.

• %iKnow.Queries.EntityWSAPI — provides web methods that you can use to get information about entities. These web methods are equivalent to the methods in the %iKnow.Queries.EntityAPI class.

• %iKnow.Queries.EquivWSAPI — provides web methods that you can use to manage equivalences. These web methods are equivalent to the methods in the %iKnow.Queries.EquivAPI class.

• %iKnow.Queries.MetadataWSAPI — provides web methods that you can use to manage and get information about source metadata. These web methods are equivalent to the methods in the %iKnow.Queries.MetadataAPI class.

• %iKnow.Queries.PathWSAPI — provides web methods that you can use to get information about paths. These web methods are equivalent to the methods in the %iKnow.Queries.PathAPI class.

• %iKnow.Queries.SentenceWSAPI — provides web methods that you can use to get information about sentences. These web methods are equivalent to the methods in the %iKnow.Queries.SentenceAPI class.

• %iKnow.Queries.SourceWSAPI — provides web methods that you can use to get information about sources. These web methods are equivalent to the methods in the %iKnow.Queries.SourceAPI class.
The %iKnow.Queries package also contains classes that the compiler generated when these classes were compiled. For example, when the compiler compiled the class %iKnow.Queries.CcWSAPI, it generated the classes in the %iKnow.Queries.CcWSAPI package. These generated classes are not intended for direct use.

13.2 Using an iKnow Web Service

To use a web service, you create and use a web client that communicates with it. To do so for an iKnow web service, you use the same procedure as with any other web service:

1. Create a web client that can communicate with the web service. Typically, to do so, you generate the web client using a tool provided by the client technology and you provide the WSDL of the web service as input. This process generates a set of client classes.

   For example, in Caché, use the Studio SOAP wizard. For a Caché web service, Caché conveniently publishes the WSDL at the following URL:

   ```
   http://hostname:port/csp/namespace/web_service_class.cls&WSDL
   ```

   Where:

   - `hostname` is the server on which Caché is running.
   - `port` is the port on which the web server is running.
   - `namespace` is the namespace name.
   - `web_service_class` is the full package and class name of the web service with .cls at the end.

   For example, for the class %iKnow.Queries.EntityWSAPI, use %25iKnow.Queries.SourceWSAPI.cls

   **Important:** Be sure to use to replace the leading percent sign of the package with the URL escape sequence %25

   For example:

   ```
   http://localhost:57772/csp/samples/%25iKnow.Queries.SourceWSAPI.cls&WSDL
   ```

2. Rather than editing the generated client classes, create an additional class or routine that uses them. The details depend on the technology.

   In Caché, to use a web client, you create an instance of the web client class and then invoke its instance methods. See the following example.

13.3 Example

For demonstration purposes, let us generate and use a Caché web client to work with an iKnow web service on the same machine. To do this in the SAMPLES namespace:

1. In Studio, click **Tools > Add-ins > SOAP Wizard.**
2. On the first screen, click **FILE.**
3. Type the following URL:

   ```
   ```
If needed, replace 57772 with the port that your web server uses for this installation of Caché.

4. Click **Next**.

5. For **Proxy Class Package**, type a package name such as **MyClient**

6. Click **Next** twice.

   The wizard generates and compiles the classes and displays a list of these classes.

   This process generates the class **MyClient.iKnow.Queries.EntityWSAPISoap**. This class defines a set of methods, one for each web method defined in the web service on which this client is based. Each of these methods looks like this (with illegal line breaks added here):

   ```
   Method GetByFilter(domainid As %Integer, filter As %String, filtermode As %Integer, enttype As %Integer,
   blacklistIds As %ListOfDataTypes(ELEMENTTYPE="%String",XMLITEMNAME="blackListIdsItem",XMLNAME="blackListIds"))
   As %XML.DataSet [ Final, ProcedureBlock = 1, SoapBindingStyle = document, SoapBodyUse = literal, WebMethod ]
   {
   Quit ..WebMethod("GetByFilter").
   .domainid,.filter,.filtermode,.enttype,.blackListIds)
   }
   ```

   The signature of this method is the same as for the corresponding method in the web service.

   This process also generates a set of classes in the **MyClient.iKnow.Queries.EntityWSAPISoap**, which are not for direct use.

7. Click **Finish**.

8. To use the generated client, enter the following in the Terminal in the **SAMPLES** namespace:

   ```
   set client=##class(MyClient.iKnow.Queries.EntityWSAPISoap).%New()
   ```

   This creates an instance of the client class, which can then communicate with the web service.

9. Execute methods of this class. For example:

   ```
   write client.GetCountByDomain(2)
   ```

---

### 13.4 Comparison of iKnow Web Services with Primary iKnow APIs

The primary iKnow APIs use arguments that cannot be easily represented in SOAP messages, so the methods in iKnow web services have different signatures than do the methods in the primary iKnow APIs. In particular, note the following differences:

- Instead of `%Library.List`, the web services (and their clients) use `%Library.ListOfDataTypes`. This means that to build a list for a web service, you create an instance of `%Library.ListOfDataTypes` and then use its **SetAt()** method to add items. You cannot use list functions such as $LISTBUILD with this instance.

- Instead of an instance of `%IKnow.Filters.Filter`, the web services (and their clients) use a string of the form returned by the **ToString()** method of that class.

- For APIs that return complex results, rather than returning results by reference as a multidimensional array, the web services (and their clients) return an instance of `%XML.DataSet`. 

Comparison of iKnow Web Services with Primary iKnow APIs
Important: This structure is supported only in .NET and in Caché. Other web technologies do not recognize this format.

13.5 See Also

For information on web services and clients in Caché, see Creating Web Services and Web Clients in Caché.
DeepSee dashboard technology enables you to create web-based dashboards for your end users. Despite the specific name, these dashboards are not reserved solely for DeepSee users and can display data other than DeepSee data items. Among other items, dashboards can display iKnow KPIs (key performance indicators).

In general, a KPI is a query that can be executed and displayed on dashboards; when the dashboard is displayed, the query is executed. An iKnow KPI uses an iKnow ObjectScript query.

This section describes how to create iKnow KPIs and display them on dashboards. It discusses the following topics:

- KPI terminology
- Basic steps to define a KPI
- Available KPI filters
- How to override KPI properties
- Example KPI
- How to create a dashboard to display the KPI
- How to provide access to dashboards
- Additional sources of information

For information on using iKnow data within DeepSee cubes, see “Using Unstructured Data in Cubes” in *Defining DeepSee Models*. (Or, for information on the older form of cube integration, see the appendix “DeepSee Cube Integration (Deprecated Form)".)

### 14.1 KPI Terminology

In a KPI, each row returned by the query is a separate *series* of the KPI. The following shows some of the series of a KPI (as seen on the KPI test page, discussed later in this chapter). The series names are shown in the first column of the **KPI Values** table.
For iKnow KPIs, the name of a series (by default), is the value in the first column returned by the query.

A KPI also contains properties, each of which corresponds to a column in the returned data. In the previous example, the KPI has five properties. When you define an iKnow KPI, you can override these property names and their order.

The KPI query can include parameters. These are called KPI filters, because they usually (but not always) filter the values returned by the KPI. The KPI test page shows all the available KPI filters. The iKnow KPI mechanism automatically provides a set of KPI filters for any iKnow KPI.

A KPI can also define actions, which execute custom code. When you add a KPI to a dashboard, you can add controls to the dashboard to execute these actions. For any iKnow KPI based on a query that uses paging, that KPI defines the Previous page and Next page actions.

### 14.2 Defining a KPI That Uses an iKnow Query

To define a KPI that uses an iKnow query, do the following:

1. Create a class that extends one of the following classes:
   - `%iKnow.DeepSee.GenericKPI` — Use this for most queries. You specify the query in the next step.
   - `%iKnow.DeepSee.SourceListFilterKPI` — Use this to display information about the sources. In this case, the system uses either the `GetByEntities()` or the `GetByDomain()` method of `%iKnow.Queries.SourceAPI`.

2. In this class, specify the iKnow domain to use. To do so, do either of the following:
   - Specify the integer ID of the domain. To do so, override the `IKDOMAINID` class parameter and set it equal to the integer ID of the iKnow domain.
   - Specify the DeepSee cube and iKnow measure that define the domain. To do so, override the following class parameters:
     - `IKCUBENAME` — Should equal the logical name of a DeepSee cube.
- **IKMEASURENAME** — Should equal the local name of an iKnow measure in the given cube.

This technique is possible only if you are using the DeepSee cube integration; see “Using Unstructured Data in Cubes” in *Defining DeepSee Models*.

3. Also override the following class parameters, depending on the superclass you used:
   - **IKPAGESIZE** — Should equal the number of rows to display on any page. The default is 10.
     Within DeepSee dashboards, this affects the number of rows shown per page in a dashboard widget.
   - **IKQUERYCLASS** — (Only if you subclass %iKnow.DeepSee.GenericKPI) Should equal the complete package and class name of the iKnow API. Use one of the API classes that belong to the ObjectScript API. For example: %iKnow.Queries.EntityAPI
   - **IKQUERYNAME** — (Only if you subclass %iKnow.DeepSee.GenericKPI) Should equal the name of a method in that class.

4. Add an XData block like the following to the class; this specifies the logical name and the display name for the KPI:

   ```
   /// This XData definition defines the KPI.
   XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
   {<kpi name="MyKPI" displayName="My KPI">
   </kpi>
   }
   ```

   Specify the `name` and `displayName` values as needed. Note that `name` should not include any spaces or punctuation. The value of `displayName` is localizable. You can omit `displayName`; by default, `name` is used.

   Also see the later subsection “Overriding the KPI Properties.”

5. Compile the class.

   Optionally use the KPI test page. If you are currently viewing the class in Studio, click View > Web Page.

   To test the KPI, use the drop-down lists in the Filters section. Then click Submit Query.

   These drop-down lists include all the available KPI filters; see the next subsection.

### 14.3 Available KPI Filters

The query of the KPI can include parameters; these are called *KPI filters*, because they usually (but not always) filter the values returned by the KPI. For an iKnow KPI, the following KPI filters are automatically available:

- A meaningful subset of iKnow query parameters for the query that you are using (some query parameters, such as the iKnow domain ID, are handled automatically and are not exposed)
- All public iKnow source metadata fields for this domain
- The NAMECOLUMN parameter, which lets you specify the column to use as the series name
Important: Many of the iKnow queries include a `filtermode` argument that you use to specify the statistical reprocessing to perform after applying a filter; see “Filter Modes,” earlier in this book. When you use such queries directly, the default `filtermode` is $$$FILTERONLY, which performs no reprocessing.

When you expose such queries as an iKnow KPI, the `filtermode` argument is always specified as $$$FILTERALLANDSORT, so that the frequency and spread statistics are recomputed and the results are re-sorted. Therefore you might see different results when using these queries as iKnow KPIs than when using them directly, depending on how you specify `filtermode`.

### 14.4 Overriding the KPI Properties

By default, any KPI based on `%iKnow.DeepSee.GenericKPI` exposes all result columns in the same order as the query result, with the same names as in the query result. You can modify the order, change names, and hide columns.

Each result column is a KPI property. Users see the following when they create DeepSee dashboards:

- When a user adds a pivot table widget based on a KPI, the properties are shown in the same order and with the same names as defined in the KPI, by default.
- When a user adds a scorecard widget based on a KPI, the user chooses the properties to display. The drop-down list of choices contains the properties in the same order and with the same names as defined in the KPI.
- In both cases, users can modify the order of the KPI properties and change the titles displayed for them.

To modify the order of the KPI properties, change their names, and hide properties, add a set of `<property>` elements within the XData block, as follows:

```xml
XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
{
  <kpi name="MyKPI" displayName="My KPI">
    <property name="entity" displayName="Entity" />
    <property name="frequency" displayName="Frequency" />
    <property name="spread" displayName="Spread" />
  </kpi>
}
```

For `<property>`, the value for `name` must exactly match the name of the field returned by the query. For `displayName`, use the name that should be visible to users. List the `<property>` elements in the desired order, and list all the properties that should be visible.

### 14.5 Example

The following shows an example KPI:

```java
Class GIKNOW.TopEntitiesKPI Extends %iKnow.DeepSee.GenericKPI 
{
  Parameter IKDOMAINID = 1;
  Parameter IKPAGESIZE As %Integer = 10;
  Parameter IKQUERYCLASS = "%iKnow.Queries.EntityAPI";
  Parameter IKQUERYNAME = "GetTop";

  /// This XData definition defines the KPI.
  XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
  {
    <kpi name="TopEntities" displayName="Top Entities in iKnow domain 1" >
```
For another example, see “Example Dashboard with iKnow KPI,” in the next section.

### 14.6 Creating a Dashboard to Display the KPI

To make a KPI available to users, you add it to a dashboard. Users can then access the dashboard in various ways; see “Providing Access to Dashboards,” later in this chapter.

- How to create a dashboard
- Changing the series name
- Configuring the properties
- Adding previous page and next page buttons
- Example dashboard

#### 14.6.1 Creating Dashboards: Basics

The following instructions briefly explain how to create a simple dashboard that displays a KPI:

1. Click Home, DeepSee, User Portal and then click View.
   The system then displays the User Portal, which lists any existing public dashboards and pivot tables in this namespace.

2. Click Menu > New Dashboard.
   The system displays a dialog box that prompts you for basic information about the new dashboard.

3. Type a value for Dashboard Name.

4. Optionally specify a value for Folder.

5. For Page Layout, click the third option (no workboxes).

6. Click OK.
   The system creates, saves, and displays the dashboard, which is initially empty.
7. Add a widget to display a KPI. To do so:
   a. Click Menu > Add New Widget....
   b. In the left area, click an item to show a list of choices.
      Select either a pivot table widget or a scorecard.
   c. Click the type of the widget to use.
   d. On the Data Source tab, click the Search button next to Data source.
   e. Click the name of the KPI.
   f. Click OK.
   g. If you chose a scorecard, see “Configuring the Properties,” later in this section.
   h. Click OK to add this widget.

8. Resize the widget and click Menu > Save again.

9. For configuration options, see the next subsections.

The result might be as follows:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Frequency</th>
<th>Spread</th>
</tr>
</thead>
<tbody>
<tr>
<td>researchers</td>
<td>31</td>
<td>15</td>
</tr>
<tr>
<td>animals</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>study</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>birds</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>news</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>time</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>animal</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>dinosaurs</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>colleagues</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>insects</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

The last section of this chapter provides links to additional information.

### 14.6.2 Changing the Series Names

When you display an iKnow KPI in a pivot table widget, the series names are used as the names of the rows. By default, the series names are taken from the first column of the returned values, which may not be useful in this scenario. For example, for the following KPI, the first column is the result number:
To choose a different column to use as the series names, do the following:

1. Click the Reconfigure button on the widget.
2. Click the Controls tab.
3. Click the Add button to the right of the table, which is initially empty.
4. For Type, select Hidden.
5. For Action, select Filter.
6. For Filter, select Series name column.
7. For Default Value, click the Add button.
8. Select the name of the column to use (for example, entity for the KPI shown here).
9. Click OK to add the default value.
10. Click OK to add the control.
11. Click OK to complete the widget reconfiguration.

Now the row names have changed. For example:

### 14.6.3 Configuring the Properties

When you display a KPI in a widget, you may need to reconfigure the properties shown in that widget:

- For a scorecard widget, you must configure the properties. No columns are shown by default.
- For a pivot table widget, you might need to configure the properties. All columns are shown by default.

To configure the properties:
1. Click the Reconfigure button on the widget.
2. Click the Data Properties tab.
   On this tab, you specify the properties (or columns) of the KPI.
3. To configure a property, click the Add button.
4. For Data Value, select a value from the drop-down list; this lists the result columns of the KPI.
5. Optionally type a caption into Label.
6. Optionally type a format string into Format. By default, numeric values are shown with the thousands separator used in your locale. To format the numbers without the thousands separator, type # into Format.
7. Click OK.

   After you have added each result column of the KPI, the dialog box should looks something like this, depending on your KPI:

   ![Data Properties dialog box](image)

   Set of data properties displayed by this widget

8. Click OK to complete the reconfiguration.

### 14.6.4 Adding Previous Page and Next Page Buttons

For any iKnow KPI based on a query that uses paging, that KPI defines the Previous page and Next page actions. You can configure the widget to include buttons that execute these actions. To do so:

1. Click the Reconfigure button on the widget.
2. Click the Controls tab.
3. Add a Previous Page buttons to this widget, as follows
   a. Click the Add button to the right of the table, which is initially empty.
      The system displays a dialog box where you specify the control.
   b. For Action, select Previous page.
   c. For Control Label or Icon, type Previous Page
   d. Click OK to add the control.
Use similar steps to add the **Next Page** button.

Now the dialog box looks like this:

![Image of widget properties with controls](image)

Set of controls placed on this widget

4. Click **OK** to complete the widget reconfiguration.

Now the widget includes buttons like this:

![Image of widget with buttons](image)

### 14.6.5 Example Dashboard with iKnow KPI

The SAMPLES namespace provides an example dashboard that displays an iKnow KPI. To see this dashboard:

1. In the Management Portal, access the SAMPLES namespace.
2. Click **Home,DeepSee,User Portal** and then click **View**.
   
   The system then displays the User Portal, which lists any existing public dashboards and pivot tables in this namespace.
3. Click **Aviation event reports**.
   
   The system displays the requested dashboard.

In this dashboard, the widget **Top concepts in Event Reports** displays an iKnow KPI. This widget appears as follows:
This widget displays the KPI defined in the class Aviation.KPI.TopConcepts.

**14.7 Providing Access to Dashboards**

You can provide dashboards to your users in various ways:

- You can embed a dashboard in the iFrame of a Zen page. To do so, include something like this within the `<iframe>` definition:

  ```html
  ```

  In this example, `Tutorial` is the name of the folder that contains the dashboard, and `My First Dashboard` is the name of the dashboard.

- Your application can provide direct links to dashboards.

- You can provide a user portal that has the ability to display dashboards. You can use the DeepSee User Portal or create your own.

  The DeepSee User Portal is intended for direct use by end users (in contrast to such back end tools as Studio and the Management Portal). This portal is intended for general use (and has a general appearance), despite its specific name. It is not labeled with “DeepSee.”

**14.8 See Also**

- For information on Zen pages, see *Using Zen*.

- For information on creating DeepSee dashboards, see *Creating DeepSee Dashboards*.

- For information on accessing dashboards from your application, see “Accessing Dashboards from Your Application” in the *DeepSee Implementation Guide*. 
• For information on packaging dashboard definitions into classes, see “Packaging DeepSee Elements into Classes” in the DeepSee Implementation Guide.

• For information on the DeepSee User Portal and dashboards, see the DeepSee End User Guide.
Customizing iKnow

You can customize how the iKnow semantic analysis engine loads and lists source texts. The following types of customization are supported:

- **Lister.** By default, iKnow provides several listers that correspond to common sources of data. You can create a custom lister appropriate for your data.

- **Processor.** By default, iKnow uses the processor that corresponds to the lister. You can create a custom processor and associate it with your custom lister, or you can associate an existing processor to your custom lister.

- **Converter.** A Converter is an object that can transform complex input text into the plain text expected by the iKnow engine. For example, a Converter can extract plain text from a PDF or RTF document, or select specific nodes from an XML document. By default, iKnow does not use a converter. You can create a custom converter and specify it in the Lister `SetConverter()` or `Init()` method.

A Lister can optionally specify its Configuration, Processor, and Converter. You can specify these using the `SetConfig()`, `SetProcessor()`, and `SetConverter()` methods, or specify all three using the `Init()` method.

The following example shows how to specify a custom processor and/or a custom converter using the optional Lister `Init()` method. You can specify an empty string (""") for any of the `Init()` method parameter to take the default for the specified Lister.

```plaintext
SET flister=##class(%iKnow.Source.File.Lister).%New(domId)
DO flister.Init(configuration,myprocessor,processorparams,myconverter,converterparams)
```

### 15.1 Custom Lister

iKnow provides a base lister class and five subclasses containing listers specific to different types of input sources.

In order to implement a custom lister you begin with the base Lister class, `%iKnow.Source.Lister`, and override several of its defaults.
15.1.1 Lister name

In order to be able to work with the lister, the lister needs to specify the format in which the external id for each source is presented. The external id for a lister consists of the lister name and the full reference. The full reference consists of the groupName and the localRef. An external id is shown in the following example:

```
:MYLISTER:groupname:localref
```

In this example, MYLISTER is the lister name alias. If you don’t provide an alias, the full classname of the lister class is used. To determine the alias for your lister, use the `GetAlias()` method.

A lister name alias must be unique within the current namespace. If you specify a lister name alias that already exists, iKnow generates a `$$IKListerAliasInUse` error.

15.1.2 SplitFullRef() and BuildFullRef()

You must specify a `SplitFullRef()` instance method for your custom lister. This method is used to extract the groupName and the localRef from the fullRef string. Its results are supplied to the `SplitExtId` class method. Assume your lister has an external id format like this: :MYLISTER:groupname:localref.

In this simple example, fullRef consists of the string groupname:localref, so the groupName is `$PIECE(fullRef,"\:"",1)` and localRef is `$PIECE(fullRef,"\:"",2)`. Note that this is a very simple example; it does not work if the groupName or localRef parts contain ":" characters.

You must specify a `BuildFullRef()` instance method for your custom lister. This method is used to combine the groupName and the localRef to form the fullRef string. Its results are supplied to the `BuildExtId` class method.

15.1.3 Default Processor

A Processor is a class that takes as input the list populated by the Lister, reads the corresponding source text, and directs the source data to the iKnow engine for indexing. It can, optionally, pass this source data through a Converter.

iKnow Processors are subclasses of the `%iKnow.Source.Processor` class. Each processor subclass is designed to read sources of a specific type, such as the `%iKnow.Source.File.Processor` which reads files from a directory.

Every Lister has a default processor that is capable of processing the sources from that lister. By default, it uses a class called Processor in the same package as the Lister. If there is no processor corresponding to the specified lister, or if you wish to use the generic `%iKnow.Source.Temp.Processor`, you should override the `DefaultProcessor()` method and specify the desired default processor.

15.1.4 Expand List

The `ExpandList()` method is responsible for listing all sources that need to be indexed. This method should be overridden by user-defined subclasses that implement how to scan through the particular type of source location or structures for your custom Lister. The parameters for this method are the same as those used when invoking the corresponding `AddListToBatch()` method. The parameters may differ, depending on the Lister that you implement. Make sure that the Lister-specific `ExpandList()` parameters are documented, so that a user knows which parameters to supply to the `lister.AddListToBatch()` method or the `loader.ProcessList()` method.

The `ExpandList()` parameters are as follows (in order):

- **Path**: the location where the sources are located, specified as a string.
- **Extensions**: one or more file extension suffixes that identify which sources are to be listed. Specified as a `%List` of strings.
• Recursive: a boolean value that specifies whether to search subdirectories of the path for sources.
• Filter: a string specifying a filter used to limit which sources are to be listed.

For further details, refer to Lister Parameters in the chapter “Loading Text Data into iKnow”.

15.2 Custom Processor

A processor can either copy the complete source into a temporary global for iKnow processing, or it can store a reference to the source in a temporary global. These temporary globals are used by the iKnow engine to index the text and store the results in iKnow globals.

If a Lister does not have a corresponding processor, the %iKnow.Source.Temp.Processor is the default processor. It copies the complete text of each source into a temporary global. The other supplied processors store a reference to the source in a temporary global. You can use ..StoreTemp to specify copying the source, or ..StoreRef to specify storing a reference to the source.

15.2.1 Metadata

While listing sources, the Lister is capable of extracting metadata that should be added to the sources. In order to let the system know which metadata the Lister will provide, you can call the function ..RegisterMetadataKeys(metaFieldNames). The metaFieldNames parameter is a %List containing the keys for the metadata key-value pairs. After that you can provide the metadata values by using the function ..SetMetadataValues(ref, metaValues). The metaValues parameter is a %List containing the values for the metadata key-value pairs. They should appear in the same order as the keys are listed.

After establishing the metadata in the Lister, you can access this metadata in your processor by implementing the GetMetadataKeys() method. This method should return a %List of keys from the metadata key-value pairs. In the FetchSource() method the processor can then set the appropriate values for calling ..SetCurrentMetadataValues(values), where values is a %List of the values of the metadata key-value pairs, in the same order as the keys were reported.

15.3 Custom Converter

A Converter converts source text to plain text by removing tags from the source text. Tags are non-content elements used to format the text for display or printing. For example, you might use a converter to remove tags from RTF (Microsoft Rich Text Format) files, or to extract plain text from a PDF file. A converter is invoked by the Lister and applied prior to indexing the source text. Depending on the format of your source documents, the use of a source converter is an optional step.

iKnow provides one sample converter, the subclass %iKnow.Source.Converter.Html, which you can use to remove HTML tags from source text. This is a basic HTML converter; you may need to customize your instance of this converter to support full conversion of your HTML source texts.

In order to implement a custom Converter you need to override several methods from the base converter class %iKnow.Source.Converter.

15.3.1 %OnNew

The user-provided %OnNew() method is invoked by the %New() method. It takes as its parameter a %List of any parameters that the Converter requires.
15.3.2 Buffer String

The BufferString() method will be called as many times as needed to buffer the complete document into the Converter. Each call will provide a chunk of text by means of the data parameter (max 32K). When no more data is to be buffered, the Convert() method will be called.

15.3.3 Convert

The Convert() method is responsible for processing the buffered content and converting the data into plain text (for example, RTF file conversion), or extracting the required data from the buffer (for example, node extraction from xml). The converted or extracted data will need to be buffered, as the converted data can be larger than 32K.

15.3.4 Next Converted Part

The NextConvertedPart() method is called after the Convert() method. This method must return the converted data in chunks of 32K. Every time this method is called, you need to return the next chunk. If no more data is available, this method should return the empty string ("") to indicate that it has finished extracting the converted data.
16

Language Identification

Before using iKnow on source data, you must create a Configuration. This establishes the language environment for source document content. A Configuration is independent of any specified set of source data.

16.1 Configuring Automatic Language Identification

An iKnow Configuration specifies the environment for handling source documents. A configuration defines the following language options:

- What language(s) the source documents contain, and therefore which languages to test for and which language models to apply. The available options are Dutch (nl), English (en), French (fr), German (de), Portuguese (pt), and Spanish (es). Specify a language using the ISO two-letter code. You can specify multiple languages as a Caché list structure.

- When specifying more than one language, specify a boolean value to activate automatic language identification.

The following example creates a configuration that assumes all source texts will be in English or French, and supports automatic language identification:

```cachelang
SET myconfig="EnglishFrench"
IF ##class(%iKnow.Configuration).Exists(myconfig) {
    SET cfg=##class(%iKnow.Configuration).Open(myconfig)
    WRITE "Opened existing configuration ",myconfig,"!"
} ELSE {
    SET cfg=##class(%iKnow.Configuration).%New(myconfig,1,$LISTBUILD("en","fr"),"",1)
    IF ##class(%iKnow.Configuration).Exists(myconfig) {
        WRITE "Configuration ",myconfig," now exists","!
    } ELSE {WRITE "Configuration creation error" QUIT }
}
SET cfgId=cfg.Id
WRITE "with configuration ID ",cfgId,"!
SET rnd=$RANDOM(2)
IF rnd {
    SET stat=##class(%iKnow.Configuration).%DeleteId(cfgId)
    IF stat {WRITE "Deleted the ",myconfig," configuration" }
} ELSE {WRITE "No delete this time","!
```

16.2 Using Automatic Language Identification

iKnow performs automatic language identification on a per-sentence basis. When the current configuration has activated automatic language identification, iKnow tests each sentence in each source text to determine which of the languages
specified in the Configuration is the language used in that sentence. This identification is a statistical probability. This has the following consequences:

- If a sentence contains text in more than one language specified in the Configuration, iKnow will assign the sentence to what it determines is the predominant language of the sentence.
- If a sentence is in a language not specified in the Configuration (or a language not supported by iKnow), iKnow will assign the sentence to one of the specified Configuration languages.

iKnow subsequently uses this language determination in determining CRCs and other iKnow analysis.

Thus, source texts and sentences within a source text can be in different languages. iKnow automatically determines which language model to apply. Automatic language identification also assigns a confidence level in its language identification as an integer indicating a percentage. These range from 100 (complete confidence) to 0 (indeterminate). If automatic language identification is not active, all sentences are assigned a confidence level of 0.

The following example shows automatic language identification of sentences. The `GetLanguage()` method returns the language as a two character abbreviation (in this case, “en”) and the confidence level as a percentage between 0 and 100. Note that the confidence level is rarely (if ever) 100%.

```plaintext
Configuration
SET myconfig="EnFr"
IF ##class(%iKnow.Configuration).Exists(myconfig)
{SET cfg=##class(%iKnow.Configuration).Open(myconfig) }
ELSE {SET cfg=##class(%iKnow.Configuration).%New(myconfig,1,$LISTBUILD("en","fr"),"",1)
DO cfg.%Save() }
SET cfgId=cfg.Id
DomainCreateOrOpen
SET dname="mydomain"
IF ##class(%iKnow.Domain).Exists(dname)
{SET domoref=##class(%iKnow.Domain).Open(dname)
GOTO DeleteOldData }
ELSE {SET domoref=##class(%iKnow.Domain).%New(dname)
DO domoref.%Save() 
GOTO ListAndLoader }
DeleteOldData
SET stat=domoref.DropData()
IF stat =- 1 { WRITE "DropData error ",$System.Status.DisplayError(stat)
QUIT }
ELSE {SET domId=domoref.Id
GOTO ListAndLoader }
ListAndLoader
SET myloader=##class(%iKnow.Source.Loader).%New(domId)
SET mylister=##class(%iKnow.Source.File.Lister).%New(domId)
SET stat=mylister.SetConfig(myconfig)
IF stat =- 1 { WRITE "SetConfig error ",$System.Status.DisplayError(stat)
QUIT }
SET stat=myloader.SetLister(mylister)
SET install=$SYSTEM.Util.InstallDirectory()
SET dirpath=install_"dev\tutorials\iknow\data\reviews"
SET stat=myloader.ProcessList(dirpath,$LB("txt"),0,"")
IF stat =- 1 { WRITE "Loader error ",$System.Status.DisplayError(stat)
QUIT }
GetOneSource
DO ##class(%iKnow.Queries.SourceAPI).GetByDomain(.result,domId)
FOR i=1:1:10 {
IF $DATA(result(i)) { 
SET intId = $LISTGET(result(i),1)
SET extId = $LISTGET(result(i),2)
SET myconf=0
SET numSents = ##class(%iKnow.Queries.SentenceAPI).GetCountBySource(domId,result(i))
WRITE !,"sentence ",numSents," sentences"!
GETSentencesInSource
SET sentStat=##class(%iKnow.Queries.SentenceAPI).GetBySource(.sent,domId,intId)
IF sentStat=1 { 
SET i=1
WHILE $DATA(sent(i)) {
SET sentnum=$LISTGET(sent(i),1)
WRITE "sentence number ":sentnum
WRITE " language ":lang," confidence ":myconf,!
SET i=i+1
}
ELSE { WRITE !,"That's all folks!" }
}
}````
16.2.1 Overriding Automatic Language Identification

You can use the LanguageFieldName domain parameter to override Automatic Language Identification. If activated, this parameter determines which language to apply by accessing a metadata field for each source. This metadata field contains the ISO language code. If the metadata field data is present, Automatic Language Identification is overridden for that source. If the metadata field is empty or invalid, Automatic Language Identification is used for that source. The LanguageFieldName domain parameter is inactive by default. For further details, refer to the Domain Parameters appendix of this manual.
Domain Parameters

This appendix lists the available domain parameters. Domain parameter names are case-sensitive. Each domain parameter has a %IKPublic macro equivalent (for example, $$$IKPFULLMATCHONLY). The recommended programming practice is to specify a domain parameter by its macro equivalent, not its parameter name. For information on setting domain parameters, refer to “Defining an iKnow Domain”.

Domain parameters are divided into two groups, Basic and Advanced. Basic parameters are useful for customizing iKnow default behavior. Advanced parameters significantly change iKnow behavior and performance, and should be used with caution.
Table I–1: Basic Domain Parameters
### Domain Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultConfig</td>
<td>A string specifying the name of the iKnow Configuration used for the domain. By default, no Configuration is assigned to a domain; this is indicated by returning the string DEFAULT. You can specify the name of any Configuration defined in the current namespace. The Configuration must exist when you assign it to this parameter. If the Configuration does not exist, this parameter remains unchanged. The DefaultConfig value can be overridden by SetConfig() or ProcessBatch().</td>
</tr>
<tr>
<td>EnableFilterCache</td>
<td>A boolean parameter specifying whether the results of a filter should be cached for reuse. A filter selects a subset of the indexed sources based on some criteria; these selected sources are supplied to a query. If set to 1, iKnow caches the results of a filter and reuses the list of selected sources for each query that specifies that filter. This can improve performance when the same filter is used multiple times. If set to 0, the filter results are not cached; each query that specifies a filter executes that filter against the full set of indexed sources. This is preferable if the set of indexed sources (or associated metadata) is likely to change between invocations of the filter, or if each filter is only invoked once. The default is 1.</td>
</tr>
<tr>
<td>EnableNgrams</td>
<td>A boolean parameter. If set to 1, iKnow generates n-grams for the domain. n-Grams are used for Similarity entity matching within words. If set to 0, iKnow matches only parts, whole words or the beginning letters of words. The default is 0. At the domain level, you can only change the EnableNgrams setting for an empty domain (a domain the does not yet contain iKnow data). At the systemwide level, you can only change the EnableNgrams setting if there are no domians defined on the system. n-Gram matching greatly increases the size of the data stored by iKnow, and can have a significant performance impact. It should only be enabled when required. You should not use n-gram matching with most languages. However, matching operations on German text often requires n-gram matching.</td>
</tr>
<tr>
<td>IgnoreDuplicateExtIds</td>
<td>A boolean parameter. If set to 1, iKnow does not log an error if a source is loaded that has the same external Id as an already-loaded source. 1 is the recommended setting when loading sources from a location previously loaded in order to include the new sources added since the last load. The default is 0.</td>
</tr>
</tbody>
</table>
### IgnoreEmptyBatch

**IKPIGNOREEMPTYBATCH:** A boolean parameter. If set to 0, the Loader issues a **IKNothingToProcess** error when a batch load is specified that specifies no sources. If set to 1, the Loader does not generate this error. The default is 0.

### MAT:DefaultProfile

**IKPMATDEFAULTPROFILE:** This parameter takes the name of a user-defined matching profile that you wish to establish as the default for the domain. The matching profile must exist when you assign it to this parameter. If the user-defined matching profile is defined as namespace-wide (not specific to a domain), you must add a zero colon (0:) preface to the name. For example, "0:NoDomainProfile". If you do not set this parameter, the iKnow default matching profile is used as the domain default. For further details, refer to [Defining a Matching Profile](chapter) in the “Smart Matching: Using a Dictionary” chapter.

You can override the domain default matching profile by specifying a custom matching profile in the **MatchSource()** or **MatchSources()** methods.

### MAT:SkipRelations

**IKPMATSkipRELS:** A boolean parameter for dictionary matching. A value of 1 (the default) specifies that only concepts, not relations, will be matched during entity matching. (Relations are matched during CRC and path matching operations.) Skipping relation entity matching can significantly improve performance. A value of 0 performs relation entity matching. You should only set this parameter to 0 if your dictionary includes single-entity terms that are relations.
| **SortField** | $$$IKPSORTFIELD: A boolean parameter. Every iKnow entity has two integer counts associated with it: frequency (number of occurrences of an entity in all sources) and spread (number of sources in which the entity occurs). If set to 0, the iKnow default is to sort by frequency ($$$SORTBYFREQUENCY). If set to 1, the iKnow default is to sort by spread ($$$SORTBYSREAD). The default is 0. At the domain level, you can only change the SortField setting for an empty domain (a domain that does not yet contain iKnow data). At the systemwide level, you can only change the SortField setting if there are no domains defined on the system. Alternatively, you can change this sort order doma- wide by specifying a second parameter (sortField) to the Create() or GetOrCreateId() method of the %iKnow.Domain class. 0=sort by frequency (the default). 1=sort by spread. The sort order default should only be changed when required. You can only change the sort order default for an empty domain (a domain that does not yet contain iKnow data). In certain individual queries you can also specify sort order ($$$SORTBYFREQUENCY or $$$SORTBYSREAD) or use the current domain default ($$$SORTBYDOMAINDEFAULT). |
| **Status**    | $$$IKPSTATUS: A boolean parameter. If set to 1, iKnow displays detailed status information on the progress of the source loading process. If set to 0, iKnow does not display this information. The default is 0. |
Table I-2: Advanced Domain Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityLevelMatchOnly</td>
<td>$$\text{IKPENTLEVELMATCHONLY: A boolean parameter that is used to limit the types of match operations performed when matching against a dictionary. By default iKnow matches entities, CRCs, paths, and sentences. You can set this parameter to limit matching to entities only. Note that this may result in a much larger number of match results. The default is 0. Changing this parameter affects any subsequent match operations for this domain. Therefore, sources already matched before you changed this parameter must be explicitly re-matched to reflect this change.}$$</td>
</tr>
<tr>
<td>FullMatchOnly</td>
<td>$$\text{IKPFULLMATCHONLY: A boolean parameter that is used to limit the types of match operations performed when matching against a dictionary. You can set this parameter to restrict matching to exact matches only. When this option is set (1), partial matches and disordered matches are ignored. The default is 0. Changing this parameter affects any subsequent match operations for this domain. Therefore, sources already matched before you changed this parameter must be explicitly re-matched to reflect this change.}$$</td>
</tr>
<tr>
<td>Jobs</td>
<td>$$\text{IKPJOBS: An integer that specifies the number of consecutive jobs and processors iKnow uses for indexing. When larger than 1, iKnow invokes multiple jobs to perform an indexing operation, enabling it to take advantage of multiple processors. iKnow sets this parameter's default based on the number of available processors. Users should not change this value.}$$</td>
</tr>
<tr>
<td>LanguageFieldName</td>
<td>$$\text{IKPLANGUAGEFIELDNAME: A string that specifies the name of a metadata field. When set to an existing metadata field that was populated during source loading, iKnow uses that metadata field's value (if set) as the language to be used when processing the corresponding source. This option overrides automatic language identification. The metadata field value must be a two-letter ISO language code (see $$\text{IKLANGUAGES}$$) for a language that has been specified in the current configuration object.}$$</td>
</tr>
<tr>
<td>MetadataAPI</td>
<td>$$\text{IKPMETADATAAPI: A string that specifies which metadata API class to use to extend %iKnow.Queries.MetadataI. The default is %iKnow.Queries.MetadataAPI. You can only change the MetadataAPI setting for an empty domain (a domain the does not yet contain iKnow data).}$$</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>QUERY:MinTopConceptLength</strong></td>
<td>$$\text{IKPMINTOPCONCEPTLENGTH}: \text{An integer that specifies the smallest concept (fewest number of characters) that a GetTop() query can return. This parameter is used to filter meaninglessly short concepts from the GetTop() result. The default is 3, specifying that concepts that are 3 letters in length or larger are returned by GetTop(). This minimum character count is inclusive of spaces between words and punctuation symbols in a concept.}$$</td>
</tr>
<tr>
<td><strong>SimpleExtIds</strong></td>
<td>$$\text{IKPSIMPLEEXTIDS}: \text{A boolean parameter that is used to specify the format for external IDs for sources. If set to 0, iKnow stores the full reference as the external ID. If set to 1, iKnow stores the local reference as the external ID. The default is 0. You can only change the SimpleExtIds setting for an empty domain (a domain the does not yet contain iKnow data).}$$</td>
</tr>
<tr>
<td><strong>SkipExtIdCheck</strong></td>
<td>$$\text{IKPSKIPEXTIDCHECK}: \text{A boolean parameter that specifies whether to check for duplicate external IDs. If set to 1, iKnow skips checking whether a duplicate external ID already exists when loading sources. If set to 0, iKnow checks for duplicate external IDs. The default is 0.}$$</td>
</tr>
</tbody>
</table>
This appendix describes how to use older, deprecated form of integration between iKnow and DeepSee cubes. It discusses the following topics:

- Overview
- How to generate iKnow source metadata fields from an associated DeepSee cube
- How to generate DeepSee cube classes for an iKnow domain
- Additional sources of information

For information on using the newer form of integration, see “Using Unstructured Data in Cubes” in Defining DeepSee Models.

Also see the chapter “iKnow KPIs and DeepSee Dashboards.”

B.1 Overview

This section provides an overview of the older, deprecated form of integration between iKnow and DeepSee cubes. DeepSee is an analytics engine and reporting platform that is built into Caché in the same way that iKnow is. The primary purpose of DeepSee is to enable you to create interactive dashboards that you can embed on Zen pages or that users can access via the DeepSee User Portal (which is intended for all end users, not just end users of DeepSee).

B.1.1 Background

A DeepSee cube is a data model that enables you to create DeepSee queries by drag and drop actions. The queries are known as pivot tables. You create pivot tables so that you can display them on DeepSee dashboards. When the dashboard is displayed, the query is executed.

Each cube is based on a source table. You build a cube, and this process generates a fact table that has the same number of records as the source table (or possibly fewer if the model skips some records deliberately). The queries use the fact table rather than the source table. DeepSee provides multiple complementary mechanisms to keep the fact table synchronized with the source table.
The following figure shows an example pivot table that shows the number of patients and the average allergy count per patient, grouped by age and gender.

<table>
<thead>
<tr>
<th>Age Group</th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Patient Count</td>
<td>Avg Allergy Count</td>
</tr>
<tr>
<td>0 to 29</td>
<td>630</td>
<td>0.60</td>
</tr>
<tr>
<td>10 to 19</td>
<td>756</td>
<td>0.66</td>
</tr>
<tr>
<td>20 to 29</td>
<td>651</td>
<td>0.64</td>
</tr>
<tr>
<td>30 to 59</td>
<td>815</td>
<td>0.63</td>
</tr>
<tr>
<td>40 to 49</td>
<td>728</td>
<td>0.68</td>
</tr>
<tr>
<td>50 to 59</td>
<td>586</td>
<td>0.58</td>
</tr>
<tr>
<td>60+</td>
<td>397</td>
<td>0.64</td>
</tr>
<tr>
<td>70 to 79</td>
<td>304</td>
<td>0.58</td>
</tr>
<tr>
<td>80+</td>
<td>217</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Using this example as a reference, let us discuss the key terms in DeepSee cubes:

- A **level** is used to group the records. In this case, we are grouping patients. Each record in the source table corresponds to one patient, as does each record in the fact table.

- A **measure** is a value displayed in the body of the pivot table. It is based on values ultimately taken from the source data. For a given context, a measure aggregates the values for all applicable records and represents them with a single value (typically by summing them).

  For example, the measure **Patient Count** is the number of patients, and the measure **Avg Allergy Count** is the average number of allergies per patient.

A cube can also define **detail listings**, whose purpose is to display lowest-level data associated with the current context. To access a listing, you select one or more cells and click the Listing button (not shown in the previous picture). DeepSee then displays a table that lists selected fields for all the records used in the current context.

### B.1.2 Relationship between a Cube and iKnow Sources

The iKnow/cube integration makes the following assumption: there is a single iKnow source associated with each record of the cube’s source class. That is, the cube serves as additional information about the iKnow sources. (Phrased differently, the iKnow sources contain additional information associated with each record of the cube’s source class.)

The goal is to be able to use DeepSee and iKnow filters interchangeably where possible.

You can use utility methods to do the following tasks:

- To generate iKnow source metadata fields from an existing DeepSee cube
- To generate DeepSee cube classes for an iKnow domain
B.2 Generating iKnow Source Metadata Fields for an Associated DeepSee Cube

If a DeepSee cube uses a source class that includes a reference to iKnow data source, you can use the levels of that cube as iKnow source metadata fields for those sources. This means that you can use DeepSee filters within iKnow queries (including when you access those queries via the DeepSee KPI mechanism as described in see “Using Unstructured Data (iKnow)” in Defining DeepSee Models.).

This section assumes that the cube is defined and has been built. The cube does not need to be modified or rebuilt.

B.2.1 Prerequisites

Before you can generate iKnow source metadata fields for a cube:

1. Load the iKnow sources.

   When you do so, make sure that the external ID of each source can be readily associated with a specific record of the source class of the cube.

   For example:
   
   • If you are using a file lister, you might make sure that the file names match the patient identifiers.
   • If you are using an SQL lister, you might make sure that the IDs of the records match the patient identifiers.

2. Make sure that the source class of the cube includes a property whose value is the external ID of the corresponding iKnow source.

   For example, this class could include a calculated property whose value follows the naming convention established in the preceding step.

   ```
   Property DocumentId As %String [ Calculated ];
   Method DocumentIdGet() As %String
   {
      quit "\FILE:\c:\patient-data\patient\"..PatientNumber\".txt"
   }
   ```

B.2.2 Generating the Metadata Fields from the Cube

To generate the metadata fields, you use the `GenerateMDFieldsFromDSDims()` method of `%iKnow.DeepSee.Utils`. This method has the following signature:

```plaintext
ClassMethod GenerateMDFieldsFromDSDims(domainId As %Integer, cubeClassName As %String, ByRef levels = 1, extIdProperty = "", killExistingMDFields As %Boolean = 0) As %Status
```

Where:

- `domainId` is the integer ID of an iKnow domain.
- `cubeClassName` is the name of the cube class.
- `levels` controls which levels are processed. If you omit this argument, the system processes all levels, skipping the ones that are not supported by the iKnow engine.

This argument, if specified, must be a multidimensional array of the following form:
ValueNode

Either 0 or 1. If you specify 1, the iKnow metadata field is generated as a bitstring, which is suitable when you expect only a small number of values.

arrayname(level_name) where level_name is the complete MDX identifier of the level, in double quotes. For example: "[DocD].[H1].[Doctor]"

• extIdProperty is the property you added in the previous subsection. For example: DocumentId

More generally, extIdProperty is a property (in the cube’s source class) whose value equals the external ID of an iKnow source in the given domain.

• killExistingMDFields specifies whether to remove any existing iKnow metadata fields.

B.2.3 Using DeepSee Filters with an iKnow KPI

If you also create iKnow KPIs in this domain, then you can use DeepSee filters in the KPI, provided that the KPI and the cube are both associated with the same iKnow domain.

To use this mechanism:

• Display the KPI in a widget on a dashboard.
• In the DeepSee Analyzer, create a pivot table.
• On the same dashboard, add a widget that displays the pivot table. Configure this widget to use one or more levels of the cube as filters, and specify the filter target as * (all widgets).

B.3 Generating DeepSee Cubes for an iKnow Domain

This release provides a preliminary version of a utility that generates DeepSee cubes from iKnow engine results. The generated cubes enable you to view entities, entity occurrences, and match results grouped in various ways. With a DeepSee cube, you can perform analysis and exploration via drag and drop actions in the DeepSee Analyzer.

You can use these DeepSee cubes in the same way that you use other DeepSee cubes; you can explore them in the Analyzer, create pivot tables, and add those pivot tables to DeepSee dashboards.

This section describes how to generate and (briefly) how to use DeepSee cubes, given an iKnow domain. It discusses the following topics:

• How to generate the cubes
• A brief look at the cubes
• How to rebuild the cubes

If you can achieve a required result through an existing iKnow query, use that query rather than trying to create an equivalent query with the generated cubes. The iKnow queries are optimized to use internal and precalculated indices and are faster as a consequence.

B.3.1 Generating the Cubes

To generate DeepSee cubes for an iKnow domain, you use the GenerateSourceObjectAndCubes() method of %iKnow.DeepSee.Utils. This method has the following signature:
class method GenerateSourceObjectAndCubes(domainId,
    packageName="User",
    createCube=2,
    overwriteExisting=0,
    buildCubes=0) as %Status

Where:

•  **domainId** is the integer ID of an iKnow domain.
•  **packageName** is the package in which to place the generated classes.
•  **createCube** controls which classes are generated:
  – If **createCube** is 0, Caché generates and compiles the classes on which the cubes are based, but does not generate other classes. You might use this option if you wanted to create DeepSee cubes manually in the DeepSee Architect. These classes define mappings to access the iKnow data.
  – If **createCube** is 1, Caché also generates and compiles a cube that represents the iKnow sources.
  – If **createCube** is 2, Caché also generates and compiles a cube that represents unique entities, and another cube that represents entity occurrences.
  – If **createCube** is 3, Caché also generates and compiles a cube that represents matching results.
•  **overwriteExisting** specifies whether to overwrite any existing classes.
•  **buildCubes** specifies whether to build the generated cubes. This has no effect if **createCube** is 0.

The phrase *build a cube* refers to the process of populating the structures that the cube uses. This process makes it possible to use the cube.

### B.3.2 A Brief Look at the Analyzer and the Cubes

This section briefly introduces the Analyzer and the generated cubes. To access the Analyzer and display one of these cubes:

1. Click **Home, DeepSee, Analyzer**.
2. Click the Change button.
3. Click the cube name.
4. Click **OK**.

For example, the entity occurrence cube provides information about the occurrence of entities in the source documents. The name of this cube ends with **EntityOccurrences**. When you view this cube, the left area of the page displays its contents, as follows:
This cube defines one measure, Count, which counts entity occurrences. This measure is used by default.

The **Dimensions** section shows three dimensions. GeneratedD1CubesSource and Entity are displayed differently from Entity Type because they are defined differently. You can use this dimensions as follows:

- **Entity Type** — This dimension enables you to categorize entity occurrences by entity type. If you expand this folder and the Entity Type folder within it, the system displays the following:

  - **Entity Type**
    - All Entity Types
    - Entity Type
      - concept
      - relation

- **GeneratedD1CubesSource** — This dimension enables you to categorize entity occurrences by the source document to which they belong. If you expand this folder, the system displays the following:

  - **GeneratedD1CubesSources**
    - **Source**
      - All Sources
      - Source
    - **DateIndexed**
      - All DateIndexed
      - DateIndexedYear
      - DateIndexedMonthYear
      - DateIndexedDayMonthYear

  If you expand the SourceID folder, the system displays something like this, depending on your data:

  - **GeneratedD1CubesSources**
    - **Source**
      - All Sources
      - SourceID
        - **ExternalID**
          - :FILE: c:\test\tknow-
          - :FILE: c:\test\tknow-
          - :FILE: c:\test\tknow-
          - :FILE: c:\test\tknow-
          - :FILE: c:\test\tknow-
          - :FILE: c:\test\tknow-

- **Entity** — This dimension enables you to categorize entity occurrences by entity value. If you expand this folder and the top-level folders in it, the system displays the following:

  - **Entity**
    - **Cluster Length**
      - Any length
      - **Cluster Length**
    - **Entity Value**
      - Entity

  If you expand the Entity folder, the system displays something like this, depending on your data:
The right area of the page looks like this:

![Pivot Table](image)

The area at the bottom is a simple pivot table. It displays the total value for Count. In this example, there are 7118 entity occurrences in the domain.

The area above that enables you to create pivot tables. You can drag and drop items from the left to this area, and the system automatically modifies the pivot table. For example, if you drag Entity (the inner Entity folder) to the Rows box, the system displays the following:

![Pivot Table](image)

This pivot table shows both relations and concepts because we have not applied any filtering. To filter the pivot table to show only concepts:

1. Expand the **Entity Type** dimension on the left:

![Entity Type](image)
2. Drag concept and drop it into the Filters box on the right.
   This immediately updates the display.

Now you might see something like this:

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Entity Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept</td>
<td>ancient viruses</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>and</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>animal</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>animal care</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>animal pictures</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>animal's nerve cell</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>animal-induced response</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>animals</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>animals getting fatter</td>
<td>1</td>
</tr>
</tbody>
</table>

The EntityOccurrences cube defines a detail listing. To access it, select one or more cells and click the Listing button (In this example, we click the cell for ancient viruses.) The system then displays something like the following:

<table>
<thead>
<tr>
<th>#</th>
<th>EntOccId</th>
<th>EntTypeId</th>
<th>EntUnId</th>
<th>EntityValue</th>
<th>CorpuFrequency</th>
<th>CorpusSpread</th>
<th>Sourcedl</th>
<th>ExternalId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5420</td>
<td>6</td>
<td>253</td>
<td>ancient viruses</td>
<td>2</td>
<td>1</td>
<td>23</td>
<td>FILE: \wet\iKnow\input1\news3.txt</td>
</tr>
<tr>
<td>2</td>
<td>5682</td>
<td>6</td>
<td>253</td>
<td>ancient viruses</td>
<td>2</td>
<td>1</td>
<td>23</td>
<td>FILE: \wet\iKnow\input1\news3.txt</td>
</tr>
</tbody>
</table>

**B.3.3 Rebuilding the Cubes**

After you process more sources for this iKnow domain, it is not necessary to regenerate the classes but it is necessary to rebuild the DeepSee cubes. It is also necessary to rebuild the cubes in a specific order:

1. Rebuild the sources cube, which is the cube whose name ends in Sources.
2. Rebuild the other cubes, in any order.

To rebuild a cube, open it in the Architect and click Rebuild.

**B.4 See Also**

- For information on Zen pages, see Using Zen.
- For information on creating DeepSee dashboards, see Creating DeepSee Dashboards.
- For information on accessing dashboards from your application, see “Accessing Dashboards from Your Application” in the DeepSee Implementation Guide.
- For information on packaging dashboard definitions into classes, see “Packaging DeepSee Elements into Classes” in the DeepSee Implementation Guide.
- For information on the DeepSee User Portal and dashboards, see the DeepSee End User Guide.
For information on the DeepSee Architect, see *Creating DeepSee Models*. 