AltaVista Search Intranet Developer's Kit

April 1999

This manual is a compilation of the HTML files for the product in book format.

Revision/Update Information: Version 2.6

COPYRIGHT INFORMATION

Contents

1

Product Overview

The AltaVista™ Search Developer's Kit Version lets you build your own search and retrieval application or add AltaVista Search-powered search capabilities to database applications and file repositories. Users can find what they need quickly and easily without special database training. The software also provides system integrators and software developers with the tools they need to integrate the AltaVista Search engine technology into custom applications designed to provide search and retrieval capabilities for data repositories not supported by standard AltaVista Search products.

Typical examples of these kinds of data repositories are:

Type of Database	Description
Unstructured	Contain discrete files. Shared folders and directories containing large numbers of documents on LANs are examples of unstructured repository.
Structured	Contain fielded data. Database applications like Oracle, Sybase, Ingres, Microsoft Access, SQL and DB2 are examples of structured repository. These structured repositories are also not web-based.

You can also use AltaVista Search Developer's Kit to index Gopher sites.

Note: The Developer's Kit does **not** provide a web crawler or user interface.

Requirements

The following are the minimum hardware and software requirements for installing the AltaVista Search Developer's Kit:

Requirements	Description
Hardware	Any Alpha system running either Microsoft® Windows NT® Version 4.0 or *Compaq Tru64 UNIX* (Digital UNIX™) Version 4.0 Intel™ Pentium system with a 133 MHz processor running Microsoft Windows NT Version 4.0. Windows 95 and Windows 98 are also supported operating systems for Intel hardware. Sun® SPARC system running Solaris® Version 2.5.1, 2.6 or 2.7 IBM™ RS6000 system running AIX version 4.21 or later

Requirements	Description
	Intel™ Pentium System running Red Hat Linux Version 5.0
Software	C language compiler and standard libraries. Or Microsoft's Visual Basic A web browser for viewing the documentation.
Application Runtime	Sufficient RAM to get desired performance which is dependent on application behavior and index size. There is no specific minimum requirement for RAM. Approximately 1 GB of disk space after installation for building and storing a moderately-sized index

Components

The Developer's Kit includes these components:

Indexing engine — This is the same indexing engine used by the AltaVista Search Intranet product and the AltaVista Public Search Service on the World Wide Web.

APIs — Routines that allow applications to access and manipulate the AltaVista Search index. The Developer's Kit provides the following language support:

- C language
- Visual Basic

Converter libraries and converter API — A document converter API that contains document conversion technologies from Inso Corporation, Adobe Systems, Inc., and Compaq Computer Corporation. Using the libraries and the API, you can convert various document types to text. Only PDF documents can be converted to HTML at this time.

AltaVista Search Intranet compatibility API — An API that provides compatibility with AltaVista Search Intranet V2.3 by allowing the Developer's Kit applications to read data from and write data to indexes created by the AltaVista Search Intranet product. This new feature is available with the C API only and runs on the following operating systems:

- Any Alpha system running either Microsoft® Windows NT® Version 4.0 or Compaq Tru64 UNIX (Digital UNIX™) Version 4.0
- Intel™ Pentium system with a 133 MHz processor running Microsoft Windows NT Version 4.0.
- Sun® SPARC system running Solaris® Version 2.5.1, 2.6 or 2.7

Documentation — Covers descriptions of the APIs, API usage, along with project planning and programming suggestions.

Multi-purpose-programming examples

Most programming options in C and Visual Basic are coded in the sample programs. These coding examples can be copied and modified by developers. The sample programs are compiled for you and are available to run on the appropriate operating systems:

- C API - Compaq Tru64 UNIX, Sun Solaris, IBM AIX, or Linux on Intel IX86(avs_sample), and Microsoft Windows NT (avs_sample.exe).
- Visual Basic - on Microsoft Windows NT (vb_sample.exe)

There are other examples provided as samples for interfacing the Developer's Kit to additional programming languages:

- A Java JDirect implementation
- C++ implementation with includes a sample database application using Data Access Object (DAO) and the Developer's Kit
- A Tcl implementation

Development license

Allows developers to create, demonstrate, and pilot their AltaVista Search powered solution on one server. (Runtime licenses are required to implement and sell the AltaVista Search-powered solution.)

The AltaVista Search Developer's Kit does not provide a user or query interface to the index. This is part of the development partner's added value. You are free to use any user interface model that meets your customer's needs, for example, Web browsers, Visual Basic-based UIs, existing end user applications, and so forth.

The AltaVista Search C library lets you create and maintain an inverted word index. You can make calls to the AltaVista Search index from any language that links with C, or use Visual Basic on the Microsoft Windows NT platform.

New Features in Version 2.6

The following table lists the new features that have been added to the AltaVista Search Developer's Kit:

New Features	Description
Document Converter Libraries	A new document converter library, which converts various document types to text, is available for the C and Visual Basic APIs. The sample C program contains an example that uses the new converter API calls. This feature is available on the following: any Alpha system running either Microsoft® Windows NT® Version 4.0 or Compaq Tru64 UNIX (Digital UNIX) Version 4.0; Intel™ Pentium system with a 133 MHz processor running Microsoft Windows NT Version 4.0.; or Sun® SPARC system running Solaris® Version 2.5.1, 2.6 or 2.7. For more information, see Converting Documents for Indexing.

New Features	Description
AltaVista Search Intranet compatibility API	An API that provides compatibility with AltaVista Search Intranet V2.3 by allowing the Developer's Kit applications to read data from and write data to indexes created by the AltaVista Search Intranet product. This new feature is available with the C API only and runs on any Alpha system running either Microsoft ® Windows NT ® Version 4.0 or Compaq Tru64 UNIX (Digital UNIX) Version 4.0; Intel™ Pentium system with a 133 MHz processor running Microsoft Windows NT Version 4.0.; or Sun® SPARC system running Solaris® Version 2.5.1, 2.6, or 2.7. For more information, see Using the AltaVista Search Intranet Compatibility API.
New Platforms Available	The AltaVista Search Developer's Kit is available on IBM RS6000 running AIX Version 4.21 or later, as well as, an Intel Pentium system running Red Hat Linux Version 5.0. The converter libraries are not available with these platforms.
New word weighting option	A new search option allows you to eliminate the higher weighting of words occurring at the beginning of a document.
Wildcard feature available with word count.	The word count feature now supports the use of wildcards. The wildcard characters '?', '*', and '**' can be used to represent 1, 0 to 5, and 0 to unlimited characters, respectively.
New sample programs	To demonstrate the use of the new conversion technologies, the Visual Basic sample program and the C program have been updated. The sample files and their executable files, are contained in the Developer's Kit. The C sample program also contains an example of using the AVSI compatibility API

General Indexing Process

The AltaVista Search programming interface implements a number of procedures for managing text indexes and document filters. You can use the programming interface to do the following things:

- Create a new index.
- Optionally convert your documents to text.
- Add *documents* to the index .
- Use *filters* as helper procedures to parse the contents of a document and customize the way it is indexed. The filters that you use depend on the type and format of the document you are including in your index.
- Using your own query interface, submit queries to the index and retrieve documents that match the queries.
- Delete documents from the index, or replace existing documents with updated versions.
- Periodically write the contents of the in-memory index to disk, and merge the on-disk information into a single, streamlined file.

The C Programmer's Reference and the Visual Basic Programmer's Reference provide complete details on every procedure and function in the APIs.

2

Indexing, Searching, and Converting

This section describes the processes of indexing, searching, and converting. It gives a conceptual overview of each process, and then step-by-step instructions on the following:

- Creating the Index
- Searching the Index
- Converting Documents for Indexing (optional)
- Using the AltaVista Search Intranet Compatibility API

Creating the Index

Creating and searching your index can be made easy if you (as an AltaVista application programmer) adopt a model of how to handle the material to be indexed and for which you will subsequently search. The *AltaVista Model of Indexing* uses documents which, in turn, consist of a sequence of words. Each word occupies a location within the document, and these locations are sequential for adjacent words.

The Indexing Method

To index a document, your application calls the indexer for each word in the document, passing an integer location along with the word to indicate where the word is put. It then calls a procedure to give the indexer some identification data that can be retrieved for documents matching a query, for example, in title, filename, URL, and so forth. The integer locations can be anything you want, but, normally, you would number the words in each document sequentially. The integer location for each word is then just its number. This indexing method takes about 30% of the size of the original document. It allows phrased queries and fielded queries.

What is a Word?

The definition of a word in your index depends on the character set that you have assigned to the index. It is necessary that all calls to the API use the same character set. You can index any character string you want (with some restrictions) as a *word*, using **avs_addliteral**, and you can find the documents with these words using the advanced boolean expression in the form {word}.

The **avs_addword** algorithm treats any contiguous sequence of alphanumeric characters as a word. For each word to be indexed, **avs_addword** does the following:

- Enters the word into the index at the current or next location

- Determines if there is any capitalization in the word, and if so, also indexes a lower-cased version
- Determines if there is a common mapping from any of the characters in the word to ASCII, and if so, indexes one or two additional versions of the word using these mappings.

For non-alphabetic languages however, this algorithm is modified. Currently, each non-alphabetic, non-digit, non-separator character is treated as a word by itself. Special query processing algorithms are employed to efficiently locate the character sequences that users of the language would normally identify as words.

The Importance of Locations

AltaVista Search uses location values for the following:

- Matches words to documents
- Matches fields within documents
- Determines the adjacency of words (for example, phrase searches)

The boundaries between documents and words are important for finding and returning meaningful search results. In addition, word location is important for processing advanced queries where the position of certain words in relation to each other is important (for example, in searching for phrases, in proximity searches, or in searching for certain words within a specified field).

It is critically important to assign location values properly during indexing. The assigning of locations is fully automatic in the simplest case where **avs_addword** does all the work. In this case the words of the document are laid out end to end and are numbered sequentially starting with the value returned by **avs_newdoc** or **avs_startdoc**. The same is true for field boundaries and for *values* (indexed quantities like dates that can be range-searched).

The following figure shows how two very short documents would be stored in the index database.

	Document1			Document2					
Word	this	is	a	short	document	Another another	even	shorter	one
Location	1	2	3	4	5	6	7	8	9

As the figure illustrates, each word is actually stored as a word-location pair. The index also contains information about the beginning and ending locations of each document. Document1 starts at location 1, and Document2 starts at location 6.

In Document2, the first word contains an uppercase letter, so the word is indexed twice: once with case preserved and once in all lowercase. Both versions of the word are at the same location, so that the word would be found appropriately regardless of whether a query is case sensitive or case-insensitive.

Tracking the Location of the Documents

In the C API, the **avs_newdoc** procedure, in cooperation with the filter, keeps track of the starting and ending location of documents in the index. When the **avs_newdoc** or **avs_startdoc** procedure calls the filter, it passes the filter a location at which to start adding words to the index. When the filter starts adding words, it in turn passes the starting location to the

avs_addword procedure. Based on the starting location, the **avs_addword** procedure increments the location number for each word that it indexes, thereby assigning each word a unique virtual address in the indexed document.

If the filter calls **avs_addword** multiple times (for example, once for each line in the document), the filter must increment the starting location each time by the number of words indexed in the previous **avs_addword** procedure.

When the filter completes its work, it returns the total number of words that it added to the index. This also allows filters to be nested.

Steps to Indexing

Your application should follow these basic steps to create an index:

1. Optionally use the document conversion API to convert documents to text before indexing.
2. Create the directory for your index. An AltaVista index needs a place on disk to maintain its storage. When the C API mentions *index path*, it is referring to the storage area of the index. You should create and reserve a directory for the use of each index, and pass the path name of this directory to **avs_open** in order to access the index.
3. Optionally, define value types with **avs_define_valtype** or **avs_define_valtype_multiple** to be used for adding searchable numeric values or ranking terms to the documents to be indexed.
4. Open the index in mode *rw*, using avs_open

   The C API supports a parameter block which is passed to **avs_open**, and which can be used to alter the customary default options for creating and managing indexes. See **avs_parameters** structure for a description of this block. The first time **avs_open** is called for a new or empty directory, the parameters then in effect are used to create and initialize the index structure. The index is empty at this point, but its parameters are fixed.
5. For each document to be added:
   - Initialize the document and assign its document identifier, using avs_newdoc or avs_startdoc.
   - Add the words, fields, and values belonging to the document, using the avs_addword, avs_adddate, avs_addliteral, avs_addvalue, and avs_addfield procedures.
   - Assign a date to the document, using avs_setdocdate or avs_setdocdatetime (for ranking by date).
   - Assign an opaque data structure to meet any needs of your search and display logic, using avs_setdocdata -- this structure is retrievable from a search result, along with the document's unique document identifier and date.
   - Assign any additional ranking values, using avs_setrankval
   - Finalize the document, using avs_enddoc if initialized with **avs_startdoc**
6. Every so many documents (every half million words or so), and when the last document has been processed, make the actual update to the index, by using avs_makestable.

After each call to **avs_makestable**, call **avs_compactionneeded** and if indicated, do a loop of **avs_compact_minor** calls to tune the index structure.

7. Close the index with **avs_close**.
8. Release any previously defined value types with **avs_release_valtypes**.

An index that is open in *rw* mode is also available for performing searches. You can perform a search at any time, in the same thread as the updates to the index or in other threads. You can even open the index in *r* mode in another process, perform a simple search, and then close the index again.

Differences Between *avs_newdoc* and *avs_startdoc*

The procedures **avs_newdoc** and the **avs_startdoc**/**avs_enddoc** pair are used to enter documents into the index but have different structures. The **avs_newdoc** procedure uses the callback function and implicitly calls a filter to process the data being entered into the index.

For languages that cannot use the callback mechanism (for example, Visual Basic and Java), **avs_startdoc** and **avs_enddoc** procedures handle entering the document data with an explicit call to the filter function that processes the data.

The following code fragments have the same results

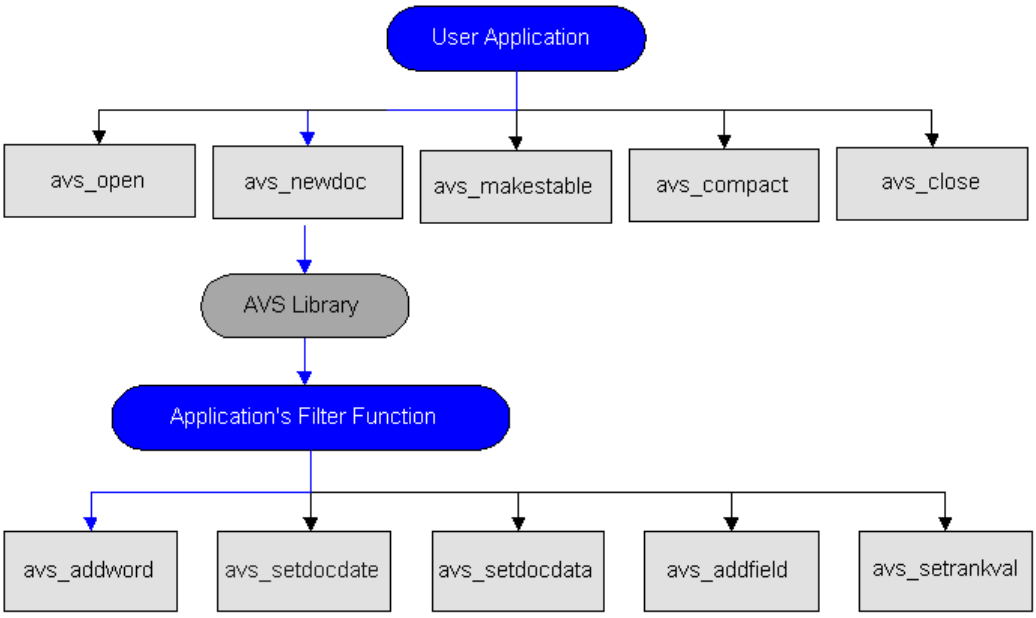
```
1. avs_newdoc(idx, datapath,filter, docid, flags,&numWords);

2. avs_startdoc(idx, docid, flags, &startloc);
filter(idx, datapath, startloc, &numWords);
   avs_enddoc (idx);
```

Adding New Documents Efficiently

You can add new documents, update existing documents, and delete old documents in your index at any time. You will find it is more efficient to do several additions and deletions at once, rather than doing them one at a time. You can still query your existing index while the additions and deletions take place.

The following diagram illustrates the interaction of your application code and the AltaVista Search library during the addition of a new document to the index using the new **avs_newdoc** procedure.



From your application, you make the appropriate C API calls to open the index you are populating. Use the filters to process or convert the documents you are adding to your index. Your application calls **avs_makestable** to write the index to disk, and closes the index. To access the newly-built index, use the query interface you have created and start querying the index.

Creating a Filter Procedure

The **avs_newdoc** procedure defines a block of text as a document and establishes an identifier with which the document can be found in the index. The **avs_newdoc** procedure also defines a filter, which is written or supplied by you, the application programmer. The filter function is called each time your application calls avs_newdoc to create a separate document in the index. If the **avs_startdoc** and **avs_enddoc** procedures are used instead of **avs_newdoc**, the filter procedure is called between these two calls.

The filter does the bulk of the work of preparing the document to be indexed. It is at the filter stage where any necessary document type conversion takes place. The document conversion libraries may be used at this point to convert documents to text before being added to the index. Alternatively, the application could do a bulk conversion of the documents beforehand. The filter function is called using the following required arguments:

```
IN  avshdl_t idx     (index handle)
IN  void *pFname     (information sufficient for the filter to access
the document contents)
IN unsigned long startloc  (starting location for adding words)
OUT unsigned long *pNumWords        (number of words added to the
index)
```

Once the filter is finished processing a block of text, it can pass the text (in the form of a line, a paragraph, or even the entire document), to the avs_addword procedure. The **avs_addword** procedure parses the text into words and adds those words to the index. It interprets as a word any sequence of letters or digits that is surrounded by spaces or other non-alphanumeric characters. When it adds a word to the index, the **avs_addword** procedure preserves the case of

the word as it appears in the document. If the word contains any uppercase letters, the software also indexes a lowercase version of the word, to support case-insensitive searching.

In addition to preparing the document so that each word in it can be indexed by **avs_addword**, you can use these calls in the filter to also do the following:

Use...	To...
avs_setdocdate	Set a date for the document
avs_setdocdata	Specify a data string to be returned as a search result
avs_setdocdatetime	Set a date and time for the document.
avs_addfield	Identify certain words to be indexed as fields.
avs_addliteral	Add a single word exactly as entered to a document index.
avs_adddate	Index the supplied date at the specified location.
avs_addvalue	Index the supplied value at the specified location.
avs_setrankval	Add a numeric value to a document index that can be used for custom ranking.

For example, if you are indexing mail messages and want users to be able to search based on the subject line of a message, you might identify the subject text of each document as the "Subject:" field and use the **avs_addfield** procedure to index it as such.

Examples of Indexing Techniques

The next several sections discuss some indexing techniques that could be useful in building the kind of index you need.

Defining Searchable Numeric Values

With the Developer's Kit, you can add your own searchable values.

To add your own value types for searching and ordering search results, follow these steps in structuring your application:

1. Use **avs_define_valtype** procedure to define your value type. For example, suppose you want to define a new value type called *lines* to count the number of lines per document. You must supply the name (lines) and the lowest and highest possible values. The following code example defines the value type for searching, in this case, the number of lines per document.

```
 error = avs_define_valtype ( "lines", 0, 10000, NULL,
&linesvaltype);
if (error != AVS_OK) {
printf ("avs_define_valtype returned %s\n", avs_errmsg(error));
return 1;
}
```

   Note: You should call **avs_define_valtype** only in your main thread when no indexes are open.

2. To enter a searchable value, call the **avs_addvalue** procedure. The following code examples adds a searchable value (number of lines). You can use *lines* in a Boolean query to constrain the results, for example, [lines:50-200].

```
error =  avs_addvalue (idx, linesvaltype, lineo, startloc);
if (error != AVS_OK) {
printf ("avs_addvalue returned: %s\n" avs_errmsg(error));
}
```

3. Call the **avs_release_valtypes** procedure after the last call to **avs_close** to release the values.

If the natural way to specify a searchable value is not an integer, you may supply a function that can convert the search string into an integer value in the right range. The function is then passed to **avs_define_valtype**. For example, you may want to search or rank results by a part number which consists of an alphanumeric string. Your application code may contain the following: `avs_define_valtype ("partnum", 0, 1000, partnum2int(), &partnumvaltype)` where *partnum2int( )* is a function supplied by your application that converts an alphanumeric part number to an integer value.

Defining Your Own Ranking Values

With the Developers Kit extended ranking capability, you can add your own ranking values for ordering search results. Rank values include a type and the value.

To add your own value types for ordering search results, follow these steps in structuring your application:

1. Use **avs_define_valtype** procedure to define your value type. For example, suppose you want to define a new value type called *rlines* to order search results by the the number of lines per document. You must supply the name (rlines), the lowest and highest possible values. The following code example defines the value type for extended ranking of search results, in this case, the number of lines per document.

```
error = avs_define_valtype ( "rlines", 0, 10000, NULL,
&rlinesvaltype);
    if (error != AVS_OK) {
        printf ("avs_define_valtype returned %s\n",
avs_errmsg(error));
        return 1;
}
```

   When defining value types for ranking purposes, always specify a minimum value of zero. Values are normalized to zero when stored in the index. If you choose to use a minimum value other than zero, your application must normalize the values to zero before performing a search.

   Note: You should call **avs_define_valtype** only in your main thread when no indexes are open.

2. To assign a rank value for a document, use a call to the **avs_setrankval** procedure. The following example adds a ranking value (the number of lines):

```
error = avs_setrankval (idx, rlinesvaltype, lineno);
if (error != AVS_OK)  {
printf ("avs_setrankval returned: %s\n:, avs_errmsg(error));
}
```

3. Call the **avs_release_valtypes** procedure after the last call to **avs_close** to release the values.

If the natural way to specify the filter value is not an integer, you may supply a function that can convert the search string into an integer value in the right range. The function is then passed to **avs_define_valtype**. For example, you may want to rank results by a part number which consists of an alphanumeric string. Your application code may contain the following: `avs_define_valtype ("partnum", 0, 1000, partnum2int(), &partnumvaltype)` where *partnum2int( )* is a function supplied by your application that converts an alphanumeric part number to an integer value.

Defining Multiple Values

With the Developer's Kit, you can add multiple non-zero values to documents which can be used to filter search results.

To add your multiple value type for filtering search results, follow these steps in structuring your application:

1. Use **avs_define_valtype_multiple** procedure to define your value type. For example, suppose you want to define a new value type called *multi* to represent the number of words per line in a document. You must supply the name (multi), the lowest and highest possible values, and the maximum number of multiple filtering values. The following code example defines the value type for filtering search results on multiple values, in this case, the number of words per line.

   ```
   error = avs_define_valtype_multiple ( "multi", 0, 255, 255,
   &multivaltype);
      if (error != AVS_OK) {
      printf ("avs_define_valtype_multiple returned %s\n",
   avs_errmsg(error));
           return 1;
     }
   ```

   When defining value types for filtering purposes, the minimum value must be zero.

   Note: You should call **avs_define_valtype_multiple** only in your main thread when no indexes are open.

2. To assign a rank value for a document, use a call to the **avs_setrankval** procedure. The following example adds filter values for the number of words per line (numwords).

   ```
    if (numwords <256) numwordflags[numwords]++;
        for (i="0;" i><256' i++) {
          if (numwordflags[i]) {
          error="avs_setrankval" (idx, multivaltype, i);
          if (error !="AVS_OK)" {
          printf ("avs_setrankval returned: %s\n",
   avs_errmsg(error));
          }
        }
   }
   ```

3. Call the **avs_release_valtypes** procedure after the last call to **avs_close** to release the values.

If the natural way to specify the filter value is not an integer, you may supply a function that can convert the search string into an integer value in the right range. The function is then passed to **avs_define_valtype_multiple** . For example, you may want to filter search results by a part number which consists of an alphanumeric string. Your application code may contain the

following: `avs_define_valtype_multiple ("partnum", 0, 255, 255, partnum2int(), &partnumvaltype)` where *partnum2int( )* is a function supplied by your application that converts an alphanumeric part number to an integer value.

Unicode Support

The Developer's Kit provides the capability to index full **Unicode** texts using the UTF-8 encoding. Unicode is a multi-byte encoding framework that provides for 2**32 character positions, of which only about 2**16 are filled to date.

Unicode is aimed at multilingual environments and internationalization. The Developer's Kit has limited support for automatic parsing of texts into words for the common European character sets. The standard includes the following languages: Latin, Greek, Cyrillic, Armenian, Hebrew, Arabic, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayan, Thai, Lao, Georgian, Tibetan, Japanese kana, the complete set of modern Korean hangul, and a unified set of Chinese/Japanese/Korean (CJK) ideographs. The standard is extensible and is continually enhanced. Refer to http://www.unicode.org/unicode/standard for more information on the Unicode Standard.

You can still use the single-byte ISO Latin-1 code set as an alternative, and a single-byte *ASCII8* mode is also provided which passes 8-bit characters through unchanged to the index. The character set must be specified in the *avs_parameter* structure. LATIN1 is the factory default. In the *avs.h* file the allowable character sets are defined as:

- AVS_CHARSET_LATIN1 - 0
- AVS_CHARSET_UTF8 - 1
- AVS_CHARSET_ASCII8 - 2

Handling HTML and SGML Special Characters

The **avs_addword** procedure will optionally detect and handle HTML or SGML entity substitutions, for example, for accented characters as in *é*, so that ASCII HTML pages containing accented words can be indexed properly.

The **avs_addword** parsing options are defined with the **avs_setparseflags** procedure. To enable SGML parsing, the parse flag is set as follows:

```
#define AVS_PARSE_SGML 1
```

Indexing Documents with Dates

When you index your documents, you can set a date for each one through the **avs_setdocdate** or the **avs_setdocdatetime** procedures. Once the dates are in the index, you can use the dates or date ranges to limit your searches. The date is returned in the search results.

The Developer's Kit is capable of storing dates from 01/01/0100 through 12/31/2148.

You can limit your query with a date range added as an extra Boolean term. The format of the date range is **[dd/mm/yyyy-dd/mm/yyyy]**. If you omit the beginning date, your query will return everything in the index with a date before the end date. If you omit the end date, your query result will contain all documents with dates after the beginning date. If you want only the documents indexed on one date, use the same beginning and ending dates. The end dates are part of the range.

Multiple Dates Associated with a Document

The procedure **avs_adddate** allows the indexing of multiple dates at various locations within a document, for example, as a part of a field. A document can be found using any or all of the associated dates.

For example, suppose your application gets the time the document was last modified from the operating system and uses this value with **avs_setdocdate**. You can also associate another date with the document - the date it was indexed. This code snippet demonstrates the use of the **avs_adddate** and **avs_addfield** to create a field called "Indexed" containing the current date.

```
 time( &ltime );
    curtime = gmtime( &ltime );
    avs_adddate (idx, 1900+curtime->tm_year, 1+curtime->tm_mon,
curtime->tm_mday, startloc);
    avs_addfield (idx, "Indexed", startloc, startloc + 1);
    startloc += 1;
```

Documents with multiple dates will be returned in date range searches if any of the dates entered by **avs_adddate** or **avs_setdocdate/avs_setdocdatetime** are in a given document. This can be deceiving because only dates set by **avs_setdocdate** and **avs_setdocdatetime** are displayed as a result of calling **avs_search_getdate**.

Searching the Index

With the Developer's Kit, you can perform the following types of searches:

- simple
- advanced - using Boolean terms
- advanced - using the ranking mechanism
- advanced - using a combination of Boolean and ranking

Use the avs_search, avs_search_ex, or the avs_search_genrank procedure to search the index you have created.

Both Boolean and ranking searches use an *avs_options* data structure, in which you can specify the maximum number of documents to return, and an optional timer limit (for multi-threaded applications only). The options structure is defined in the *avs.h* header file and is initialized by a call to the avs_default_options procedure.

Understanding Relevance Ranking

A feature of AltaVista searching is the optional ranking of results based on their probable relevance to the search query.

The search engine ranks the results of a search based on a weight value assigned to each word in the query, and a resulting overall relevance rating of each document that meets the search criteria.

A document earns a relevance rating based on the number of words in the search query that it contains, and the weight value of each of those words. The document containing the most words with the highest weight value is considered most relevant. The closer the relevance rating is to a value of 1, the more likely it is that a document meets the search criteria.

A search result can also have a relevancy ranking of zero (0). In this case, all results have the same weight or are equally relevant. A relevancy ranking of zero can happen in the case where the application did not specify a ranking in the query.

The weight of a word is determined by the number of occurrences of that word in the entire index. A word that occurs less frequently in the index earns a higher weight, based on the assumption that it is more precise and specific than a word that occurs frequently.

For example, the word "programming" might occur many times in an index, whereas the word "COBOL" would probably occur less frequently. "COBOL" would be given a higher weight than "programming" in a search query containing both words, because a document containing only the word "COBOL" would be more likely to match the searcher's interest than a document containing only the word "programming." A document containing both "COBOL" and "programming" would earn the highest relevancy ranking.

Note: The position of the word in the document, and the frequency of occurrence of the word in a single document, have some bearing on the ranking of a document. The most significant factor in determining ranking is the combined weight of words in the search query. Also, the search engine considers only words without an operator preceding them when it does ranking. If operators precede all words in the search query, the results are returned in no particular order.

Simple Query Syntax

To perform a basic search, use the operators plus (+) and minus (-) to indicate words or phrases that are required or prohibited in the search results. For example, the following query expression requests documents that must contain the word *results* and can also contain the phrase *year end*:

```
"year end" +results
```

Boolean Query Syntax

For Boolean searches, use the logic operators AND, OR, NOT, NEAR, and WITHIN. For example, the following query requests that either of the words apple or *pear* appear in the same document with either of the words *tart* or *pie.*

```
(apple OR pear) AND (tart OR pie)
```

The following query requests that both the words *spreadsheet* and *training* appear in a document's *title*: field.

```
title:(spreadsheet AND training)
```

Rules for Query Processing

Both the ranking and Boolean search procedures follow the same basic rules for processing queries:

- Like the indexer, the search engine interprets a word as any string of letters and digits that is delineated by non-alphanumeric characters. Consequently, AltaVista Search ignores punctuation except to interpret it as a separator for words.
- A group of two or more words enclosed in double quotes indicates a phrase. Phrasing ensures that the search engine finds the words together, instead of looking for separate instances of each word individually.

- An asterisk (*), double asterisk (**), or question mark (?) following three or more characters indicates a wildcard; the search engine will find all words that match the specified pattern. For more information refer to the Searching with Wildcards
- Case sensitivity of a search is based on the case of each word in the query. A word in all lowercase letters results in a case-insensitive search, whereas if a word contains any uppercase letters, the software searches for an exact-case match.

Basic Steps to Search the Index

The AltaVista Search programming interface provides a way to examine the contents of an index once it has been created. Use the **avs_search** and the **avs_counts** procedures to search for the presence of a specified word or words.

1. Ensure that any application-specific value types used during indexing with **avs_define_valtype** or **avs_define_valtype_multiple** are defined.
2. Use the avs_open procedure to open the index in *r* mode (or *rw* if you are also updating the index at the same time).
3. For each search request:
    - Call avs_search (or avs_search_ex or avs_search_genrank).
    - Call avs_getsearchresults to retrieve each document that meets the search criteria.
    - Optionally call any of the following procedures to retrieve information about the results:
        - avs_search_getdatalen
        - avs_search_getdata
        - avs_search_getdata_copy
        - avs_search_getdate
        - avs_search_getdocid
        - avs_search_getrelevance
    - Use the returned search handle and other returned values to fetch any or all result data.
    - Use avs_search_close to end the procedure and free the resources allocated for the search.
4. For each counts request:
    - Call avs_count to initialize the counting process, specify a word or word prefix to search for, and obtain a handle for the count. To enumerate the entire index from a to z, specify a null value for the word (*pWordprefix*) argument.
    - Pass the count handle to avs_countnext, which retrieves the first index entry. Continue calling **avs_countnext** to find subsequent entries that match the search criteria, until the procedure returns NO_MORE_WORDS.

- After each call to avs_countnext, use avs_count_getcount to return the total number of times the word occurs in the index.
- Use avs_count_getword to retrieve the word associated with the current count.
- Finally, end the procedure with avs_count_close.

5. Close the index using **avs_close**.
6. Release any previously defined value types with **avs_release_valtypes**.

You can use the avs_count and related procedures as diagnostic tools to test for the presence of a specific word or word stem in the index, or to get a count of words or groups of words. You might use the count procedures to learn why users do not get the results they expect from a query, or to obtain a general idea of the makeup of your index.

How Results Are Ordered

The counts results are ordered lexicographically. The search results are ordered in one of several ways, depending on which search call is used, and on the parameters of that call.

- The **avs_search** and **avs_search_ex** procedures assign to each matching document a *score* based on how well that document matches the set of ranking terms provided in the search call. If no ranking terms were provided, the results are presented in the same order as they were added to the index.
- The **avs_search_genrank** procedure orders the results according to a named value within the document, or by the document date or time. Higher values move documents to the front of the sequence. This value name is supplied in the *pRankTerms* parameter and can be prefixed with a minus sign (-) to specify inverse ordering (lower values first). The predefined ranking term *#date* is the name assigned to the document's date and time as set by **avs_setdocdate** and **avs_setdocdatetime**.

Searching for Numeric Values

With the Developer's Kit you can extend the type syntax to index and subsequently search for documents based on specific numeric values or ranges. For more information about indexing and defining value types, see Defining Searchable Numeric Values.

After you have added numeric values to your index with the **avs_addvalue** procedure, you can use the **avs_search** procedure to define how to search for the Boolean Query expression and the numeric range expression. The *pBoolQuery* argument in the **avs_search** procedure is used to specify the range of the numeric value type. For example, use the following format for an application-defined value type like *lines*:

```
[lines:<min value> - <max value>]
```

Your C programs must provide a function to convert numeric values that are not entered in numeric format. For example, your user may want to search documents based on a range of currency values that contain non-numeric characters. Specify a function to convert these values to integers when defining the value type with the **avs_define_valtype** or **avs_define_valtype_multiple**.

With your application you can also control whether your results are returned in descending or ascending order. In order to return the results in ascending order, place the minus sign (-) in front of the value type.

Searching for a Field

The following query requests documents that must contain the field *Subject:reorganization* and must not contain the field *Date:07/07/97*. The documents can also contain the word *CEO* but are not required to.

```
CEO +Subject:reorganization -Date:07/07/97
```

Searching for Literal Entries

Once you have added the literal index entries with the **avs_addliteral** function, you can perform an advanced search to find the literal string. If the string you are looking for contains special characters (for example, the forward slash (/)), you can use curly braces ({}) in the query string as in the following example: {cnn/xyz}. All characters between the matching curly braces are treated as part of a word except the asterisk (*) which still works as a wildcard.

Searching with Wildcards

AltaVista Search Intranet Developer's Kit provides extended wildcard support in all kinds of searching. Wildcards are limited to the following characters:

Asterisk (*)	After 3 specified characters will search for matches in up to 5 trailing letters.
Question Mark (?)	After 3 specified characters will match exactly one more character.
Double Asterisk (**)	More flexible as it will search for matches for an unlimited number of trailing characters.

With the Developer's kit, you can set the minimum number of characters before the wildcard in the *avs_parameter* block when the default of 3 characters is not sufficient for your needs. You also have the ability use the wildcards interchangeably and more than once in the same search string, for example:

```
ser*ip?t*
```

This could possibly find the word serendipity.

You can also determine whether to limit to 50 the number of words found by the wildcard character search or allow all instances of the word stem in the index you are searching. In the *avs_parameter* block, set the *unlimited_wild_words* flag to 1 to avoid the 50 word limit.

Searching and Ranking with Dates

Documents with multiple dates will be returned in date range searches if any of the dates entered by **avs_adddate** or **avs_setdocdate/avs_setdocdatetime** are in a given document. This can be deceiving since only dates set by **avs_setdocdate** and **avs_setdocdatetime** are displayed as a result of calling **avs_search_getdate**.

Only dates set with **avs_setdocdate** or **avs_setdocdatetime** can be used to rank documents by date.

Ranking Search Results

With the Developer's Kit, you can extend the type syntax to index and subsequently order search results by the named ranking value. For more information about indexing and defining value types, see Defining Your Own Ranking Values.

After you have added extended types to your index by the **avs_setrankval** procedure, you can use the **avs_search_genrank** procedure to search for the Boolean query expression using the generic ranking expression. The *pRankTerms* argument in the **avs_search_genrank** procedure can be either a predefined term, for example, *#date*, or an application defined value type like *rlines*. The special ranking setup argument *pRankSetup* should always be set to NULL for this release.

For example, you could use **avs_define_valtype** procedure to define the number of lines in a document. Use **avs_search_genrank** to rank search results by number of lines per document.

With your application you can also control whether your results are returned in descending or ascending order. In order to return the results in ascending order, place a minus sign (-) in front of the value type.

Your C program must provide a function to convert ranking values that are not entered in numeric format. For example, your user may want to rank by part numbers that contain alphanumeric characters. Because ranking values are all integers, your function must be able to convert the query string into an integer value to return a search result.

Filtering Search Results

With the Developer's Kit you can extend the type syntax to index and filter search results. For more information about indexing and defining value types, see Defining Your Own Ranking Values and Defining Multiple Values.

After you have added extended rank values to your index with the **avs_setrankval** procedure, you can use the **avs_search_genrank** procedure to search using a Boolean query expression and filter the results. The *pRankTerms* argument in the **avs_search_genrank** procedure can be a filter expression containing an application-defined value type. For example, given the Boolean query `pizza NEAR "to go"` and the filter expression `multi?((1-3, 5)`, the index is searched for all documents containing *pizza*, near the words *to go*, and then returns only those documents whose extended value type *multi* is between 1-3 or 5.

Your C programs must provide a function to convert ranking values that are not entered in numeric format. For example, your user may want to search for a part number that contains alphanumeric characters. Because filter values are all integers, your function must be able to convert the query string into an integer value to return a search result.

Incremental Searching

Your application can call the **avs_getsearchversion** procedure to return the version of the index for the current search result. This version string can then be passed to a subsequent search operation to limit results to those documents added to the index since the previous search.

The *searchsince* option is available on the **avs_search_ex** and the **avs_search_genrank** procedures. This value can be passed to these procedures to limit the results of a subsequent search to data indexed since the last search.

Proximity Searching

AltaVista Search keeps track of the positional relationships between words as it indexes them. The advanced search capability provides support for Boolean searches, including AND, OR, NOT and NEAR (proximity) searches. This allows for phrase searching and proximity searching to be performed on indexed documents.

With the Developer's Kit you can use the WITHIN ## (where ## is the number of words) command to control the number of words apart the words in your query string can be. For example, if you want to find the word *Mary* within 5 words of *lamb*, use the Boolean query string:

```
"Mary WITHIN 5 lamb"
```

This query will bring a result for Mary and lamb when they are not more than 5 words apart instead of the default of 10 words apart. Using NEAR in your search is the same as using WITHIN 10.

Query Processing Timeout Support

A new option on the **avs_search** (timeout) and a new api call (**avs_timer**) allows multi-threaded applications to enforce maximum query processing times.

The **avs_timer** procedure is used by an application's timer thread to pass a current timer value from the application to the AltaVista search operation. In this way, search operations can be be limited in processing duration. If the application does not have a timer thread, no search timeouts will occur.

In *avs_options.timeout*, you can set the number of timer units allowed for that query. At the start of each search, AltaVista Search sets a timer limit equal to the current timer value plus the value of *avs_options.timeout*. It periodically checks the current timer value against the timer limit. When the current timer value is greater than the limit, the search process stops and returns the partial results accumulated so far.

Converting Documents for Indexing

The AltaVista Search Developer's Kit now provides a set of converter libraries and API calls that you can use to convert your documents to text or HTML before indexing. The document conversion technologies are those offered by Adobe Systems, Inc., Inso Corporation, and Compaq Computer Corporation. Most file types are supported. For a complete list of file types that can be converted, click here. The Developer's Kit provides the following components for your conversion needs:

Converter Component	Description
Converter Libraries	The C API now comes with a document converter library (avscvt26.dll on Windows NT, libavscvt26.a on UNIX). The *avscvt.h* file provides calls that inspect the file to determine the file type and then converts the document to either text or HTML. Note: Only PDF documents can be converted to HTML at this time.

Converter Component	Description
ActiveX Component for NT Systems	For Windows NT, there are two supplemental files used with the ActiveX component: 1. *avscvt26X.dll* is the component that provides an API to the ActiveX clients, for example, Visual Basic. 2. *register_avscom.bat* is the batch file used with registering the ActiveX component.
Programming Samples	The two sample programs that use the document conversion technologies: 1. *avs_sample.c* contains an example of using the new document converter API. Using this sample program, you can covert documents to text before adding their contents to the index. 2. *vb_sample.exe* is the Visual Basic program that uses the document converter ActiveX object. (NT only)
Document Conversion Runtime Directory	This directory contains the runtime libraries that must be placed in your path for the document converters to run.

Using the AltaVista Search Intranet Compatibility API

The AltaVista Search Intranet Compatibility API allows an application to:

1. Search an index created by the AltaVista Intranet product (V2.3), and retrieve the ***document data***. The document data contains the document's URL, title, abstract, language and character encoding set.
2. Add a document to an index created by the AltaVista Search Intranet product (V2.3), or create a new index with compatible document data.

Use compatibility libraries (avsi26_mt.dll on Windows NT or libavsi26_r.a on UNIX) in conjuntion with the procedures outlined in avs_compat.h.

When you open the index, be sure to initialize the *avs_parameters* structure as follows:

avs_parameters_t params = AVS_PARAMETERS_AVSI_COMPATIBILITY;

The initialization ensures that dates and other metadata are stored and retrieved in the same format as the AVSI index.

- Use **avsi_setdocdata()** instead of **avs_setdocdata()**.
- Use **avsi_getdocdata()** instead of **avs_getdata().**
- Use **avsi_url2docid()** to generate compatible document IDs from URLs (See `avsi_sample.c` for sample code).

Note:
The AVSI index files on UNIX are owned by *daemon*. If documents are added to the AVSI index through the Developer's Kit, make sure that the documents are still owned by daemon before restarting AVSI or it will have problems reading the index.

Restrictions:

You may share an index between an SDK application and AltaVista Search Intranet V2.3 for searching only. The AltaVista Intranet indexer must be shut down before updating the index with your SDK application.

3

Advanced Concepts and Techniques

Managing a Growing Index

Once indexing is in progress, there are several things you can do to manage the contents:

- Use avs_makestable to write the contents of the in-memory index to disk.
- Use avs_compact to merge and streamline existing index files on disk.
- Use avs_deletedocid to remove an obsolete document from the index.

One of the reasons AltaVista Search indexing is so fast is that newly-indexed information is stored in memory until your application explicitly writes the information to disk. The avs_makestable procedure writes the most recent index content to disk and integrates it with the existing index. As a rule of thumb, you should call this procedure after approximately a half million words are indexed. This action preserves the data and prevents the index from consuming too much memory. You should also call the **avs_makestable** procedure before closing the index if any documents have been added or deleted since the previous **avs_makestable** call.

Each time you call the **avs_makestable** procedure, the newly-added document information in memory is written to a new, set of files on disk. So after several makestable calls, the on-disk index will actually consist of several sets of files. You should periodically use the avs_compact procedure to merge the existing set of files into one. You might compact the index once a day, during periods of least frequent use (the index is still available for queries during compacting, but you cannot add, update, or delete documents until compacting is complete). When compacting the index would be detrimental to your system resources, call the **avs_compact_minor** procedure. This will cause compaction without recovering space from deleted index entries.

Occasionally, a document may become obsolete and you will need to delete it from the index. Use the avs_deletedocid procedure to remove the document from the index database. Pass the identifier that the document received when the **avs_newdoc** procedure created it. The document will be marked for deletion and at the next call to the **avs_makestable** procedure, it will be removed from the index.

If there are any documents in the pending document set (those documents added since the last call to avs_makestable), an error (AS_DOC_EXISTS) is returned and no documents are deleted.

Note that compacting the index is needed to actually free the space occupied by deleted documents.

Optimizing for Speed

The AltaVista Search index can operate in either *query* mode or *build* mode. Use query mode when the index is relatively static and query load is high. Use build mode when updates are frequent (and query performance does not have to be optimal).

The **avs_buildmode_ex** procedure provides more control over points on the build mode spectrum. It allows one of the dynamic index parameters to be adjusted to provide less than the maximum buildmode performance to avoid the worst case query performance hit.

The default index parameters, however, are usually adequate for all but the most demanding applications. It is best to just leave the parameters at their default values, stay in build mode for updates and querying, do a full compaction when it is convenient (low update and query load), and adjust only when necessary.

If update performance is not a problem, but query performance is, try setting a value, for example, 5 or 6, in the **avs_buildmode_ex** procedure (not **avs_buildmode**). This will cause your program to loop more frequently through the minor compaction procedure, costing some time, but keeping the index in a higher state of query-processing tune.

A minor compaction (using **avs_compact_minor**) is used during build mode to keep the index size and complexity from exceeding reasonable bounds. It does the least amount of work it can while accomplishing this critical task. A full compaction on the other hand tries to make the index *optimal* for future query processing and updates, and takes more time to do this. Naturally, an index is more *compact* after a full compaction than after a minor one.

Programming Models

With the Developer's Kit, you can use different programming models which directly effect the performance of your indexing and querying processes.

Using the Multi-Threaded Model

The high performance model for using the Developer's kit is to do everything within a single multi-threaded process. In this model, at most one thread is responsible for managing the index, and as many threads as desired can issue queries. All threads use fine-grain thread locks to keep the shared in-memory data structure stable. The index is opened only once by the main thread. This model is often set up as a server with users' requests coming in and being serviced through socket connections or some other mechanism.

Using the Multiple Process Model

You can also have a multiple process model: one process for handling updates and other management functions, and as many processes as desired doing queries. The update process should open the index in read/write (rw) mode. The other processes should open the index in read (r) mode. This model works well on the relatively short-lived query processes, because at a key point in the update (rw) process, it waits for all the processes in read mode to be closed before proceeding.

In either model, only the unique thread or process which is managing the index is allowed to issue **avs_querymode** or **avs_buildmode** calls. These calls actually change the on-disk format of the index and incorporate compaction operations (in the case of **avs_querymode**) to be effective.

The proper sequence for the update process is as follows:

9. Open the index in read/write mode.
10. Place the index in build mode.
11. Perform the whole set of updates.
12. Place the index in query mode.
13. Close the index.

The sequence for the query processes is as follows:

1. Open the index in read mode.
2. Issue one or more search requests and the accompanying **avs_search_getdata** calls.
3. Close the index.

To get the best performance, it is recommended that the update process be run during the time of low query activity because the combination of updating and performing queries at the same time could be detrimental to your performance.

Tuning Compaq Tru64 UNIX for Large Indexes

It is imperative for optimal indexing and searching speed to let the AltaVista Search Developer's Kit memory-map the index files for reading. The indexer performs a lookup for every document at indexing time to check for duplicate document identifiers. It also reads all the data during a compaction.

To avoid using excessive virtual memory on smaller machines, the kit limits the size of files that will be mapped.

You can modify the upper limit to the size of files that are memory-mapped. In your application, use the cache threshold entry in the *avs_parameters* structure to set your cache threshold to a large enough value to map your largest file. For example, suppose your largest file in the index directory is less than 500 MB, set the *cache_threshold* parameter as follows:

```
param.cache_threshold = (500000000)
```

Increasing Virtual Memory for Processes

In order to take advantage of the higher cache threshold, you must also increase the amount of virtual memory that processes are allowed to use. The following process settings are recommended for large index files:

```
proc: max-per-proc-address-space = 137438953472
proc: max-per-proc-data-size = 17179869184
proc: per-proc-address-space = 137438953472
proc: per-proc-data-size = 17129869184
```

This allows your processes to have a large virtual address space.

Modifying the *vm-mapentries* Attribute

The *vm-mapentries* attribute specifies the maximum number of memory-mapped files in a user process, and limits the number of memory-mapped files available to each process. Each map entry describes one unique disjoint portion of a virtual address space. The default value is 200.

You may want to increase the value of the *vm-mapentries* attribute for very-large memory systems to 20000. This will increase the limit on file mapping. However, this attribute affects all processes, and increasing its value will increase the demand for memory. If there are

potentially many files, the number of **mmap** entries allowed on your system must also be high. Set the *vm-mapentries* value as follows:

```
vm-mapentries = 2000
vm-maxvas = 1337438953472
```

Modifying the *ubc-maxpercent* Attribute

Indexing usually consumes a moderate amount of virtual memory and uses a large set of files. The virtual memory subsystem and the Unified Buffer Cache (UBC), which caches file system data, share the physical memory that is not wired by the kernel. The default value of the *ubc-maxpercent* attribute is 100 (percent).

You can determine whether you should increase or lower the value of *ubc-maxpercent* by looking at the amount of free space after the application has been in use for some time. Use the **vmstat** command to display information about virtual memory. You should aim for a few megabytes of free space and no page outs ever. The measurement should occur during the indexing process because indexing uses more RAM.

Too much memory allocated to the UBC may cause excessive paging and swapping, which may degrade performance. However, an insufficient amount of memory allocated to the UBC may cause excessive disk I/O operations. You can reduce the value of the *ubc-maxpercent* attribute but the value can only be determined by experimenting on your machine. If your system exhibits excessive paging and swapping, reduce the value in decrements of 10%. Do not decrease the value to the point at which file system performance is degraded.

If *ubc-maxpercent* is set to a low value, you may want to increase the value if your disks are busy with file system I/O but the system has a high free page count. You should attempt to keep in memory the working set of your processes, even if this means increasing the amount of UBC misses.

Note: Changing the *ubc-maxpercent* attribute value in `/etc/sysconfigtab` requires that you reboot your system. To avoid this step, you could use the **dbx** utility to modify the *ubc-maxpages* attribute, for example:

```
dbx -k /vmunix
p ubc_maxpages
assign ubc maxpages  = ubc-maxpages – 1000
```

The conversion factor between the *ubc-maxpercent* attribute and *ubc-maxpages* attribute is:

```
ubc-maxpages = total_phys_pages_in_machine * ubc-maxpercent/100
```

If you in crease the value of *ubc-maxpages*, the buffer cache will use the memory within 10's of seconds if the machine is active. If you decrease the value, it takes much longer (tens of minutes) for the change to take effect. Once you have established the right value, calculate the value of *ubc-maxpercent*, and modify `/etc/sysconfigtab` using the new value.

How the Public AltaVista Search Site Sets the Virtual Memory Attributes

The AltaVista Search site on the web has the following setting for its virtual memory attributes:

```
vm-mapentries = 1000
vm-maxvas = 1337438953472
ubc-maxpercent = 70
```

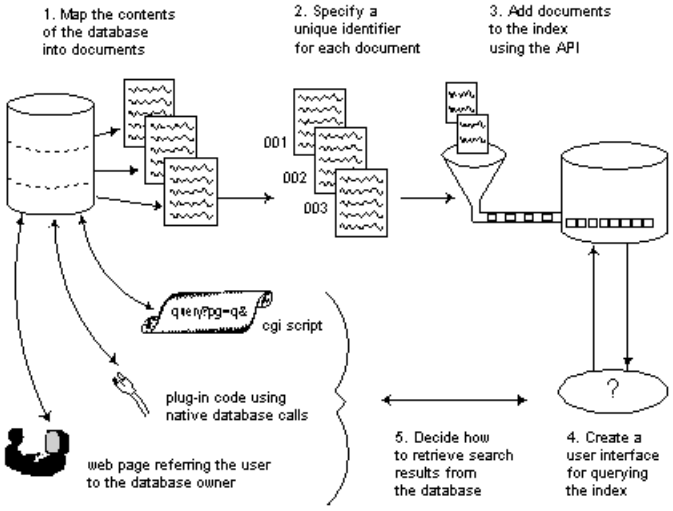
The following are settings for processes:

```
max-per-proc-address-space = 137438953472
max-per-proc-data-size = 17179869184
per-proc-address-space = 137438953472
per-proc-data-size = 17179869184
max-proc-per-user = 256
max-threads-per-user = 2048
```

Typically these machines are larger than average: 8-processor, 6-8 GB.

Indexing with Database Applications

With the Developer's Kit, the application developer has total control over what is indexed, what is retrieved from a search, and how to interpret the search results before showing them to the user. Assuming the application has access to the database, the application can simply store the retrieval identifier or the primary key with the index data and then retrieve and compute the appropriate display from the database. It is even possible to store the whole row as the retrievable index data and avoid query-time access to the database if your index is small enough and does not require frequent reindexing.



The following steps describe the process you use to create an index containing the contents of your database:

1. Determine the collection of data from the database that constitutes a document statement, for example, tables, records, or reports.
2. For each document determine the metadata that uniquely associates the document with the data source from step 1 (a URL, an SQL instruction, and so forth).
3. Using the Developer's Kit API calls, add words to the index.
4. Define a user interface to query the index.
5. Use the metadata from the documents returned by the query to retrieve the data source and display the results. For an example of a database implementation, see the *avs_sampledb.cpp* in the *dao_sample* directory.

Customizing Your Index with the Developer's Kit

In a new parameter block (*avs_parameters*) passed to the **avs_open** procedure, you now have control over the following:

- **Index scaling** parameters with *ntiers* and *nbuckets* parameters.

  At index creation, you can set the exact number of buckets and the maximum number of tiers to be used. Buckets spread the index entries across multiple files. Tiers are incremented by one every time **avs_makestable** is called. They are decreased by **avs_compact**. If these parameters are set to zero (0), the default values used are:

Platform	Tiers	Buckets
Windows NT and Solaris	6	4
Compaq Tru64 UNIX	12	12

  The tiers are tunable for special purposes within a nominal range from 4-40. The index uses fewer resources when these numbers are smaller, both in terms of the number of files used and the number of files the index has open at once. All of which could effect memory usage and other performance metrics.

  The maximum number of tiers are not necessarily the active tiers. The number of active tiers depend on what is occurring with the index, for example, **avs_makestable** adds a tier (unless the maximum setting would be exceeded). A full compact reduces a tier and, therefore, gives the best subsequent query performance, and at the same time recovers the index space of the deleted documents.

  The tier value set at indexing time will be used by the index as the maximum number of tiers throughout the life of the index. After the index has been built, do a full compaction of the index. For small updates to the index, use **avs_buildmode_ex** with a smaller tiers value. Call **avs_makestable** and **avs_compact_minor as needed**

  Periodically, perform a full compaction at a time when system resources are not under a heavy load. Full compactions take longer on a larger index. Do not use avs_querymode after every update - just use avs_buildmode_ex and avs_compact_minor. The only effect of avs_querymode is to do a full compaction and set the tier limit to a smaller number.

  Memory mapping:

  The *cache_threshold* parameter sets the maximum size in bytes of an index file that can be memory-mapped. The default is 500,000. If you have a large index and lots of virtual memory, you can set this number to a larger value to get better performance for the index.

- **Index Format:**

  The *indexformat* parameter allows you to force the on-disk format to a specific version of the index. The default value is the 'latest' index format, which is currently version 2. Version 1 indexes were slightly smaller, but also slightly slower to search. With version 2, the maximum allowable size for any index file on NT 4.0 and SUN platforms is 512MB; with version 1 the size limitation was 2 GB. Normally, the default index format should be sufficient for most indexing needs.

- **Wildcard grammar rules:**

  You can change the minimum number of characters required in a search word before a wild card character using the *chars_before_wildcard* parameter. The default is 3.

- **Unlimited wild word searching**

  The normal behavior of the wild card search expansion is that each wild-carded term will match a maximum of 50 words. If there are more than 50 words that match, the 50 most frequent words in the index will be used.

  To disable this behavior, set *unlimited_wild_words* parameter to 1.

- Ranking word maximum frequency:

  The *ignore_thresh* parameter is expressed in one hundredths of a percent, for example, `1000 = 10%` Any word that occurs in the index more frequently than this percentage is not counted for ranking purposes. The word is still counted for Boolean ranking.

  This is a performance optimization. If this value is set to be smaller than the default (1000), ranked searches will run faster but the ranking is less precise. If the value is set higher than the default, the ranked search is slower, but the ranking is more precise. The range for this parameter is 1- 1000.

- Optional indexing features:

  If you have no need for date ranking, date-range searching, or search-since features, they can be disabled in the *avs_parameter* block options. All these features are enabled by default. By selectively disabling some of these features, you can achieve a smaller index size. To disable, set one or more of the following to 0:

  - AVS_OPTION_SEARCHSINCE
  - AVS_OPTIONS_RANKBYDATE
  - AVS_OPTION_SEARCHBYDATE

- Assumed character set:

  The index supports three possible character sets. You can set the character set to one of the following:

  - ISO LATIN1
  - ASCII8
  - UTF8

  The default character set is ISO LATIN1. Mixing character sets in an index is not supported.

Initializing the *avs_parameter* Structure In the C API

To declare and initialize the *avs_parameters* structure to the default values, use the following statement in your application:

```
avs_parameters_t myparms = AVS_PARAMETERS_INIT;
```

Use `avs_parameters_avsi_compatibility` to initialize the `avs_parameters__t` structure when opening an AltaVista Search Intranet Index.

Modifying the Parameters in the C API

This example modifies each of the possible parameters by setting each one to its (already initialized) default value. Change to different values if needed.

```
myparms.ignored_thresh = 1000;
myparms.chars_before_wildcard = 3;
myparms.unlimited_wild_words = 0;
myparms.cache_threshold = 500000L;
myparms.indexformat = 0;"
myparms.options =
AVS_OPTION_SEARCHSINCE |
AVS_OPTION_RANKBYDATE |
AVS_OPTION_SEARCHBYDATE;
myparms.charset = AVS_CHARSET_LATIN1;
myparms.ntiers = 0;
myparms.nbuckets = 0;
```

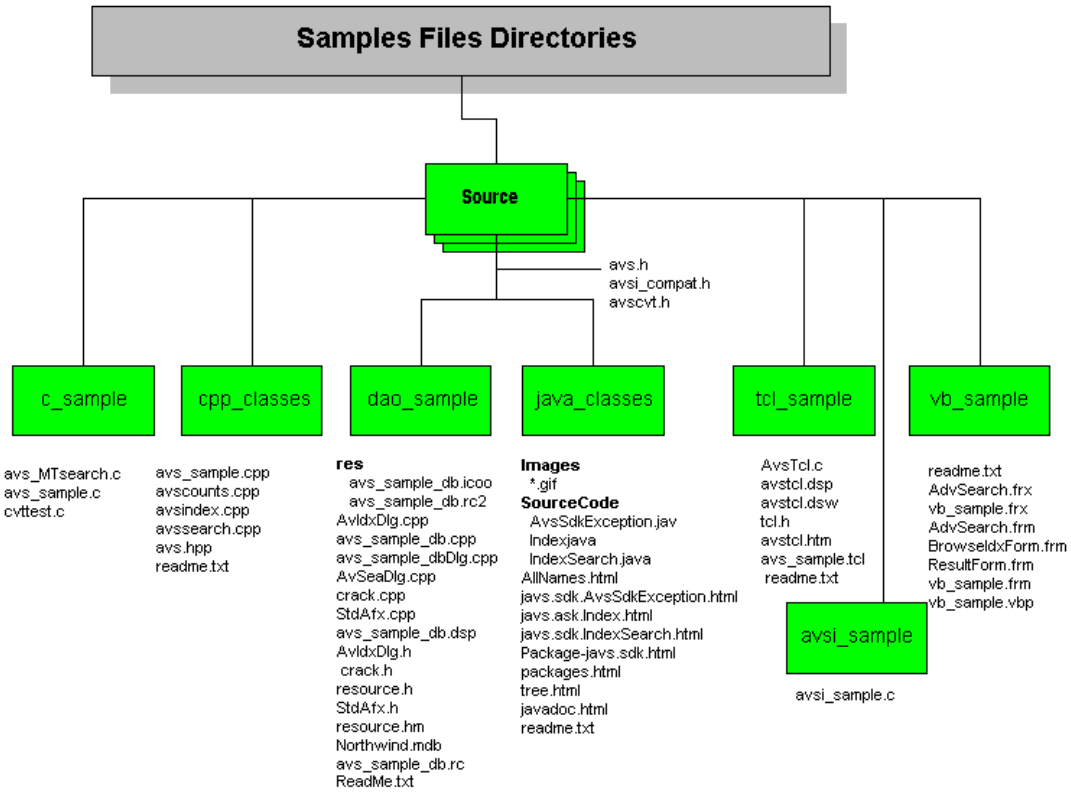
Opening the Index

To open an index using the initialized and modified avs_parameters, use the following:

```
error = avs_open ( &myparms, indexpath, mode, &handle );
```

4
Using the Sample Programs

This section describes the several sample programs provided with the AltaVista Search Developer's Kit Version 2.6. The following figure displays the directory structure under the source directory:



- The *c_sample* directory contains a sample C program (*avs_sample*.c) that uses most of the available procedures to build a simple text index and uses command-line querying. It also includes a multi-threaded search command contained in a separate module (*avs_MTsearch.c*). The *cvttest.c* module lets you test the conversion of a file to text.
- The *cpp_classes* directory contains an example program (*avs_sample.cpp*) that behaves similarly to the *avs_sample.c* program using one possible implementation of a C++ API. This API is used in the *dao_sample* as well.

- The *dao_sample* directory contains a database example that is specific to Windows NT and uses Microsoft Foundation Classes (MFC) and Data Access Objects (DAO) to build and search an index created from a Microsoft Access database.
- The *java_classes* directory contains sample Java classes to present an Object Oriented interface layered on top of the C API.
- The *tcl_sample* directory contains a sample Tcl application (*avs_sample.tcl*) similar to the *avs_sample.c* program.
- The *vb_sample* contains a sample Visual Basic application (including a working executable) that demonstrates the use of the Developer's Kit ActiveX control for Windows NT.
- The avsi_sample directory contains a sample program that demonstrates how to use the AltaVista Search compatibility API with the AltaVista Search Intranet V2.3 product.

Only the C Language and Visual Basic APIs are supported for this release. The other sample programs are used to demonstrate an implementation in the respective languages (Tcl, C++, or JDirect).

Understanding the C Sample Program

This section is a quick summary of the *avs_sample.c* program, organized by typical tasks. For information on compiling the programs, and using the command line tasks, refer to the section Compiling and Linking the C Sample Program.

What the Sample Program Does

If you learn best by example, you can use this program as a starting point for coding your own application. You can find the sample files in the directory in which you installed the developer's kit. The sample programs are compiled for you and the executable files are located in the the following directories:

- unix/aix
- unix/dunix
- unix/linux_ix86
- unix/solaris
- win32/alpha
- win32/ix86

Try some of the tasks listed below.

- Create an index from a set of text files.
- Search the index you have created.
- Perform a Boolean search using ranking terms.
- Perform a multi-threaded search.
- Count the word occurrences in the index.
- Delete a file from the index.
- Compact the index.

Creating an Index

Suppose you want to create an index containing all the words in a given set of text files. For the UNIX platforms, the files you want to include in the index could be in a separate directory, so they can be read from standard input. To read the files from standard input and pipe them to your application, use the following command line:

```
ls *.txt | avs_sample -c newdoc -i <indexdir>
```

Another way to specify files to be added to the index is to create a text file (`<filelist>`) containing the list of files (using the absolute path) to be indexed, one per line. Specify this text file as follows:

```
avs_sample -c newdoc -i <indexdir> -f <filelist>
```

where *newdoc* is the command and *indexdir* is the path to the index directory (the index directory must already exist). As a result, all the words in the text files are added to the index. One document is created for each text file. Each file is identified in the index by the name of the text file.

On Microsoft Windows NT, from the MS-DOS level, you can create your index in the same way as long as the files you want to include are in the same directory as your sample program and DLL files. Or you can put the location of the executable and DLL files in your path. To read the files from your index directory and pipe them to your application, use the following command line:

```
> dir/b *.txt | avs_sample -c newdoc -i <indexdir>
```

Searching an Index

To perform a simple search of the index, use the following command line:

```
avs_sample -c search -i <indexdir> -q "<simple query expression>"
```

where *indexdir* is the pathname to the index directory, and <simple query expression> is the query expression which may contain "+" and "-" filter expressions. For example, use the form

```
avs_sample -c search -i testdir -q "pizza \"deep dish\" +Chicago"
```

to search your index for Chicago, deep dish pizza. You can perform an advanced search by using *-b* instead of *-q* in your command line.

Performing a Boolean Search with Ranking Terms

Ranking words determine the order of listings in the results display. When you do not specify any ranking words, AltaVista Search returns the results in no particular order. Ranking is a very important way to ensure that the documents of most interest to you appear at the top of the results list.

To rank matches, enter terms in the Ranking argument; otherwise, the results will appear in no particular order. You could enter words that are part of your query or enter new words as an additional way to refine your search. For example, you could further narrow a search for COBOL AND programming by entering advanced and experienced in the Ranking argument.

Note: If you are interested only in a count of the number of documents that match your query, you may not want to use ranking.

To perform a boolean search with ranking terms, use the following syntax:

```
avs_sample -c search -i <index path> -b <boolean query> -q
<ranking terms>
```

where *index path* is the full path name of the index directory, *boolean query* is the query expression using the Boolean form (AND, OR, NOT, NEAR, WITHIN), and *ranking terms* is the term or terms that effect the ranking of results returned by the search. Documents that have the most frequent occurrence of the specified ranking term or terms will be at the top of the search results. For example:

```
avs_sample -c search -i indexdir -b "vegetable AND (NOT broccoli)"
-q "carrot"
```

Documents with the most frequent occurrences of the term "carrot" will be at the top of the search results.

Restricting Advanced Searches by Date

You can restrict an Advanced Search to find only documents last modified during a specific time frame. The date of the document is set by the application when the document is added to the index.

When entering To and From dates, use the format *dd/mmm/yy*, where *dd* is the day of the month, *mmm* is the name of the month, and *yy* is the last two digits of the year. Be sure to use the name of the month instead of a number; this eliminates ambiguity between date formats in different countries. For example, use *09/jan/96*.

If you omit the year when entering a date, the AltaVista assumes that the date is in the current year. If you omit both the year and the month and specify only numbers for days, the search assumes the current month and year. For example, entering a *From date* of 09/jan indicates that you want documents dated no earlier than January 9 of the current year. Entering a *From date* of 09 indicates that you want documents dated no earlier than the ninth day of the current month in the current year. To perform a boolean search with date ranges, use the following syntax:

```
avs_sample -c search -i <index path> -b <boolean query with date
ranges>
```

where *index path* is the full path name of the index directory, *boolean query with date ranges* is the Boolean query expression including the date ranges which restrict the dates and determine the results returned by the search. For example:

```
avs_sample -c search -i indexdir -b "vegetable AND (NOT broccoli)
[01/jun/97 - 31-jul-97]"
```

Only documents that fall within the date ranges are returned by the search results.

The date ranges are enclosed with square brackets ([ ]). You can use the following formats for date ranges:

- dd/mmm/yy - dd/mmm/yy
- dd/mmm/yyyy - dd/mmm/yyyy
- mm/dd/yy - mm/dd/yy
- mm/dd/yyyy - mm/dd/yyyy
- dd/mmm/yy - (for documents modified on or after the specified date)
- - dd/mm/yy (for documents modified on or before the specified date)

Performing a Multi-Threaded Search

To perform a multi-threaded search, use the following command line:

```
avs_sample -c mtsearch -i <indexdir> -q <queryfile>
```

Where *queryfile* is a text file containing a set of query strings which are processed by one of the threads.

Counting Word Occurrences in Your Index

To count how many times a word occurs in the index, use the form:

```
avs_sample -c counts -i <indexdir> -q <word>
```

Deleting a Document from the Index

To delete a document from the index, use the following command:

```
avs_sample_cpp -i index -c delete -d <docid>
```

Compacting an Index

To compact an index after you have made numerous additions and deletions, use the following command:

```
avs_sample -c compact -i <indexdir>
```

Compiling and Linking the C Sample Program

The AltaVista Search Developer's Kit gives you the option of compiling your application to have either single-threaded or multi-threaded capabilities. To compile and link your program on any of the supported platforms, you must have included the appropriate header file in your source code. The following table lists the additional files you need to compile, link and execute your program:

Operating System	Files in local directory
The Unix platforms	avs.h libavs26.a or libavs26_r.a
Microsoft Windows NT	avs26.dll or avs26_MT.dll avs.h avs26.lib or avs26_MT.lib

The following table lists the commands and switches used in running the program:

Switches	Description
-b	<boolean query string>
-c	<command> One of newdoc, search, genrank, counts, compact, delete, mtsearch (with multi-threaded build only)
-d	<docid>
-f	<filename> file contains list of files to index
-i	<index name>
-l	Turn off detailed logging of queries for mtsearch command
-q	<query string> use query string for rank only <query file> file with query string(s) for mtsearch command
+q	<query string> use query string for rank and selection
-s	<search since string>
-t	<number of threads> for mtsearch command (greater than 0)"

Compiling on a Compaq Tru64 UNIX System

The following command compiles the file *avs_sample.c* and creates an executable called avs_sample with the single threaded library:

```
cc -o avs_sample avs_sample.c libavs26.a -lm
```

or with the multi-threaded library:

```
cc -DPTHREADS -pthread -o avs_sample avs_sample.c avs_MTsearch.c
   libavs26_r.a -lm -lpthreads
```

Compiling on Microsoft Windows NT

The following command compiles the file *avs_sample.c* and creates an executable called avs_sample.exe with the single threaded library:

```
cl avs_sample.c /DWIN32 /D_NDEBUG /D_CONSOLE /ML avs26.lib
```

or with the multi-threaded library:

```
cl /Feavs_sample.exe avs_sample.c avs_MTsearch.c
/DWIN32 /D_NDEBUG /DPTHREADS /D_CONSOLE /MT avs26_MT.lib
```

Compiling on Solaris and Linux Systems

You can compile the file *avs_sample.c* and create an executable called avs_sample with a single-threaded library:

```
cc -o avs_sample avs_sample.c libavs26.a –lm
```

or with the multi-threaded library:

```
cc -DPTHREADS -o avs_sample avs_sample.c avs_MTsearch.c
libavs26_r.a -lpthreads –lm
```

Note: The indexing process opens many files. If the open file descriptor limit is too low, your program will abort on the Sun Solaris system. Set the descriptor limit to unlimited as follows:

```
   limit descriptors unlimited   (csh)
or
  ulimit -n unlimited   (sh)
```

Compiling on AIX Systems

You can compile the file *avs_sample.c* and create an executable called *avs_sample* with a single-threaded library:

```
cc -o avs_sample avs_sample.c libavs26.a –lc
```

or with the multi-threaded library:

```
cc_r -o avs_sample avs_sample.c avs_MTsearch.c
libavs26_r.a -lpthreads -lc_r
```

Compiling and Linking with Document Converters

By default, the document converters are not compiled into the C sample program. To use the converter API, compile the sample program according to the platform-specific instructions above with the following additions:

- Compile with the symbol `DOC_CONVERTERS` defined:

  `/DDOC_CONVERTERS` (Windows NT) or `-DDOC_CONVERTERS` (UNIX)

- Link with the library *avscvt26.dll* (Windows NT) or *libavscvt26.a, libsc_da.so,* and *libsc_ta.so* (Unix)

- Make sure that the document converter directory files are available at runtime. For Windows NT, add the directory to the PATH environment variable. For UNIX, set the environment variable LD_LIBRARY_PATH to this directory.

Note: The document conversion API is not available on the AIX or Linux/Intel platforms.

Using the Document Conversion Test Program

A new document conversion test program has been included with the product software. This program invokes the document converters to convert a specified file to text. The program, `cvttest.exe` or `cvttest` can be found in the platform specific directory. The source file `cvttest.c` can be found in the `source/c_sample` directory.

Understanding the Database Example

The sample database application (*dao_sample\avs_sample_db.cpp*) introduces the concepts and techniques associated with using the AltaVista Search Developer's Kit to index a Microsoft Access database. This example has the following characteristics:

- Runs on Microsoft Windows NT.
- Uses Microsoft's Visual C++ Version 5.0.
- Uses the sample Developer's Kit C++ API implementation (cpp_classes).
- Access to database through Data Access Objects (DAO).
- Sample Microsoft Jet Database created in Microsoft Access.

Applications using other data access methods, such as, Open Database Connectivity (ODBC) would involve similar design decisions with slightly different data exchange characteristics.

The sample database application demonstrates the following tasks:

- Creating an index from a database
- Searching the generated index
- Retrieving data from the database as a result of a search
- Deleting a record from the database and the index

Creating the Index

The application builds an index from the database creating one document for every record of a given database table. As each document is added to the index, a unique document identifier is generated to allow the original database record to be retrieved following a successful search.

For simplicity, this sample assumes one database per index. In addition, only the tables (versus queries, forms, reports) can be indexed since the tables represent the full compliment of data in the database. With these limitations, the document id is constructed from the name of the table and the primary key value of the record. Unlike record positional information, the primary key information in a table remains constant across sessions. Additional record specific information not subject to searches is included in the document data.

Searching the Index

Once the document id has been constructed, the fields of the record are indexed by adding them to the document. In order to allow field-specific searches, a field identifier is added along with the contents of the field in the index. A primitive user interface takes a user specified query string and searches the index. Search results are displayed in a list box to allow the user to select specific document from the index.

Retrieving Data from the Database

After the user selects one of the documents returned from a search, the corresponding database record can be displayed. This part of the application takes the record-specific information stored in the document id and document data to locate the appropriate record in the appropriate table in the database.

Deleting a Document

The sample database application also allows you to delete a selected document from the index and the corresponding record from the database provided the record does not have relationships with other records in the database. This process uses the same method of retrieving the database record from the document id and document data stored in the index.

Synchronizing the Index with the Database

To keep the index synchronized with the contents of the database, use one of the following methods:

1. If all fields in all database tables are kept in the index, keeping the index current is simply a matter of periodic reindexing. Reindexing the database will behave in the same manner as a special update procedure.
2. If only selected fields of selected tables are kept in the index, an update procedure would need to use information about the contents of the index for selective updates. Such information could be stored in a file upon index creation.

The sample database application and all files used to build and run it can be found in the *dao_sample* subdirectory of the AltaVista Search Developer's kit. See the *ReadMe.txt* files for an outline of the files included with this sample. See the source code comments for more details on the how the application works.

This sample database application can be run on a non-development system provided the following files are present:

- MSVCRT.DLL
- MFC42.DLL
- DAO and the Microsoft Jet Database Engine (in *dao\disk1\setup.exe*)

- OLE Automation (in *oadist.exe*) if requested during DAO setup.

These items are included on the kit in the *msvc* directory tree.

The DAO setup has been redistributed from the VC++ V5.0 CD and the OLE Automation setup can be downloaded from Microsoft.

See articles @167523 and @164529 from Microsoft Technical Support for more information

Sample Java Application

The Developer's Kit contains a sample JDirect implementation. The sample Java classes present an object oriented (OO) interface layered on top of the the C API. The Java sample directory contains documentation and a working sample.

Tcl Sample Application

You can use the Tcl programming language on multiple platforms to access the AltaVista Search Developer's Kit. The kit includes an adaptor module (*avstcl*) that implements the AltaVista Search API as a loadable Tcl extension. It has been tested on NT with Tcl Version 8.02 and using Microsoft Visual C++ 5.0.

To run on an NT x86 system:

1. Install Tcl using *tcl80p2.exe* if you do not already have it installed on your system.
2. Use the supplied *avstcl* MSVC project files to build the extension.
3. In the Tcl shell, a command like `load release/avstcl.dll avs` will load the extension (see *avs_sample.tcl*).

To run the application use the following command line:

```
tclsh80 avs_sample.tcl -i index -c search -q "word"
```

Visual Basic Sample Application

You can use Visual Basic on Windows NT platforms to access the AltaVista Search Developer's Kit. The kit includes an ActiveX control embodying the interface to AltaVista Search (*avs26X_mt.dll*). The control can be found in the ix86 or alpha directory after the installation.

In addition, this kit includes an ActiveX control for the document converters called *avscvt26X_mt.dll*.

For more information on this sample program, see the *Readme.txt* file included in the `win32\source\vb_sample` directory.

AltaVista Search Intranet Compatibility API Sample

You can now add or create an index compatible with the AltaVista Search Intranet V2.3 (AVSI) product. The Developer's Kit AVSI compatibility API lets your application read data from and write data to indexes created by AVSI.

The C API versions of the AVSI compatibility components are called `avsi26_mt.dll` and `libavsi26_r.a` for Windows NT and Unix, respectively. Please see the include file `avsi_compat.h` (source directory) and the sample code in `avsi_sample.c` (source/avsi_sample directory) for information on how to use the C API.

The AVSI index files on UNIX are owned by *daemon*. If documents are added to the AVSI index through the Developer's Kit, make sure that the documents are still owned by daemon before restarting AVSI or it will have problems reading the index.

Note: The AltaVista Search Intranet Compatibility API is not available on the AIX or Linux/Intel platforms.

5

C Programmer's Reference

This reference section provides a description of the C language procedures, data structures, and status codes provided by the AltaVista Search Developer's Kit.

Contents

avs_adddate

Indexes the supplied date in standard format at the indicated location.

C Synopsis

```
AVSAPI(int) avs_adddate (
  IN avs_idxHdl_t idx,/* Index handle (from avs_open) */
  IN int yr,/* date to add */
  IN int mo,/* date to add */
  IN int da,/* date to add */
  IN long startloc/* location */
);
```

Arguments

idx	The index handle.
yr	Integer that specifies the year.
mo	Integer that specifies the month.
da	Integer that specifies the day.
startloc	Location in the document for the date.

Description

The **avs_adddate** procedure indexes the supplied date in standard format at the indicated location. Dates added with **avs_adddate** can be retrieved by using range searches. Note that the date returned with a search result is the date supplied by **avs_setdocdate** not **avs_adddate**.

Return Values

AVS_OK or an error code.

See also

- avs_addfield
- avs_addword
- avs_setdocdate

avs_addfield

Marks a set of locations as belonging to a field.

C Synopsis

```
AVSAPI(int) avs_addfield (
  IN avs_idxHdl_t idx,       /* Index handle (from avs_open) */
  IN const char *pFname,     /* name of field (no spaces allowed) */
  IN long startloc,          /* first location of field */
  IN long endloc             /* first location AFTER field */
);
```

Arguments

idx	The index handle.
pFname	String that specifies the name of the field to be added to the index.
startloc	Location of the first word in the field.
endloc	Location of the first word that follows the field.

Description

The **avs_addfield** procedure marks a set of locations in a document as belonging to a field. The *startloc* and *endloc* arguments define the boundaries of the field. Application users can submit queries for specific contents of the named field, using the format *fieldname***:***value.*

Return Values

AVS_OK or an error code.

See also

- avs_addword

avs_addliteral

Adds a single word exactly as entered to a document index.

C Synopsis

```
AVSAPI(int) avs_addliteral (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN const char *pWord,        /* word to add */
  IN long loc/* location within document */
);
```

Arguments

idx	The index handle.
pWord	Single string that specifies the word to add.
loc	Location in the index of where to add the string to the index.

Description

The **avs_addliteral** function adds a single word without interpretation to a document index.This function does not scan the literal string in any way, but rather adds it to the index *as is*. Use with caution or not at all, as words added this way are possibly not searchable with the standard query procedures.

To perform a search when the literal string you are looking for contains special characters (for example, the forward slash (/)), you can use curly braces({}) in the Boolean (advanced) query string as in the following example: {cnn/xyz}. All characters between the matching curly braces are treated as a word, except the asterisk (*) which still works as a wildcard.

Note: Words starting with a non-alphanumeric character are reserved for internal use.

Return Values

This function returns AVS_OK or an error code.

avs_add_ms_callback

Adds a *makestable* callback function.

C Synopsis

```
AVSAPI (int) avs_add_ms_callback (
    IN avs_idxHdl_t pHdl,   /* ptr to index handle */
    IN int (*func) (avs_idxHdl_t, void *),  /* callback function */
    IN void *data           /* data to pass to callback function */
);
```

Arguments

pHdl	Pointer to the index handle.
(*func)	Callback function.
****data***	Data to pass to the callback function.

Description

The **avs_add_ms_callback** procedure is a synchronization point provided by AltaVista Search to the application during the processing of an **avs_makestable** call. The default AltaVista Search behaviour is to wait for any other search processes that might have the index open for a read operation to close the index before continuing; this is to avoid the potential for inconsistent search results in any of those other processes. This behaviour is not needed if the application is all in one process.

If the application provides its own **ms_callback** procedure through the **avs_add_ms_callback** call, AltaVista Search will call this procedure instead of performing its default synchronization processing. This call is made after the newly indexed data has been written to disk but before the new index version becomes available to any other threads.

Return Values

AVS_OK or an error code.

See also

- avs_makestable

avs_addvalue

Indexes the supplied value at the indicated location.

C Synopsis

```
AVSAPI(int) avs_addvalue (
   IN avs_idxHdl_t idx,/* Index handle (from avs_open) */
   IN const avs_valtype_t valtype,/* type and */
   IN unsigned long value,/* value to add */
   IN long loc/* location */
;
```

Arguments

idx	The index handle.
valtype	The value type.
value	The value to be indexed.
loc	The location in the document.

Description

The **avs_addvalue** procedure indexes the supplied value at the specified location in the index.

Return Values

AVS_OK or an error code.

See also

- avs_define_valtype

avs_addword

Adds words to the document index.

C Synopsis

```
AVSAPI(int) avs_addword (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN unsigned char *pWords,      /* words to add */
  IN long loc,/* location to add words */
  OUT long *pNumWords   /* number of words added */
```

Arguments

idx	The index handle.
pWords	Pointer to the words to add to the index.
loc	Location value to assign to the first word.
****pNumWords***	Number of words added.

Description

The **avs_addword** procedure adds words to the index. A word is defined as a contiguous string of alphanumerics, bounded by non-alphanumerics (like spaces and special characters), as defined in the ISO Latin-1 standard.

It should be called by a filter procedure, which prepares a block of text for indexing. The **avs_addword** procedure returns the number of words added to the index. Usually, the next call to **avs_addword** will be given a starting location which is **pNumWords* (the number of words added) higher than the last call.

Return Values

AVS_OK or an error code.

See also

- avs_adddate
- avs_addfield
- avs_addliteral
- avs_addvalue

avs_buildmode

Optimizes your index for building or adding documents.

C Synopsis

```
AVSAPI(int) avs_buildmode (
  IN avs_idxHdl_t idx  /* Index handle (from avs_open) */
);
```

Arguments

idx	The handle to the opened index.

Description

The **avs_buildmode** procedure optimizes the specified index for building or adding to the index. Querying during this state would degrade the response to the query. This procedure could be called from your application for those instances when you want to build an index and users would be unlikely to query the index. The new mode takes effect immediately. This state of the index will be retained for the next time the index is opened, if a call to **avs_makestable** or **avs_compact** is made.

Return Values

AVS_OK or an error code.

See also

- avs_compact
- avs_makestable
- avs_newdoc
- avs_querymode

avs_buildmode_ex

Optimizes your index for building or adding documents, and also sets the number of tiers.

C Synopsis

```
AVSAPI(int) avs_buildmode_ex (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN int ntiers         /* max tiers to use */
);
```

Arguments

idx	The handle to the opened index.
ntiers	The maximum number of tiers to use.

Description

The **avs_buildmode_ex** procedure optimizes the specified index for mixed indexing and searching the index by setting the maximum number of tiers to use. The tiers are tunable for special purposes within a nominal range from 4-500. Smaller values are appropriate for more searching while the larger values for more indexing.

The tiers parameter value determines the maximum number of sets of buckets to which the index can grow during operations that add, delete, or update the index. Each call to **avs_makestable** creates a new tier in the index, and calls to **avs_compact** or **avs_compact_minor** are then used to reduce the tiers again. When building a very large index, it is better to have a larger value as it reduces the number of compactions needed during building. However, query operations may take longer when more tiers are in use because there are more index files to examine for each index entry.

Buckets are the hash modulus for splitting the index by word. This parameter value determines the number of the index files across which index entries are spread (by hashing). If this number is increased, the number of index files is increased but the size of the individual files should be smaller

The maximum number of files used by the index is approximately 4*buckets*tiers. The index uses fewer resources when these numbers are smaller, both in terms of the number of files used and the number of files the index has open at once, which could effect memory usage and other performance metrics.

Return Values

AVS_OK or an error code.

See also

- avs_compact
- avs_makestable
- avs_newdoc
- avs_querymode

avs_close

Closes a specified index.

C Synopsis

```
AVSAPI(int) avs_close (
  IN avs_idxHdl_t idx     /* Index handle (from avs_open) */
);
```

Arguments

idx	The handle to the opened index.

Description

The **avs_close** procedure closes the specified index and releases all resources.

Return Values

AVS_OK or an error code.

See also

- avs_makestable
- avs_newdoc
- avs_open

avs_compact

Causes one level of compaction on the index.

C Synopsis

```
AVSAPI(int) avs_compact (
  IN avs_idxHdl_t idx, /* Index handle (from avs_open) */
  OUT int * bMore_p         /* TRUE iff more compaction indicated */
);
```

Arguments

idx	The index handle.
**** bMore_p***	Integer value returned to indicate whether more compaction is necessary (1 if yes, 0 if no).

Description

The **avs_compact** procedure causes one or more levels of compaction on the index. If the returned value of *bMore_p* is 0, the compaction is complete. If the returned value is 1, call the **avs_compact** procedure again to further compact the index.

You should compact the index periodically or after a series of updates is complete. Compacting the index improves subsequent query performance and frees the space used by documents that have been deleted. It is possible to submit search queries to the index (in other threads) while it is being compacted, but you cannot add, update, or delete information until compaction is complete.

Return Values

Value Returned	Description
0	The compacting of the index is complete.
1	Further compacting of the index is required.

See also

- avs_compactionneeded
- avs_compact_minor
- avs_makestable

avs_compactionneeded

Returns a non-zero value if the index needs compaction.

C Synopsis

```
AVSAPI(int) avs_compactionneeded (
  IN avs_idxHdl_t idx  /* Index handle (from avs_open) */
);
```

Arguments

idx	The index handle.

Description

The **avs_compactionneeded** returns non-zero value if the index needs compaction. When writing your application, it is reasonable to test if compaction is needed after each call to **avs_makestable**. If compaction is needed, your application should perform an **avs_compact_minor** loop until no more compaction is needed. If you defer the compaction too long, then eventually the makestable process will get infinitely slow.

Return Values

Returns 0 or a non-zero integer

See also

- avs_compact
- avs_makestable

avs_compact_minor

Causes compaction of the index with as little impact on system resources as possible.

C Synopsis

```
AVSAPI(int) avs_compact_minor (
  IN avs_idxHdl_t idx, /* Index handle (from avs_open) */
  OUT int * bMore_p         /* TRUE iff more compaction indicated */
);
```

Arguments

idx	The index handle.
**** bMore_p***	Integer value returned to indicate whether more compaction is necessary.

Description

The **avs_compact_minor** procedure causes one or more levels of compaction on the index but without recovering space from deleted index entries. Use this procedure when the effects of regular compaction would be detrimental to your system resources. If the returned value of *bMore_p* is 0, the compaction is complete. If the returned value is 1, call the **avs_compact_minor** procedure again to further compact the index.

You should compact the index periodically or after a series of updates is complete. Compacting the index improves subsequent query performance. It is possible to submit search queries to the index while it is being compacted, but you cannot add, update, or delete information until compaction is complete.

Return Values

Value Returned	Description
0	The compacting of the index is complete.
1	Further compacting of the index is required.

See also

- avs_compact
- avs_makestable

avs_convert_file2html

Converts a document to an HTML file.

C Synopsis

```
AVSAPI (int) avs_convert_file2html (
    IN  char *p_docpath,   /* pathname to document */
    OUT char *p_htmlpath,  /* pathname to a file to contain converted HTML
*/
    OUT int *pErr          /* converter error */
);
```

Arguments

p_docpath	Pathname to the document.
p_htmlpath	Pathname to a file to contain the converted HTML.
pErr	Converter error code.

Description

The **avs_convert_file2html** procedure converts a document to HTML. You must specify the pathname to the document to be converted as well as specify a pathname to the file which is to contain the converted text. At this time, the only file type that is able to be converted to HTML is the Adobe PDF file.

Return Values

AVS_OK, AVS_CVT_ERR or AVS_CVT_UNSUPTYPE.

See also

- avs_convert_file2text
- avs_convert_init

avs_convert_file2text

Convert a document to a text file.

C Synopsis

```
AVSAPI (int) avs_convert_file2text (
    IN  char *p_docpath,   /* pathname to document */
    OUT char *p_textpath,  /* pathname to a file to contain converted text
*/
    OUT int *pErr          /* converter error code */
);
```

Arguments

p_docpath	Pathname to the document.
p_textpath	Pathname to a file to contain the converted text.
pErr	Converter error code.

Description

The **avs_convert_file2text** procedure converts a document to text. You must specify the pathname to the document to be converted as well as specify a pathname to the file which is to contain the converted text. The document contents are analyzed before conversion to determine the document type.

Note: The file containing the converted text may contain no line ending characters.

The file types which are supported for conversion are listed in the following table: c_ref.htm - doctypes

File Type	Versions	Valid Extension(s)
Access	through 2.0	MDB
Adobe Acrobat Portable Document Format	2.1, 3.0	PDF
Adobe Illustrator	through 6.0	AI
Adobe PostScript, Encapsulated PostScript	Level 2	PS, EPS
Ami Draw	---	SDW
AMI/AMI Professional	through 3.1	SAM
AutoCAD Drawing Interchange Format	through 13	DXF
AutoCAD Native Drawing Format	12 and 13	DWG
AutoShade Rendering File Format	---	RND
batch file	---	BAT
bitmaps (including OS/2 DIB)	Windows	BMP, RLE, ICO, CUR, OS2
CCITT Group 3 Fax		FXS
Computer Graphics Metafile	ANSI, CALS, NIST,	CGM

File Type	Versions	Valid Extension(s)
	3.0	
Corel Clip Art Format	---	CMX
CorelDraw	through 7.0	CDR, CDW
DataEase	4.X	DBA, DBM
dBASE	through 5.0	DBF
dBXL	1.3	DBF
DCX (multi-page PCX)	---	DCX
DIGITAL WPSPlus	through 4.1	WPL, DX
DisplayWrite 2 & 3	all versions	TXT
DisplayWrite 4 & 5	through release 2.0	DOC
Enable	3.0, 4.0, 4.5	300, WPF, SSF, DBF
executables	---	EXE, DLL
First Choice	through 3.0	SS, FOL
FoxBase	2.1	DBF
FrameMaker (including vector and raster format)	through 5.0	FMV
Framework	3.0	FW3
Freelance	1 and 2 (Windows); 96 (Windows 95); 2.0 (OS/2)	PRZ, PRE
GEM Paint	---	IMG
Graphics Environment Manager Metafile	bitmap and vector	GEM
Graphics Interchange Format	---	GIF
GZIP		gz
Harvard Graphics	2.X and 3.X	CHT, CH3
Hewlett Packard Graphics Language	2.0	PGL
HyperText Markup Language	through 3.2	HTML, HTM, ASP, SHTML, NSF
IBM Final Form Text	all	FFT
IBM Graphics Data Format	1.0	GDF
IBM Picture Exchange Format	1.0	PIF
IBM Revisable Form Text	all	RFT
IBM Writing Assistant	1.01	IWA
JPEG	all	JPG, JPEG, JIFF
JustWrite	through 3.0	JW
Kodak Photo CD	1.0	PCD

File Type	Versions	Valid Extension(s)
Legacy	through 1.1	CHP
Lotus 1-2-3, 1-2-3 Charts	through 97 (DOS and Windows) through 2.0 (OS/2)	WKU, WK1, WK3, WK4, WK5, WK6
Lotus Symphony	1.0, 1.1 and 2.0	WR1
Macintosh standard raster	---	PICT1, PICT2
MacPaint	---	MAC
MacPict	---	PCT
Manuscript	through 2.0	DOC
MASS11	through 8.0	AA4, AA5, AA6, AA7, AA8
Micrografx Designer	3.1 (Windows) 6.0 (Windows 95)	DRW, DSF
Microsoft Windows Write	through 3.0	WRI
Microsoft Word	through 6.0 (DOS) 97 (Windows)	DOC
Microsoft WordPad	all	DOC
Microsoft Works	through 2.0 (DOS) through 4.0 (Windows)	WPS, WKS, WDB, WCM
Microsoft Excel	2.2 through 7.0 (Windows) 97 (Windows 95)	XLA, XLC, XLM, XLS, XLT, XLW
Microsoft PowerPoint	through 7.0 (Windows) 97 (Windows 95)	PPT
Microsoft Rich Text Format	through 2.0	RTF
Mosaic Twin	2.5	WKU
MultiMate	through 4.0	DOC, DOX, FNT, FNX
Navy DIF	all	DIF
Nota Bene	3.0	NB
Office Writer	4.0 through 6.0	OW4
Paradox	through 4.0 (DOS), through 1.0 (Windows)	DB, DB3
PC-File Letter	through 5.0 (File+ Letter, through 3.0)	LTR
PC Paintbrush	---	PCX
PFS:Professional Plan	1.0	TID
PFS:Write	A, B, and C	PFB
PIC (Lotus)	---	PIC
Portable Network Graphics Internet Format	1.0 (non-LZW Compression)	PNG

File Type	Versions	Valid Extension(s)
Professional Write	through 2.1; 1.0 (Plus)	PW1, PWP
Professional Write 2	---	22
Q&A	2.0 (DOS) 3.0 (Windows) through 2.0 (database)	QA, QW, DTF
QuattroPro	through 5.0 (DOS) through 7.0 (Windows)	WQ1, WB1, WB2
R:BASE	through 3.1; 1.0 (System V)	RBF
Reflex	2.0	R2D
Samna Word	through IV+	SAM
SmartWare II	1.02	DOC, DB, WS
Sprint	through 1.0	SPR
Standard Generalized Markup Language	---	SGML
Sun Raster File Format	---	SRS
SuperCalc 5	4.0	CAL
TAR		tar
text files	ASCII, ANSI	TXT
TGA (TARGA)	---	TGA
TIFF Format	through 6.0	TIF, TIFF
TIFF CCITT Group 3 & 4	---	TIF, TIFF
Total Word	1.2	TW
Viseo	----	VSD
Volkswriter 3 & 4	through 1.0	VW4
VP Planner 3D	1.0	WKS
Wang PC (IWP)	through 2.6	IWP
Windows Metafile	---	WMF
WordMARC	throthrough Composer Plus	WWMC
WWordPerfect	tthrough 7.0	WWPD, WPG, WPF, WP5
WWordStar	tthrough 7.0 (DOS); through 3.0 (WordStar 2000 for DOS); 1.0 (Windows)	WWS, WSD, WS2, WS4, WS6
XX-Windows Bitmap	---	XXBM
XX-Windows Pixmap	---	XXPM

File Type	Versions	Valid Extension(s)
XX-Windows Dump	---	XXWD
ZZIP file	PPKWARE through 2.04g	ZZIP

Return Values

- AVS_OK
- AVS_CVT_ERR
- AVS_DICTIONARY_ERR

See also

- avs_convert_file2html
- avs_convert_init

avs_convert_init

Initializes the document converters.

C Synopsis

```
AVSAPI (int) avs_convert_init (
    IN avs_convert_params_t *p_params  /* converter parameters */
```

Arguments

**p_params*	Converter parameters.

Description

The **avs_convert_init** procedure initializes the document converters. See *avs_convert_params* for more information.

Return Values

AVS_OK

See also

- avs_convert_file2html
- avs_convert_file2text

avs_count

Enumerates all the words beginning with a specified prefix in the index and, for each, how many times it occurs.

C Synopsis

```
   AVSAPI(int) avs_count (
  IN avs_idxHdl_t idx,           /* Index handle (from avs_open) */
  IN const char* pWordprefix,      /* first word to find (>=) (may be NULLSTRING) */
  OUT avs_countsHdl_t *pCountsHdl  /* ptr to counts handle
                                     (used with avs_countNext) */
);
```

Arguments

idx	The index handle.
pWordprefix	Pointer to a word or portion of a word. All words that begin with this character string are returned, one at a time, through the avs_countnext procedure.
pCountsHdl	Pointer to the *counts* handle.

Description

The **avs_count** procedure is used in conjunction with **avs_countnext** to enumerate index entries that match the specified word or prefix.

The count is not adjusted for deletions if they have occurred since the last time the index has been compacted.

To enumerate the entire contents of the index, use a null value for the *pWordprefix* argument. This procedure will return a count for all index entries including those that have been deleted.

Word counting is able to be used with wildcards. You could previously enumerate the index entries by counting the occurrences of *abc* in an index and the occurrences *abc* would be returned. Now you can enumerate the occurrences of *abc*d* in the index, and the number of occurrences of all the words that start with *abc*, followed by some character, and ending with *d* would be returned. The wildcard characters ?, *, and ** can be used to represent 1, 0 to 5, and 0 to unlimited characters, respectively.

Return Values

AVS_OK or an error code.

See also

- avs_countnext

avs_count_close

Ends a count request and frees allocated resources.

C Synopsis

```
AVSAPI(int) avs_count_close (
  IN avs_countsHdl_t countsHdl   /* counts handle */
);
```

Arguments

countsHdl	The counts handle.

Description

The **avs_count_close** procedure closes a count request. After all calls to **avs_countnext** are complete, call **avs_count_close** to release the resources allocated for the count.

Return Values

AVS_OK or an error code.

See also

- avs_count
- avs_countnext

avs_count_getcount

Retrieves the number of word occurrences corresponding to the most recent call to **avs_countnext**.

C Synopsis

```
AVSAPI(unsigned long) avs_count_getcount (
  IN avs_countsHdl_t countsHdl  /* counts result handle */
);
```

Arguments

countsHdl	The counts handle.

Description

The **avs_count_getcount** procedure retrieves the number of word occurrences corresponding to the most recent call to avs_countnext.

Return Values

The number of occurrences in the index of a word matching the specified word or prefix.

See also

- avs_count
- avs_count_getword
- avs_countnext

avs_countnext

Retrieves the first or next index entry matching the word or prefix specified in the **avs_count** procedure.

C Synopsis

```
AVSAPI(int) avs_countnext (
  IN avs_countsHdl_t countsHdl     /* counts handle (from avs_count) */
);
```

Arguments

countsHdl	The counts handle from avs_count.

Description

The **avs_countnext** procedure retrieves the next (or first) index entry that matches the prefix specified in **avs_count**. The procedures **avs_count_getword** and **avs_count_getcount** return the actual word and the number of times it occurs in the index, respectively. The procedure returns AVS_NOMORE_WORDS when there are no more matching words.

Return Values

AVS_OK or AVS_NOMORE_WORDS.

See also

- avs_count
- avs_count_getcount
- avs_count_getword

avs_count_getword

Retrieves the word corresponding to the most recent call to **avs_countnext**.

C Synopsis

```
AVSAPI(char *) avs_count_getword (
  IN avs_countsHdl_t countsHdl  /* counts result handle */
);
```

Arguments

countsHdl	The counts result handle.

Description

The **avs_count_getword** procedure retrieves a pointer to the word corresponding to the most recent call to avs_countnext.

Return Values

Pointer to the word corresponding to the most recent call to **avs_countnext** procedure. This pointer is only valid until the next call to **avs_countnext**.

See also

- avs_count
- avs_count_getcount
- avs_countnext

avs_cvterrmsg

Returns the pointer to the error message text corresponding to a document converter error code.

C Synopsis

```
   AVSAPI(char *)avs_cvterrmsg(int code);   /* copies error message text as
in comments above */
```

Arguments

char	The pointer to the error message text.

Description

The **avs_cvterrmsg** procedure returns the pointer to the error message text corresponding to a document converter error code. It is used to translate any non-zero status codes returned by any of the conversion procedures to a printable string (English).

Return Values

Pointer to the error message text corresponding to a document converter error code.

See also

- avs_cvterrmsg_copy

avs_cvterrmsg_copy

Copies the error message text to a buffer.

C Synopsis

```
AVSAPI(void)avs_cvterrmsg_copy(int e, char * buf, int bufsiz);
```

Arguments

buf	Buffer for the error message text.

Description

The **avs_cvterrmsg_copy** procedure copies the error message text to a buffer.

Return Values

None.

See also

- avs_cvterrmsg

avs_default_options

Initializes search options to default values.

C Synopsis

```
AVSAPI(void) avs_default_options (
  OUT avs_options_p_t pOptions  /* PTR to avs_options_t structure to be
initialized */
```

Arguments

pOptions	Pointer to the *avs_options_t* structure to be initialized.

Description

The **avs_default_options** procedure initializes search or query options to the default values. The default values for search options are:

- The maximum number of documents returned from a search is 32,000.
- The additional query option of AVS_OPT_FLAGS_RANK_TO_BOOL.
- ISO Latin 1 is the default character set.

Return Values

AVS_OK or an error code

See also

- avs_search

avs_define_valtype

Defines a new value type for adding searchable values, and ranking values.

C Synopsis

```
AVSAPI(int) avs_define_valtype (
IN const char * name,     /* up to 10 characters
IN unsigned long minval,  /* minimum value - zero recommended
IN unsigned long maxval,
IN unsigned long (*makeval) (char *),
OUT avs_valtype_t * valtype_p
);
```

Arguments

name	String that contains the name of the value type (By default, the maximum name length is 10 characters).
minval	Long with the minimum value of the value type, set to zero (0) when defining ranking types.
maxval	Long with the maximum value of the value type.
makeval	Procedure to convert string to value.
**** valtype_p***	Returned pointer to the new value type.

Description

The **avs_define_valtype** procedure lets you define your own value type, for example, the value type *lines* (to count the number of lines per document). A searchable value or a ranking value of this type can be added to a document. With the type name, you also must supply the lowest to the highest possible values. In your filter application, use a call to **avs_setrankval** procedure to set a rank value for each document in the index. Use a call to the **avs_addvalue** procedure to set a searchable numeric value.

If you expect a user's search terms to be something other than an integer, for example, a part number that may contain alpha-numeric characters, you must supply a *makeval* function that can convert the search string into an integer value.

The **avs_define_valtype** procedure is an application-wide procedure and, therefore, effects all the indexes that are open. You must call **avs_define_valtype** in your main thread before you open the index. Call the **avs_release_valtypes** procedure after the last call to **avs_close** to release the values.

Return Values

AVS_OK or an error code

See also

avs_addvalue	avs_define_valtype_multiple
avs_lookup_valtype	avs_release_valtypes
avs_search_genrank	avs_setrankval

avs_define_valtype_multiple

Defines a value type to be used for filtering on multiple non-zero values.

C Synopsis

```
   AVSAPI(int) avs_define_valtype_multiple (
     IN const char * name,     /* up to 10 characters
     IN unsigned long minval,  /* minimum value - must be zero
     IN unsigned long maxval,  /* maximum value
     IN int numvalues,         /* maximum no. of multiple filtering values
     IN unsigned long (*makeval) (char *), /* function to convert buffer to
numeric value
   OUT avs_valtype_t * valtype_p
);
```

Arguments

name	String that contains the name of the value type (the maximum name length is 10 characters).
minval	Long with the minimum value of the value type set to zero (0).
maxval	Long with the maximum value of the value type.
numvalues	Maximum number of multiple filtering values.
****makeval***	A function to convert the buffer to a numeric value.
**** valtype_p***	Returned pointer to the new value type.

Description

The **avs_define_valtype_multiple** procedure lets you define your own value type to add a set of values to a document. Subsequently, these values can be used to filter search results. With the type name, you also must supply the minimum to maximum range possible for values. The *numvalues* parameter determines the maximum number of multiple ranking values allowed for this valtype for each document.

In your filter application, use a call to **avs_setrankval** procedure to set each value for the document.

If you expect a user's search terms to be something other than an integer, for example, a part number that may contain alpha-numeric characters, you must supply a *makeval* function that can convert the search string into an integer value. Multiple value filters are designed to work with non-zero values only.

The **avs_define_valtype_multiple** procedure is an application-wide procedure and, therefore, effects all the indexes that are open. You must call **avs_define_valtype_multiple** in your main thread before you open the index. Call the **avs_release_valtypes** procedure after the last call to **avs_close** to release the values.

Return Values

AVS_OK or an error code

See also

- avs_define_valtype-
- avs_lookup_valtype

- avs_release_valtypes
- avs_search_genrank
- avs_setrankval

avs_deletedocid

Marks the specified document for deletion.

C Synopsis

```
AVSAPI(int) avs_deletedocid (
  IN avs_idxHdl_t idx,        /* Index handle (from avs_open) */
  IN const char* pDocid,      /* Document Id */
  OUT int* pCount      /* number of documents found/deleted */
);
```

Arguments

idx	The index handle.
pDocid	String that identifies the document (limited to 120 bytes and case sensitive).
pCount	The location to which the number of documents found and marked for deletion is returned.

Description

The **avs_deletedocid** procedure marks for deletion all documents with a specified docid (if any exist). If there are any documents with the specified docid in the **pending document set** (those documents added since the last call to **avs_makestable**), an error (AVS_DOC_EXISTS) is returned and no documents are deleted. The number of documents that are found is returned in the *pCount* argument.

For the documents to actually be deleted, you must call the **avs_makestable** procedure. If you insert the call to **avs_makestable** immediately after the **avs_deletedocid** procedure, the deletion will be effective immediately.

The document ID string comparison is case-sensitive.

Return Values

AVS_OK or an error code

See also

- avs_search
- avs_search_genrank

avs_enddoc

Terminates the sequence of calls for adding a document to the index begun by the **avs_startdoc** procedure.

C Synopsis

```
AVSAPI(int) avs_enddoc (
  IN avs_idxHdl_t idx
);
```

Arguments

idx	The index handle.

Description

The **avs_enddoc** procedure terminates a sequence of calls for adding a document to the index by the **avs_startdoc** procedure.

The **avs_startdoc**/**avs_enddoc** calls are an alternative to the use of **avs_newdoc** and are used with applications like Java and Visual Basic. Call **avs_startdoc** with the same arguments as the **avs_newdoc** procedure with the exception of the filter function and its argument. The first location available in the index for the new document is returned through the *pStartLoc* argument. Use this in the first call to the **avs_addword** procedure or similar kinds on procedures. When you are finished adding document contents, call the **avs_enddoc** procedure to terminate the document.

After a call to the **avs_startdoc** procedure and before a call to the **avs_enddoc** procedure, you can use exactly those calls which your filter procedure would have used, for example, **avs_addword** or **avs_addliteral**.

Return Values

AVS_OK or an error code.

See also

- avs_newdoc
- avs_startdoc

avs_errmsg

Returns a pointer to error message text associated with an error code.

C Synopsis

```
AVSAPI(char *)avs_errmsg(int code);
```

Arguments

char	Error message text.

Description

The **avs_errmsg** procedure returns a text string associated with an error code. It is used to translate any non-zero status codes returned by any of the Index procedures to a printable string (English).

Return Values

Error message text.

See also

- avs_errmsg_copy

avs_errmsg_copy

Copies the error message string to a buffer.

C Synopsis

```
AVSAPI(void)avs_errmsg_copy(int e, char * buf, int bufsiz);
```

Arguments

buf	Buffer for the error message text.

Description

The **avs_errmsg_copy** procedure copies the error message text to a buffer.

Return Values

None.

See also

- avs_errmsg

avs_getindexmode

Returns whether the current index is in build or query mode.

C Synopsis

```
AVSAPI(int) avs_getindexmode (
  IN avs_idxHdl_t pHdl  /* ptr to index handle */
);
```

Arguments

pHdl	Pointer to the index handle.

Description

Returns 0 if the index is in query mode, or 1 if the index is in build mode.

Return Values

Value Returned	Description
0	The index is in query mode.
1	The index is in build mode.

See also

- avs_compact
- avs_getindexversion
- avs_makestable

avs_getindexversion

Returns the current stable version number of the index.

C Synopsis

```
AVSAPI(int) avs_getindexversion (
  IN avs_idxHdl_t idx  /* Index handle (from avs_open) */
);
```

Arguments

idx	String specifying the index handle.

Description

Returns the current stable version of the index. The index version number reflects increments resulting from the avs_makestable or avs_compact procedures.

Return Values

Returns the version number of the index.

See also

- avs_compact
- avs_getindexmode
- avs_makestable

avs_getindexversion_counts_v

Returns the current stable version number of the index from the given counts context.

C Synopsis

```
AVSAPI(int) avs_getindexversion_counts_v (
  IN avs_countsHdl_t countsHdl  /* counts handle */
);
```

Arguments

idx	String specifying the counts handle.

Description

The **avs_getindexversion_counts_v** procedure returns the current stable version of the index from the given counts context. The index version number reflects increments resulting from the avs_makestable or avs_compact procedures.

Return Values

AVS_OK or an error code.

See also

- avs_compact
- avs_getindexmode
- avs_getindexversion
- avs_getindexversion_search_v
- avs_makestable

avs_getindexversion_search_v

Returns the current stable version number of the index from the given search context.

C Synopsis

```
AVSAPI(int) avs_getindexversion_search_v (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

idx	String specifying the index handle.

Description

The **avs_getindexversion_search_v** procedure returns the current stable version of the index from the given search context. The index version number reflects increments resulting from the avs_makestable or avs_compact procedures.

Return Values

AVS_OK or an error code.

See also

- avs_compact
- avs_getindexmode
- avs_getindexversion
- avs_makestable

avs_getsearchresults

Retrieves results of a search.

C Synopsis

```
AVSAPI(int) avs_getsearchresults (
  IN avs_searchHdl_t searchHdl,  /* search handle (from avs_search   */
  IN int resultNum               /* which document from results list */
);
```

Arguments

searchHdl	The search handle (from **avs_search**).
resultNum	Specifies the document to get from the results list.

Description

The **avs_getsearchresults** procedure is used to retrieve specific search results after calling the **avs_search** procedure and place the results in the search handle. An ordinal value specifies which result to retrieve. This ordinal must be a value between 0 and the number of documents returned by **avs_search** (in the *pDocsReturned* argument), minus 1. This procedure retrieves various document attributes, such as, relevancy value, document date, document Id, and document data and makes the value of the attributes available through the search results handle.

Return Values

AVS_OK or an error code.

See also

- avs_getsearchterms
- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance

avs_getsearchterms

Retrieves one ranking term and statistics for a search.

C Synopsis

```
AVSAPI(int) avs_getsearchterms (
  IN avs_searchHdl_t pSearchHdl,/* search handle */
  IN int termNum,/* which term from results list */
  OUT char **term,/* term (storage released by avs_search_close) */
  OUT long *count /* # occurrences...-1 means too many, ignored */
);
```

Arguments

psearchHdl	The search handle (from **avs_search**).
termNum	Specifies which term to retrieve from the results list.
****term***	Receives pointer to term string.
count	Receives the number of occurrences of a term.

Description

The **avs_getsearchterms** procedure is used to retrieve a term and term statistics for the specified search. The term to retrieve is the 0-relative result number specified in the second argument.

Return Values

AVS_OK or an error code.

See also

- avs_getsearchresults
- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance

avs_getsearchversion

Retrieves a *version string* which defines the version of the index used for this search.

C Synopsis

```
  AVSAPI(int) avs_getsearchversion (
  IN avs_searchHdl_t pSearchHdl,  /* search handle */
  OUT char * searchversion
);
```

Arguments

searchHdl	The search handle (from **avs_search**).
searchversion	Specifies the version of the index for which you have a search result.

Description

The **avs_getsearchversion** procedure retrieves a *version string* which defines the version of the index used for this search. This string can be passed to the **avs_search_ex** or **avs_search_genrank** procedures to limit results to documents added since that version.

The version string will not be more than the maximum length defined by `AVS_SEARCHVERSION_MAXLEN`. The default is 30 bytes long, including the terminating null byte.

Return Values

AVS_OK or an error code.

See also

- avs_getsearchterms
- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance

avs_lookup_valtype

Looks up a value type by name.

C Synopsis

```
AVSAPI(avs_valtype_t) avs_lookup_valtype (
  IN char * name
  );
```

Arguments

name	The pointer to the string that contains the name of the value type.

Description

The **avs_lookup_valtype** procedure is used to lookup value types defined by the **avs_define_valtype** procedure. It returns a NULL or a pointer to the type.

Return Values

NULL or a pointer to the type.

See also

- avs_define_valtype
- avs_release_valtypes
- avs_search_genrank

avs_makestable

Commits any pending index updates to disk.

C Synopsis

```
AVSAPI(int) avs_makestable (
  IN avs_idxHdl_t idx   /* Index handle (from avs_open) */
);
```

Arguments

idx	The index handle.

Description

The **avs_makestable** procedure saves recent deletions or additions made to the index file. The documents and words added to the index are flushed to the disk, and all documents marked for deletion by a previous call to **avs_deletedocid**, are deleted. This procedure should be called before closing the index and after every 500,000 or so words have been indexed.(This number depends on available memory - larger systems can wait longer between makestable procedures.)

This procedure finishes the job of adding and deleting documents and makes the newly added documents "searchable" while removing the documents marked for deletion by the **avs_deletedocid** procedure.

The application should also compact the index after a series of updates is completed, to improve subsequent query performance or to recover space from deleted entries. Your application should test if compaction is needed (**avs_compactionneeded**) after each **avs_makestable** call and perform a **avs_compact_minor** loop if the answer is yes. If you defer compaction too long, the makestable procedure will take longer and longer to complete.

Return Values

AVS_OK or an error code.

See also

- avs_close
- avs_compact
- avs_deletedocid
- avs_newdoc
- avs_open

avs_newdoc

Creates a new document or replaces an existing document in the index.

C Synopsis

```
AVSAPI(int) avs_newdoc (
  IN avs_idxHdl_t idx,       /* Index handle (from avs_open) */
  IN void * pFilterdata, /* info identifying the document -passed to filter
*/
  IN filter_p_t pFilter, /* ptr to filter function */
 IN const char *pDocId,      /* document identifier string */
  IN int flags,     /* conditions: docid must NOT already exist (1),
           docid MUST already exist (2),
           duplicate docid's allowed (4)*/
  OUT unsigned long *pNumWords /* total words in document */
);
```

Arguments

idx	The index handle.
pFilterdata	Pointer to arbitrary data need by the filter procedure, such as the file path or the database key. This pointer is passed to the filter procedure.
filter_p_t pFilter	A pointer to the filter procedure that prepares the document for indexing.
pDocId	String that names the document (limited to 120 bytes and case sensitive).
flags	Sets the conditions for creating a new document. The conditions can have the following flags: **0** Does not matter whether the document already exists. If it does not exist, create it. If it does exist, replace it. **1** The docid must not already exist. **2** The docid must already exist. **4** Duplicate document IDs are allowed. If a document with the same ID already exists, another one can also be created.
pNumWords	The total number of words in the document

Description

The **avs_newdoc** procedure creates a new document in a specified index. A filter procedure (callback function) must be defined which is responsible for adding words to the index (see Creating a Filter Procedure and **avs_addword** for more information).

The filter callback is called with the following arguments:

```
IN avs_idxHdl_t idx              (index handle )
   IN void * pFilterdata,       (info identifying the document)
   IN unsigned long startloc,   (starting location for adding words or
literals)
   OUT unsigned long *pNumWords (the number of words added to the index)
```

The function should return the number of words added to the index in the last argument. The filter function should also return AVS_OK to signal success, or AVS_FILTER_ERR to signal an error. A filter error will cause **avs_newdoc** to return AVS_FILTER_ERR.

An Alternative to the *avs_newdoc* Procedure

Call **avs_startdoc** with the same arguments as the **avs_newdoc** procedure with the exception of the filter function and its argument. The first location available in the index for the new document is returned through the *pStartLoc* argument. Use this in the first call to the **avs_addword** procedure or similar kinds on procedures. When you are finished adding document contents, call the **avs_enddoc** procedure to terminate the document.

After a call to the **avs_startdoc** procedure and before a call to the **avs_enddoc** procedure, you can use exactly those calls which your filter procedure would have used (for example, **avs_addword** or **avs_addliteral**).

Use of Flags

When you use the flags to cause automatic deletion of previous docid instances, it makes the normal, periodic document update easy. However, it does not protect against other instances of the same doc-id within the pending document set in memory, only against those documents recorded onto disk at some previous pass. If there are any documents with the specified docid in the **pending document set** (those documents added since the last call to **avs_makestable**), an error (AVS_DOC_EXISTS) is returned and no documents are deleted.

Return Values

AVS_OK, AVS_FILTER_ERR or an error code.

See also

- avs_close
- avs_makestable
- avs_open

avs_open

Opens an index for querying or for modifying.

C Synopsis

```
AVSAPI(int) avs_open (
  IN const avs_parameters_t * parameters,
  IN const char * path,        /* path name to index location on disk */
  IN const char * mode,        /* mode "r" for read-only, "rw" for modify,
"ro" for cdrom */
  OUT avs_idxHdl_t * pIdx    /* new index handle */
);
```

Arguments

parameters	Pointer to the avs_parameter block.
path	String that is the full or relative path of the index directory on disk.
mode	String that specifies whether the index should be opened as a read only (r) or a read write (rw) index.
pIdx	The location to receive the handle of the opened index. This handle is used in subsequent index functions.

Description

The **avs_open** procedure opens the index pointed to by the *path* argument. The *mode* parameter can be either "r" for read only, "rw" for read-write, and "ro" for CD-ROM. When "rw" is set, the index is opened for reading and writing. Appropriate locks are set to interlock reads and updates in other processes.

Note: The directory specified by *path* must exist. If the directory specified by *path* is empty, a new index will be created.

Return Values

AVS_OK or an error code.

See also

- avs_close
- avs_makestable
- avs_newdoc

avs_querymode

Optimizes an index for optimum query performance.

C Synopsis

```
AVSAPI(int) avs_querymode (
  IN avs_idxHdl_t idx  /* Index handle (from avs_open) */
);
```

Arguments

idx	The index handle.

Description

The **avs_querymode** procedure optimizes an index for response to search calls. This allows optimal response in querying the index when use of the query interface is high. The new mode takes effect immediately. This state is not retained at the next call to **avs_open** unless either the **avs_makestable** or **avs_compact** procedure is called.

This procedure may cause a full compaction of the index.

Return Values

AVS_OK or an error code.

See also

- avs_buildmode
- avs_makestable
- avs_newdoc

avs_release_valtypes

Releases all value type definitions.

C Synopsis

```
AVSAPI(void) avs_release_valtypes(void);
```

Description

The **avs_release_valtypes** procedure releases any value types that have been defined. This procedure should only be called after your application's last call to **avs_close**.

Return Values

AVS_OK or an error code.

See also

- avs_define_valtype
- avs_lookup_valtype

avs_search

Searches for documents that match a query expression and the given search parameters.

C Synopsis

```
AVSAPI(int) avs_search (
 IN avs_idxHdl_t idx,                    /* Index handle (from avs_open) */
 IN const char * pQuery,                 /* simple query expression  */
 IN const char * pBoolQuery,             /* boolean query expression */
 IN avs_options_p_t pOptions,            /* query options */
 OUT long *pDocsFound,                   /* no. of documents found */
 OUT long *pDocsReturned,                /* no. of documents returned */
 OUT long *pTermCount,                   /* no. of terms in rank string */
 OUT avs_searchHdl_t *pSearchHdl         /* search handle (used with
avs_getsearchresult) */
);
```

Arguments

idx	The index handle.
pQuery	Pointer to a simple (or ranking) query expression. May be NULL.
pBoolQuery	Pointer to a boolean query expression.May be NULL.
pOptions	The query options.
pDocsFound	Pointer to the number of documents found.
pDocsReturned	Pointer to the number of documents returned. This is the same as the pDocsFound argument unless an argument of the **avs_default_options** structure limits the number of documents returned, or if ranking terms are present.
pTermCount	The pointer to the number of terms in the ranking string.
pSearchHdl	The search handle to use with **avs_getsearchresults**.

Description

The **avs_search** performs a search for documents in an index. A simple query expression can contain words, phrases, the asterisk (*) wildcard character, and the + and - operators, which require or prohibit the presence of a word in the search results. A Boolean search expression uses the logic operators AND, OR, NOT, NEAR, or WITHIN.

This procedure searches the index and reports the following results:

- the number of documents found
- the number of documents actually returned for inspection (subject to the supplied limit)
- a handle for the returned documents

It is possible to use this interface to perform consistently with either the AltaVista **simple search** or the AltaVista **advanced search** function (as seen at http://altavista.digital.com). You can also effectively combine both approaches in one call.

To Perform a Simple Search

Use the *pQuery* argument to point to the simple query string. Do not use the *pBoolean* argument. Set AVS_OPT_FLAGS_RANK_TO_BOOL flag to 1.

To Perform an Advanced Search

Use the *pQuery* argument to point to ranking terms. Use the *pBoolean* argument to point to the advanced query string. Set AVS_OPT_FLAGS_RANK_TO_BOOL flag to 0.

The Combined Approach -- Simple Query with Boolean Qualifiers

Use the *pQuery* argument to point to the simple query string and use the *pBoolean* argument to point to the advanced query string. Set AVS_OPT_FLAGS_RANK_TO_BOOL flag to 1. In this case, the behavior is that of the simple query, filtered by the boolean expression, that is, the results are the **intersection** of what would be produced by the simple query and the boolean query separately.

For more information, see Searching the Index. In all cases, the results are ranked according to the relative weighted occurrence of the (non-negative) terms in the simple query (or ranking) string. If there are no such terms, the result is unranked.

Additional Query Option

The **avs_search** procedure has an option which lets you control ranking weight of search terms:

AVS_OPT_FLAGS_NO_POS_BOOST - If you set this flag to 1, the extra weight that a search term occurring early on in the document receives is suppressed. Normally, if a search term occurs in the first eight words of a document (for example, the title), it receives extra weight in the ranking process. If the search term occurs in the first 32 words of a document, it also receives extra weight but not as much as if it occurred in the first 8.

Return Values

AVS_OK or an error code.

See also

- avs_default_options
- avs_getsearchresults
- avs_getsearchterms
- avs_search_ex

avs_search_close

Closes a search request.

C Synopsis

```
AVSAPI(int) avs_search_close (
  IN avs_searchHdl_t SearchHdl  /* the search handle */
);
```

Arguments

searchHdl	The search handle.

Description

The **avs_search_close** procedure closes a search. This must be called when all calls to avs_getsearchresults are completed, to release the resources allocated for a search.

Return Values

AVS_OK or an error code.

See also

- avs_makestable
- avs_newdoc
- avs_open

avs_search_ex

Searches for documents that match a query expression and the given search parameters with the *searchsince* option.

C Synopsis

```
AVSAPI(int) avs_search_ex (
     IN avs_idxHdl_t idx,            /* Index handle  */
  IN const char * pQuery,          /* simple query expression  */
  IN const char * pBoolQuery,      /* boolean query expression  */
  IN const avs_options_p_t pOptions,     /* options           */
  IN const char * searchsince,   /* NULL, or version string */
  OUT long *pDocsFound,            /* no. of documents found */
  OUT long *pDocsReturned,         /* no. of documents returned */
  OUT long *pTermCount,   /* no. of terms used in ranking */
  OUT avs_searchHdl_t *pSearchHdl  /* search handle (used with
avs_getsearchresult)*/);
```

Arguments

idx	The index handle.
pBoolQuery	Pointer to a boolean query expression.
pOptions	The query options.
searchsince	Version string or NULL.
pDocsFound	Pointer to the number of documents found.
pDocsReturned	Pointer to the number of documents returned. This is the same as the pDocsFound argument unless an argument of the **avs_default_options** structure limits the number of documents returned.
pTermCount	Pointer to the number of terms used in ranking.
pSearchHdl	The search handle to use with **avs_getsearchresults**.

Description

The **avs_search_ex** lets you specify a *searchsince* argument that returns results from documents that have been added since the last search.

For more information, see Searching the Index.

Return Values

AVS_OK or an error code.

See also

- avs_default_options
- avs_getsearchresults
- avs_getsearchterms
- avs_search

avs_search_genrank

Searches for documents that match a query expression and ranks the results according to the ranking expression.

C Synopsis

```
   AVSAPI(int) avs_search_genrank (
  IN avs_idxHdl_t idx,             /* Index handle  */
  IN const char * pBoolQuery,      /* boolean query expression  */
  IN const char * pRankTerms,      /* generic ranking expression */
  IN avs_ranksetup_t * pRankSetup, /* NULL, or special ranking setup */
  IN const avs_options_p_t pOptions,     /* options          */
  IN const char * searchsince,   /* NULL, or version string */
  OUT long *pDocsFound,            /* no. of documents found */
  OUT long *pDocsReturned,         /* no. of documents returned */
  OUT avs_searchHdl_t *pSearchHdl  /* search handle (used with
avs_getsearchresult) */
 );
```

Arguments

idx	The index handle.
pBoolQuery	The Boolean query expression.
pRankTerms	The generic ranking expression.
pRankSetup	NULL or the special ranking expression.
pOptions	The options specified in the avs_parameter structure.
searchsince	NULL or version string.
pDocsFound	Pointer to the number of documents found.
pDocsReturned	Pointer to the number of documents returned.
avs_searchHdl_t searchHdl	The handle of the search result about which you are requesting more information.

Description

The **avs_search_genrank** procedure retrieves information about a search result and stores it in a buffer. To perform the Boolean query, use the [X-Y] form for the search terms. If you supply a ranking term that does not exist in the index, or supply no ranking term, no results are returned. This behavior is different from that of **avs_search** in that the ranking term, if it is supplied, may or may not exist in the index.

The ranking expression can consist of an **application-defined** ranking term, or one of the **pre-defined** ranking terms.

The **pre-defined** ranking terms are:

- *#date* -- ranks documents according to date
- *#time* -- ranks documents according to date and time

If the term is preceded by a minus sign (-), the rank order is reversed. For example,

- **#date** produces most-recent dates first

- **-#date** produces oldest dates first

Granularity of ranking

- When ranking documents by **#date**, the granularity is a **day** on all platforms.
- When ranking documents by **#time**, the granularity is a **minute** on Windows NT, Solaris, Linux (Intel).
- When ranking documents by **#time**, the granularity is a **second** on *Compaq Tru64 UNIX* (DIGITAL UNIX).

The **application-defined** ranking term must be added to the index with **avs_setrankval**. You can also reverse the sort order of search results by preceding this ranking term with a minus sign. The *pRankSetup* argument should always be NULL.

If you have multiple-valued rank values, the only valid use for the multiple-valued rank values is for filtering.

Return Values

Number of documents found, the number of documents returned.

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getdocid_copy
- avs_search_getrelevance

avs_search_getdata

Returns the data associated with a search result.

C Synopsis

```
AVSAPI(void *) avs_search_getdata (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result about which you are requesting more information.

Description

The **avs_search_getdata** procedure retrieves information about a search result. Often the document data consists of the title or the first several lines of the document. It must be preceded by a call to **avs_getsearchresults** and a call to **avs_search_getdatalen** to determine the size of the buffer to allocate for the document data.

If your index has documents that have multiple dates associated with it, and the dates have been added by a combination of avs_setdocdate, avs_setdocdatetime, and avs_adddate, search and ranking behavior is described below: **Search by date:** Any date associated with a document that falls within the range specified in a search will result in that document being returned.

Rank by date: Ranking documents by date will only reflect the date added by **avs_setdocdate** or **avs_setdocdatetime**.

Return Values

A pointer to the document data.

See also

- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance

avs_search_getdata_copy

Returns the data associated with a search result and stores it in a buffer.

C Synopsis

```
 AVSAPI(void) avs_search_getdata_copy (
  IN avs_searchHdl_t searchHdl,  /* search result handle */
  IN void * databuf,
  IN int buflen
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result about which you are requesting more information.
databuf	The buffer which contains requested of the search result.
buflen	The buffer length of databuf.

Description

The **avs_search_getdata_copy** procedure retrieves at most *buflen* bytes of information about a search result and stores it in *databuf*. Applications like Java or VisualBasic can retrieve the information in a usable format. Often the document data consists of the title or the first several lines of the document. It must be preceded by a call to **avs_getsearchresults**.

Return Values

None.

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getdocid_copy
- avs_search_getrelevance

avs_search_getdatalen

Returns the length of the data associated with a search result.

C Synopsis

```
 AVSAPI(int) avs_search_getdatalen (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result about which you are requesting more information.

Description

The **avs_search_getdatalen** procedure retrieves the length of the document data associated with a search result. Often the data consists of the title or the first several lines of the document.

Return Values

The length in bytes of the document data.

See also

- avs_search_getdata
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance
- avs_setdocdate

avs_search_getdate

Returns the date associated with a search result.

C Synopsis

```
 AVSAPI(void) avs_search_getdate (
  IN avs_searchHdl_t searchHdl,  /* search result handle */
  OUT int *dateY,        /* Year 0100 <> 2148*/
  OUT int *dateM,        /* Month (1-12) */
  OUT int *dateD         /* Day (1-31) */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result about which you are requesting more information.
dateY	The year greater than 01/01/0100 but less than 12/31/2148 inclusive.
dateM	The month of the year (1-12).
dateD	The day of the month (1-31).

Description

The **avs_search_getdate** procedure retrieves the date the document was indexed as set by **avs_setdocdate**. The **avs_search_getdate** procedure must be preceded by a call to **avs_getsearchresults**.

Return Values

The date of the document. (See avs_setdocdate for date format.)

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdocid
- avs_search_getrelevance
- avs_setdocdate

avs_search_getdocid

Returns the unique identifier associated with a search result.

C Synopsis

```
 AVSAPI(unsigned char *) avs_search_getdocid (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result for which you are requesting the document ID.

Description

The **avs_search_getdocid** procedure retrieves the identifier of the search result document.

Return Values

A pointer to the document identifier string. Valid until the next call to **avs_getsearchresult** or **avs_search_close**.

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getrelevance

avs_search_getdocid_copy

Returns the unique document identifier associated with a search result to the caller's buffer.

C Synopsis

```
 AVSAPI(void) avs_search_getdocid_copy (
  IN avs_searchHdl_t searchHdl,  /* search result handle */
  IN unsigned char * databuf,    /* caller's buffer */
  IN int buflen                  /* length of caller's buffer */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result for which you are requesting the document identifier.
databuf	The buffer which contains the value of search result handle.
buflen	The buffer length of databuf.

Description

The **avs_search_getdocid_copy** procedure stores the document identifier of the search result in a buffer. Because applications like Java and Visual Basic cannot use the format (char *) of the return in the **avs_search_getdocid** procedure, **avs_search_getdocid_copy** stores the document identifier of the search result in a buffer to be called by these types of applications.

Return Values

None

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getdocidlen
- avs_search_getrelevance

avs_search_getdocidlen

Returns the length of the document identifier associated with a search result.

C Synopsis

```
AVSAPI(int) avs_search_getdocidlen (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result for which you are requesting the length of the document identifier.

Description

The **avs_search_getdocidlen** procedure retrieves the length of the document identifier for the search result.

Return Values

The length of the document identifier string.

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid
- avs_search_getrelevance

avs_search_getrelevance

Returns the relevance value associated with a search result.

C Synopsis

```
AVSAPI(float) avs_search_getrelevance (
  IN avs_searchHdl_t searchHdl  /* search result handle */
);
```

Arguments

avs_searchHdl_t searchHdl	The handle of the search result for which you are requesting the relevance value.

Description

The **avs_search_getrelevance** procedure retrieves the relevance value associated with a search result. The closer the value is to 1, the more useful and relevant the search result is likely to be.

A search result can also have a relevancy ranking of zero (0). In this case, all results have the same weight or are equally relevant. A relevancy ranking of zero can happen in the case where the search did not have a ranking string.

For more details, see Understanding Relevance Ranking.

Return Values

The relevance value, expressed as a floating value (float).

See also

- avs_search_getdata
- avs_search_getdatalen
- avs_search_getdate
- avs_search_getdocid

avs_setdocdata

Sets the document data for a document being added with the avs_newdoc procedure. A filter makes this call.

C Synopsis

```
AVSAPI(int) avs_setdocdata (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN const void *pDocdata,/* ptr to document data */
  IN int len/* length of doc data (bytes) */
);
```

Arguments

idx	Index handle.
pDocdata	Pointer to the document data.
len	Length of document data in bytes.

Description

The **avs_setdocdata** procedure sets the document's data (for example, the title of the document or other descriptive information). The data can be arbitrary byte-oriented data. A filter calls this procedure after analyzing the document's content. The data is available after a successful search.

Return Values

AVS_OK or an error code.

See also

- avs_addfield
- avs_addword
- avs_setdocdate

avs_setdocdate

The **avs_setdocdate** procedure sets the date on a document being added with the avs_newdoc procedure. A filter makes this call.

C Synopsis

```
AVSAPI(int) avs_setdocdate (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN int dateY,/* Year 0100 <>2148 */
  IN int dateM,         /* Month (1-12) */
  IN int dateD          /* Day (1-31) */
);
```

Arguments

idx	The index handle.
dateY	The year greater than 01/01/0100 but less than 12/31/2148 inclusive.
dateM	The month of the year from 1 to 12.
dateD	The day of the month from 1 to 31.

Description

The **avs_setdocdate** procedure sets the date of the document. A filter calls this procedure.

The date is returned in the search results and can be retrieved by calling **avs_search_getdate()** on the search results handle. Dates are indexed and can be used to limit searches by adding a date range as an additional term in the boolean query string argument passed to **avs_search**.

The date and time (see **avs_setdocdatetime**)can also be used to order search results. For further information see **avs_search_genrank**.

Return Values

AVS_OK or an error code.

See also

- avs_addfield
- avs_addword
- avs_setdocdata

avs_setdocdatetime

Sets the date and time on a document being added with the avs_newdoc procedure. A filter makes this call.

C Synopsis

```
AVSAPI(int) avs_setdocdatetime (
  IN avs_idxHdl_t idx,  /* Index handle  */
  IN int dateY,/* Year (>0100) */
  IN int dateM,/* Month (1-12) */
  IN int dateD,/* Day (1-31) */
  IN int timeH,/* Hour (0-23) */
  IN int timeM,/* Minute (0-59) */
  IN int timeS/* Second (0-59) */
);
```

Arguments

idx	The index handle.
dateY	The year greater than 01/01/0100 but less than 12/31/2148 inclusive.
dateM	The month of the year from 1 to 12.
dateD	The day of the month from 1 to 31.
timeH	The hour from 0 to 23.
timeM	The minute from 0 to 59.
timeS	The seconds from 0 to 59.

Description

The **avs_setdocdatetime** procedure sets the date and time of the document. A filter calls this procedure.

The date is returned in the search results and can be retrieved by calling **avs_search_getdate()** on the search results handle.

Dates are indexed and can be used to limit searches by adding a date range as an additional term in the boolean query string argument passed to **avs_search**.

Return Values

AVS_OK or an error code.

See also

- avs_addword
- avs_setdocdata
- avs_setdocdate

avs_setparseflags

Sets **avs_addword** parsing options.

C Synopsis

```
AVSAPI(void) avs_setparseflags (
  IN avs_idxHdl_t idx,   /* Index handle (from avs_open) */
  IN int parseflags
);
```

Arguments

idx	The index handle.

Description

The **avs_setparseflags** procedure sets the parsing options for the **avs_addword** procedure. Currently, either zero (0) or AVS_PARSE_SGML is the only valid value for this procedure. AVS_PARSE_SGML allows the indexer to recognize SGML encoded entities as characters and add words containing those characters to the index. For example, *é* is the SGML encoding e with the acute accent.

Return Values

AVS_OK or an error code.

See also

- avs_addword

avs_setrankval

Adds a numeric value to a document index that can be used for ranking instead of document date or time.

C Synopsis

```
AVSAPI(int) avs_setrankval (
  IN avs_idxHdl_t idx,  /* Index handle (from avs_open) */
  IN const avs_valtype_t valtype, /* type to use */
  IN unsigned long value/* value */
);
```

Arguments

idx	The index handle.
valtype	The value type.
value	The value.

Description

The **avs_setrankval** procedure adds a numeric value to a document index that can be used for ranking. A given value type defined with **avs_define_valtype** should be used at most once in a document. Multiple values per document can be set if the value type is defined with **avs_define_valtype_multiple**.

Return Values

AVS_OK or an error code.

See also

- avs_addvalue
- avs_define_valtype
- avs_define_valtype_multiple
- avs_search_genrank

avs_startdoc

Prepares to create a new document in the index.

C Synopsis

```
AVSAPI(int) avs_startdoc (
  IN avs_idxHdl_t idx,   /* Index handle  */
  IN const char *pDocId,       /* document identifier string */
  IN int flags,     /* conditions: docid must NOT already exist (1),
            docid MUST already exist (2),
            duplicate docid's allowed (4)*/
  OUT unsigned long *pStartLoc /* first available location in new doc */
);
```

Arguments

idx	The index handle.
pDocId	String that identifies the document (limited to 120 bytes and case sensitive).
flags	Sets the conditions for creating a new document. The conditions can have the following flags: • **0** Does not matter whether the document already exists. If it does not exist, create it. If it does exist, replace it. • **1** New document. The docid must not already exist. • **2** Replace an existing document. • **4** Duplicate document IDs are allowed. If a document with the same ID already exists, another one can also be created.
pStartLoc	Returns the first location available in the index for the new document.

Description

The **avs_startdoc** procedure creates a new document in the index in a linear fashion useful to applications like Java and Visual Basic. This procedure must be paired with a call to **avs_enddoc** to bracket the beginning and end of the document to be added to the index.

The same arguments in **avs_newdoc** , with the exception of the filter function and its argument, are also used in this procedure.

The first location available in the index for the document is returned though the *pStartLoc* argument. Use this in the first call to the **avs_addword** procedure or similar kinds of procedures. When you are finished adding document contents, call the **avs_enddoc** procedure to terminate the document.

After a call to the **avs_startdoc** procedure and before a call to the **avs_enddoc** procedure, you can use only those calls which would be allowed in avs_newdoc filter operations (for example, avs_addword or avs_addliteral).

If the flags are set to replace an existing document, and if there are any documents with the specified docid in the **pending document set** (those documents added since the last call to **avs_makestable**), an error (AVS_DOC_EXISTS) is returned and no documents are added or deleted.

Return Values

AVS_OK or an error code.

See also

- avs_enddoc
- avs_newdoc

avs_timer

Sets a timeout value for query processing.

C Synopsis

```
AVSAPI(void) avs_timer (
  IN const unsigned long current
);
```

Arguments

const	The value of the timeout range.

Description

The **avs_timer** procedure is used by an application's timer thread to pass a current timer value from the application into AltaVista Search. In this way, search operations can be limited in processing duration. If the application does not have a timer thread, no search timeouts will occur.

In *avs_options.timeout*, you can set the number of timer units allowed per query. At the start of each search, AltaVista Search sets a timer limit equal to the current timer value plus the value of *avs_options.timeout*. It periodically checks the current timer value against the timer limit. When the current timer value is greater than the limit, the search process stops and returns the partial results accumulated so far.

Return Values

AVS_OK or an error code.

See also

- avs_enddoc
- avs_newdoc

avs_version

Returns a pointer to library version strings.

C Synopsis

```
AVSAPI(const char**) avs_version (void);
```

Description

The **avs_version** procedure returns the pointer to the library version strings. The version strings can be a search version or an index version. The strings contain such information as:

- The avs_implementation version
- The interface version
- Whether the application is single or multi-threaded
- The build number
- The license thread

These strings are useful in problem reporting and in keeping track of the current search context.

Return Values

AVS_OK or an error code.

See also

- avs_enddoc
- avs_newdoc

avsi_setdocdata

Sets the document data for an AltaVista Intranet index.

C Synopsis

```
AVSAPI(int)avsi_setdocdata (
  IN avs_idxHdl_t pIdx,         /* Index handle (from avs_open) */
  IN avsi_docdata_t *pDocdata   /* Document data */
```

Arguments

pIdx	The index handle from **avs_open**.
pDocdata	The document data.

Description

The **avsi_setdocdata** procedure sets the document data for an AltaVista Intranet index. Use the avsi_docdata structure to set the following:

- URL
- Document title
- Document abstract
- Language
- Character set encoding

Return Values

AVS_OK or an error code.

See also

- avsi_docdata structure avsi_getdocdata

avsi_getdocdata

Retrieves the document data from an AltaVista Intranet index.

C Synopsis

```
AVSAPI(int)avsi_getdocdata (
  IN  avs_searchHdl_t searchHdl,   /* search result handle */
  OUT avsi_docdata_t *pDocdata    /* Document data */
);
```

Arguments

searchHdl	The search handle result.
pDocdata	The document data.

Description

The **avsi_getdocdata** procedure retrieves the document data from an AltaVista Intranet index.

Return Values

AVS_OK or an error code.

See also

- avsi_setdocdata
- avsi_docdata structure
- avsi_url2docid

avsi_url2docid

Creates a suitable document ID from a URL.

C Synopsis

```
AVSAPI(void) avsi_url2docid (
  IN  char *pURL,        /* URL string */
  OUT char *pDocid       /* ptr to buffer to contain doc ID */
```

Arguments

pURL	The URL string.
pDocid	The pointer to buffer to contain doc ID.

Description

The **avsi_url2docid** procedure creates a suitable document ID from a URL. This ID is used to add or delete a document to or from the AltaVista Intranet index.

Return Values

AVS_OK or an error code.

See also

- avsi_getdocdata
- avsi_setdocdata
- avsi_url2docid

avsi_convert_to_UTF8

Converts a string of characters from the specified encoding character set to UTF-8.

C Synopsis

```
   AVSAPI(int) avsi_convert_to_UTF8 (
    IN    char *p_buf,      /* the string to convert */
    IN    int sizebuf,      /* length of string to convert */
    OUT   char *p_utf8buf,  /* buffer to contain UTF8 string */
    IN    int size_utf8buf, /* max size of UTF8 buffer */
    IN    char * charset    /* character set, e.g. EUCKR_NAME  */
    /* NOTE: use EUC-CN for 'Simplified Chinese', not GB */
);
```

Arguments

p_buf	The string to convert.
sizebuf	The length of the string to convert.
p_utf8buf	The buffer to contain UTF8 string.
size_utf8buf	The maximum size of the UTF8 buffer.
charset	The character set, for example, EUCKR_NAME. See `avsi_compat.h`.

Description

The **avsi_convert_to_UTF8** procedure converts a string of characters from the specified encoding character set to UTF8. It returns the length of the UTF8 string or a negative number if the conversion failed. If you are indexing non-ASCII text, use this function to convert the native characters to UTF8. Also, be sure to set the `AVS_CHARSET_UTF8` option when you open the index. Note: use **avsi_convert_cjkquery** to convert query strings for searching.

Return Values

The length of the UTF8 string or a negative number

See also

- avsi_convert_from_utf8
- avsi_getdocdata
- avsi_setdocdata
- avsi_url2docid

avsi_convert_from_UTF8

Converts a string of UTF8 characters to the specified encoding character set.

```
AVSAPI(int) avsi_convert_from_UTF8 (
    IN    char *p_utf8buf,  /* the UTF8 string to convert */
    IN    int size_utf8buf, /* length of UTF8 string to convert */
    OUT   char *p_buf,      /* buffer to contain converted string */
    IN    int sizebuf,      /* max size of of buffer */
    IN    char * charset    /* character set, e.g. EUCKR_NAME (see below) */
    /* NOTE: use EUC-CN for 'Simplified Chinese', not GB */
);
```

Arguments

p_utf8buf	The UTF8 string to convert.
size_utf8buf	The length of UTF8 string to convert.
p_buf	The buffer to contain converted string.
sizebuf	The maximum size of the buffer.
charset	The character set, for example, EUCKR_NAME. See `avsi_compat.h`.

Description

The **avsi_convert_from_UTF8** procedure converts a string of UTF8 characters to the specified encoding character set. It returns the length of the converted string or a negative number if the conversion failed.

Return Values

The length of the converted string or a negative number if the conversion failed

See also

- avsi_convert_to_utf8
- avsi_getdocdata
- avsi_setdocdata
- avsi_url2docid

avsi_convert_cjkquery

Converts a user query in one of the CJK character sets to a query in UTF8 format with space characters between CJK characters.

C Synopsis

```
   AVSAPI(int) avsi_convert_cjkquery(
    IN    char *p_buf,      /* the string to convert */
    IN    int len,          /* length of string to convert */
    OUT   char *p_buf_utf8, /* buffer to contain UTF8 string */
    IN    int maxsize_utf8, /* size of UTF8 buffer */
    /* NOTE: should be at least twice the size of the query buffer */
    IN    char * p_charset  /* character set e.g. EUCKR_NAME (see below) */
    /* NOTE: use EUC-CN for 'Simplified Chinese', not GB */
);
```

Arguments

p_buf	The string to convert.
len	The length of the string to convert.
p_buf_utf8	The buffer to contain the UTF8 string.
maxsize_utf8	The maximum size of UTF8 buffer.
p_charset	The character set, for example, EUCKR_NAME.

Description

The **avsi_convert_cjkquery** procedure converts a user query in one of the CJK character sets to a query in UTF8 format with space characters between CJK characters. It returns the length of the UTF8 string or a negative number if the conversion failed.

Note: this function assumes that the query string is null terminated.

The user query string is converted as shown in the following examples. The double lower-case letters represent double-byte Asian characters; upper-case letter represent regular ASCII.

```
input:  ALTAVISTA -aabb "SEARCH ENGINE"
output: ALTAVISTA -"aa bb" "SEARCH ENGINE"

input:  +DIGITAL aabb97cc +dd "aabb MICRO"
output: +DIGITAL "aa bb" 97 "cc" +"dd" " aa bb MICRO"

input:  title:HELLOaabbCHINA +"aabb MICRO"
output: title:HELLO "aa bb" CHINA +" aa bb MICRO"

input:  title:"HELLOaabbCHINA"
output: title:"HELLO aa bb CHINA"
```

Return Values

The length of the UTF8 string or a negative number if the conversion failed.

See also

- avsi_convert_to_utf8
- avsi_getdocdata
- avsi_setdocdata
- avsi_url2docid

Data Structures

The data structures and filter procedure are defined in the avs.h file:

avs_options

This structure is used to specify search options in the **avs_search** procedure. Call **avs_default_options** to initialize to default values. This structure contains the *timeout* parameter that controls the number of seconds to allow for each query.

```
   struct avs_options {
   long limit;     /* maximum number of documents returned
(default=32000) */
   int timeout;/* if nonzero, number of seconds to allow per query
*/
   int flags;/* additional query option flags */
   };
   typedef struct avs_options avs_options_t, *avs_options_p_t;
```

The following flags are additional query option flags set in the **avs_options** structure:

- AVS_OPT_FLAGS_RANK_TO_BOOL - if set to 1, the behavior is that of the simple query, filtered by the boolean expression, that is, the results are the of what would be produced by the simple query and the boolean query separately.
- AVS_OPT_FLAGS_NO_POS_BOOST - if set to 1, eliminate the higher weighting of words occurring at the beginning of a document.
- AVS_OPT_FLAGS_NO_LOGGING -if set to 1, no logging occurs during the querying operations.
- AVS_OPT_FLAGS_RANK_LATEST – if set to 1, the most recent documents added to the index are ranked higher. If this flag is off, documents added later will be ranked lower.

avs_parameters

The **avs_parameters** structure is used to affect all index operations. The parameters should maintain a constant value for the life of the application and if you change any, you should rebuild your index for consistency. The following operations can be managed by this structure:

- The interface version
- The license management
- The search operations
- The structure of the index

```
struct avs_parameters {
 char * _interface_version;  /* checked by library */
 char * license;            /* set to OEM license string */
 int ignored_thresh;         /* %(*100) for ranking ignore */
 int chars_before_wildcard;  /* min chars before wildcard */
 int unlimited_wild_words;   /* set to 1 to avoid 50 limit */
 int indexformat;        /* set to -1 for default, 0 for current */
 long cache_threshold;       /* max size file to cache (0=default)
*/
 int options;                /* optional indexing features */
 int charset;                /* character set in use */
 int ntiers, nbuckets;       /* max values (0 => use default) */
 };
```

These parameters affect all operations and are intended to have a constant value for the life of the application:

- _interface_version;
- license;

The next few parameters affect search operations but not the index structure itself:

- ignored_thresh
- chars_before_wildcard
- unlimited_wild_words
- indexformat
- cache_threshold

The following parameters affect the index structure, and can vary across indexes. However, they should be consistent over time for any given index or the results are undefined.

- options
- charset
- ntiers, nbuckets

Default Values

The *avs.h* file sets the following parameters with these default values:

```
typedef struct avs_parameters avs_parameters_t;
 #define AVS_PARAMETERS_INIT { \
 _AVS_INTERFACE_VERSION, \
 NULL, \
 1000, \
 3, \
 0, \
 -1, \
 500000L, \
 7, \
 AVS_CHARSET_LATIN1, \
 0,0}
```

- The interface version is set to _AVS_INTERFACE_VERSION.
- The license string is set to NULL.
- The ignore threshold is set to 1000 (ranking ignore).
- The minimum characters before a wildcard character is set to 3.
- The unlimited number of characters after a wildcard is set to 0. Set the value to 1 if you want more than 50 characters after a wildcard.
- The index format is set to the default.
- The cache threshold is set to 500,000 bytes for the maximum size file.
- The options are set to 7 to enable the AVS_OPTION_SEARCHSINCE, AVS_OPTION_RANKBYDATE, AVS_OPTION_SEARCHBYDATE features.
- The character set is set to Latin1.
- The maximum values for tiers and buckets is set to 0.

Index Management

You can disable the following features and thereby reduce index overhead somewhat:

- AVS_OPTION_SEARCHSINCE
- AVS_OPTION_RANKBYDATE
- AVS_OPTION_SEARCHBYDATE

Each option is represented by a bit position in the `options` element of the **avs_parameters** structure.

Index for AVSI Compatibility

To initialize the index for compatibility with an AltaVista Search Intranet V2.3 index use AVS_PARAMETERS_AVSI_COMPATIBILITY:

```
#define AVS_PARAMETERS_AVSI_COMPATIBILITY { \
_AVS_INTERFACE_VERSION, \

NULL, \
1000, \
3, \
0, \
1, \
500000L, \
0xf, \
AVS_CHARSET_LATIN1, \
0,0}
```

The only difference between AVS_PARAMETERS_AVSI_COMPATIBILITY and AVS_PARAMETERS_INIT is in the setting of the AVS_OPTIONS_AVSI_COMPATIBILITY bit in the *options* element.

Converter Structures and Parameters

AltaVista Search Developer's Kit document converter API converts various document types to text or HTML. It contains technology from Inso Corporation, Adobe Systems Incorporated and Compaq Computer Corporation. The Inso filters actually evaluate what type of file it is by opening the file and analyzing the contents (rather than by just looking at the file type). The exceptions are PDF and PostScript files. PDF files are handed off to the Adobe PDF filter and PostScript files are handed off to the Compaq PostScript filter. Use the following structure with the **avs_convert_init** procedure to set the various converter parameters:

```
 struct avs_convert_params {
    char *cvtpath;    /* Converter pathname ('dictionary' file
must be here) */
};
typedef struct avs_convert_params avs_convert_params_t;
```

Character Sets You Can Index

The AltaVista Search Developer's Kit supports the character sets at its API:

- ISO LATIN 1
- UTF8
- ASCII 8 bit

A given index must use only one of these character sets. The character set is specified in the *charset* element of the **avs_parameters** structure.

AVSI Compatibility Structures

You can now write or read data to or from an index created by the AltaVista Search Intranet (AVSI) product. The data written into an AVSI index by a Developer's Kit application can be **used in query operations** performed by users of the AVSI product using the AVSI **mhttpd** query server. Likewise, data written into an AVSI index by the AVSI indexer can be **read** by the Developer's Kit applications and used in query operations performed by these applications.

You may share an index between an SDK application and AltaVista Search Intranet V2.3 for searching, only. When writing into an AVSI index by a Developer's Kit application, the AVSI product must not be running.

The C API versions of the AVSI compatibility components are called *avsi26_mt.dll* and *libavs26_r.a* for NT and Unix, respectively. Please see the include file *avsi_compat.h* (source directory) and the sample code in *avsi_sample.c* (source/avsi_sample directory) for information on how to use the C API.

Use the following structure in conjunction with avsi_setdocdata and avsi_getdocdata procedures to read and write information about the document.

```
struct avsi_docdata {
    unsigned long szDoc; /* size of document, in bytes */
    char URL[AVS_MAX_URL_SIZE+1];           /* document's URL */
    char Title[AVS_MAX_TITLE_SIZE+1];       /* document's title */
    char Abstract[AVS_MAX_ABSTRACT_SIZE+1]; /* document's abstract
*/
    char Language[3];     /* doc's language code (e.g. "en").

    char Charset[15];     /* doc's character set./
};
typedef struct avsi_docdata avsi_docdata_t;
```

The maximum values for URL, title, and abstract sizes are as values:

- AVS_MAX_URL_SIZE - The maximum size of a URL is 1000 characters.
- AVS_MAX_ABSTRACT_SIZE - The maximum size of the document abstract is 155 characters.
- AVS_MAX_TITLE_SIZE - The maximum size of the document title is 80 characters.

The other options you can set with the Search product's document data is the document's language code and the document's character set.

The following table lists the ISO 639 2-character languages codes to use with *avsi_docdata* structure. All documents should have a language definition. Use of the **unknown** language type may result in incorrect translation of the document data. For more information see the `avsi_compat.h` file.

Code	Language	Code	Language	Code	Language
da	Danish	**de**	German	**en**	English
ar	Arabic	**bg**	Bulgarian	**el**	Greek
cs	Czech	**es**	Spanish	**et**	Estonian
fi	Finnish	**fr**	French	**hu**	Hungarian
is	Icelandic	**it**	Italian	**ja**	Japanese
ko	Korean	**lt**	Lithuanian	**lv**	Latvian
nl	Dutch	**no**	Norwegian	**pt**	Portuguese
ro	Romanian	**ru**	Russian	**sl**	Slovenian
sv	Swedish	**th**	Currently empty	**tr**	Turkish
zh	Chinese	**pl**	Polish	**ne**	Niger

The character set encodings are contained in the following table to use in the *avsi_docdata* structure. All documents should have a defined character set. use of the **unknown** character set may result in incorrect translation (no translation) of the document data.

Character Set		Character Set		Character Set	
iso88591	Western (ISO-8859-1)	**iso88592**	Central European (ISO-8859-2)	**iso88593**	(ISO-8859-3)
iso88594	(ISO-8859-4)	**iso88595**	Cyrillic (ISO-8859-5)	**iso88596**	Arabic (ISO-8859-6)
iso88597	Greek (ISO-8859-7)	**iso88598**	Hebrew (ISO-8859-8)	**iso88599**	Turkish
iso885910	Latin 6 - Lappish/Eskimo/Nordic languages	**koi8r**	Cyrillic (KOI8-R)	**ascii**	ASCII
jis	Japanese	**sjis**	Japanese (Shift-JIS)	**euc**	Japanese
gb	Simplified Chinese	**big5**	Traditional Chinese (Big5)	**euctw**	Traditional Chinese
euccn	Chinese (GB)	**eucjp**	Japanese (EUC)	**euckr**	Korean (KSC)
utf8	Unicode	**hz**	Mixed Chinese and ASCII characters	**cp1250**	Central European (Windows-1250)
cp1251	Cyrillic (Windows-1251)	**cp1252**	Western (Windows-1252)	**cp1253**	Greek (Windows-1253)
cp1254	Arabic (Windows-1254)	**cp1255**	Hebrew (Windows-1255)	**cp1256**	Arabic
cp1257	Baltic (Windows-1257)	**cp1258**	Vietnamese (Windows-1258)	**jis0212**	Japanese

At this time ISO88599, ISO885910, JIS, EUC, GB, EUCTW, UTF8, HZ, CP1256, JIS0212. are not supported in the AVSI product.

Filter Procedure

The **avs_newdoc** filter procedure must match the following filter prototype:

```
typedef int (*filter_p_t)
       (
       IN avs_idxHdl_t,
       IN void *,
       IN unsigned long,
       OUT unsigned long *
       );
```

6

Visual Basic Reference Section

The AltaVista Search Developer's Kit includes an ActiveX component that greatly simplifies development of applications using Visual Basic. The kit includes the following files:

- avs26X_mt.dll for the AVSIndex Class.
- avscvt26.dll for AVSDocument Class.

It is assumed that you have Visual Basic Version 5.0 installed on your system and are running on Windows NT Version 4.0.

Naming Conventions

The naming conventions of various index properties and search results properties are:

- iopt - Index options
- sopt -Search options
- sres - Search results
- cres - Counts results

Class AvsIndex

The AVSIndex object contains the index methods and properties.

adddate function

Adds an additional date to the document.

```
Function adddate(year As Long, month As Long, day As Long,
startloc As Long) As Long
```

Argument

year	Integer that specifies the year.
month	Integer that specifies the month.
Day	Integer that specifies the day.
startloc	Location in the document for the date.

Description

The adddate function indexes the supplied date in standard format at the indicated location. Applications can submit advanced queries for specific dates or date ranges contained in a field using the format fied:[date range]. This function can also be used to associate multiple dates with a document.

Return Value

Returns 0 or an error code.

See Also

- addfield
- addliteral
- addvalue
- addword
- setdocdate
- setdocdatetime

addfield function

Adds a field to document

```
Function addfield(fieldName As String, startloc As Long, endloc
As Long)As Long
```

Argument

fieldName	String that specifies the name of the field to be added to the index.
Startloc	Location of the first word in the field.
endloc	Location of the first word that follows the field.

Description

The addfield function marks a set of locations in a document as belonging to a field. The startloc and endloc arguments define the boundaries of the field. Application users can submit queries for specific contents of the named field, using the format fieldname:value.

Return Value

Returns 0 or an error code.

See Also

- adddate
- addliteral
- addvalue
- addword

addliteral function

Adds a literal string to the document.

```
Function addliteral(word As String, startloc As Long) As Long
```

Argument

word	Single string that specifies the word to add.
startloc	Location in the index of where to add the string to the index.

Description

The addliteral function adds a single word and without interpretation or conversion to a document index.This function does not scan the literal string in any way, but rather adds it to the index as is. Use with caution or not at all, as words added this way are possibly not searchable with the standard query procedures.

To perform a search when the literal string you are looking for contains special characters (for example, the forward slash (/)), you can use curly braces({}) in the Boolean (advanced) query string as in the following example: {cnn/xyz}. All characters between the matching curly braces are treated as a word, except the asterisk (*) which still works as a wildcard.

Note: Words starting with a non-alphanumeric character are reserved for internal use.

Return Value

Returns 0 or an error code.

See Also

- adddate
- addfield
- addliteral
- addword

addvalue function

Adds a searchable value to the document.

```
Function addvalue(type_name As String, value As Long, startloc
As Long)As Long
```

Argument

type_name	Name of a value type defined previously with define_valtype.
value	The value (integer) of the type to be added to the index.
startloc	The location of the type in the index.

Description

The addvalue function indexes the supplied value at the specified location in the index. Value types are defined with the define_valtype function. The addvalue function allows the value to be searched in a Boolean query expression, for example, [lines:1-100].

Return Value

Returns 0 or an error code.

See Also

- adddate
- addfield
- addword
- define_valtype
- setrankval

addword function

Adds a word or words to a document.

```
Function addword(words As String, startloc As Long) As Long
```

Argument

words	The words to add to the index.
startloc	Location value to assign to the first word.

Description

The addword function adds words to the index. A word is defined as a contiguous string of alphanumerics, bounded by non-alphanumerics (like spaces and special characters), as defined in the ISO Latin-1 standard.

This function should be called after startdoc. Call the addword_numwords property to get the number of words added to the index.

Return Value

Returns 0 or an error code.

See Also

- adddate
- addfield
- addliteral
- addvalue
- addword_numwords

addword_numwords property

Returns the number of words added.

```
Property addword_numwords As Long
read-only
```

Description

The addword_numwords property returns the number of words added to the index by the addword function.

Return Value

Returns the number of words added to the index.

See Also

addword

avs_version property

Returns the version number of the Developer's Kit.

```
Property avs_version As String
read-only
```

Description

The avs_version function returns a list of index version strings. Each string in the list is enclosed in double quotes (""). The list contains information relevant to the product version, such as, the implementation version, the interface version, whether it is single or multi-threaded, the build number, and the license version.

Return Value

Returns list of index version strings.

buildmode function

Sets the index to buildmode.

```
Function buildmode(nTiers As Long) As Long
```

Argument

nTiers	The number of tiers (set of buckets) the index is allowed to use.

Description

The buildmode function optimizes the specified index for building or adding to the index. Querying during this state would degrade the response to the query. This function could be called from your application for those instances when you want to build an index and users would be unlikely to query the index. The new mode takes effect immediately. This state of the index will be retained for the next time the index is opened, if a call to makestable or compact is made.

Return Value

Returns 0 on success or an error code.

See Also

- compact
- makestable

close function

Closes the index.

```
Function close ( ) As Long
```

Description

The close function closes the specified index and releases all resources. Make sure you have closed any outstanding search or counts handles before closing the index.

Return Value

Returns 0 or an error code.

See Also

- makestable
- open

compact function

Compacts an index.

Function compact( ) As Long

Description

The compact function causes one or more levels of compaction on the index. If the returned value of a call to compact_moreneeded is 0, the compaction is complete. If the returned value is 1, call the compact function again to further compact the index.

You should compact the index periodically or after a series of updates is complete. Compacting the index improves subsequent query performance and frees the space used by documents that have been deleted. It is possible to submit search queries to the index (in other threads) while it is being compacted, but you cannot add, update, or delete information until compaction is complete.

Return Value

Returns 0 or an error code.

See Also

- compact
- compact_moreneeded
- makestable

compact_minor function

Performs minor index compaction.

```
Function compact_minor ( ) As Long
```

Description

The compact_minor function causes one or more levels of compaction on the index but without recovering space from deleted index entries. Use this function when the effects of regular compaction would be detrimental to your system resources. If a call to compact_moreneeded returns a value of 0, the compaction is complete. If compact_moreneeded returns 1, call the compact_minor function again to further compact the index.

You should compact the index periodically or after a series of updates is complete. Compacting the index improves subsequent query performance. It is possible to submit search queries to the index while it is being compacted, but you cannot add, update, or delete information until compaction is complete.

Return Value

Returns 0 or an error code.

See Also

- compact
- compact_moreneeded
- makestable

compact_moreneeded property

Returns a value which specifies if more compaction is needed.

```
Property compact_moreneeded As Long
read-only
```

Description

The compact_moreneeded property returns a value of 0 or 1. If the return is zero (0), there is no further compaction required on the index. If the return is 1, then more compaction is required.

Use this property in conjunction with compact and compact_minor functions.

Return Value

Returns a non-zero value if the index requires further compaction.

See Also

- compact
- compact_minor
- makestable

compactionneeded function

Returns a compaction needed status.

```
Function compactionneeded( ) As Long
```

Description

The compactionneeded function returns non-zero value if the index needs compaction.

Return Value

Returns non-zero value if the index needs compaction.

See Also

- compact
- compact_minor
- compact_moreneeded
- makestable

count function

Counts word occurrences.

```
Function count(WordPrefix As String) As Long
```

Argument

WordPrefix	The value of a word or portion of a word. All words that begin with this character string are returned, one at a time, through the countnext function.

Description

The count function is used in conjunction with cres_countnext to enumerate index entries that match the specified word or prefix.

The count is not adjusted for deletions if they have occurred since the last time the index has been compacted.

To enumerate the entire contents of the index, use a null string ("")for the WordPrefix argument. This function will return a count handle for all index entries including those that have been deleted.

The count function returns the counts handle or a -1 if there is an error. To get the error, call lasterror.

Return Value

Returns the counts handle or -1.

See Also

- count_close
- cres_countnext
- cres_word
- cres_wordcount
- lasterror

count_close function

Terminates a count.

```
Function count_close(counthandle As Long) As Long
```

Argument

counthandle	The value of the counthandle.

Description

The count_close function closes a count request. After all calls to countnext are complete, call count_close to release the resources allocated for the count.

Return Value

Returns 0 or an error code.

See Also

- count
- cres_countnext
- cres_word
- cres_wordcount

cres_countnext function

Retrieves the next word occurrence.

```
Function cres_countnext(counthandle As Long) As Long
```

Argument

counthandle	The value of the counts handle from count.

Description

The cres_countnext function retrieves the next (or first) index entry that matches the prefix specified in the count function. The properties cres_word and cres_wordcount return the actual word and the number of times it occurs in the index, respectively. The function returns a non-zero status code there are no more word occurrences to retrieve.

Return Value

Returns a non-zero status code when there are no more word occurrences to retrieve.

See Also

- count
- count_close
- cres_word
- cres_wordcount

cres_word property

Retrieves a word.

```
Property cres_word(counthandle As Long) As String
read-only
```

Argument

counthandle	The value of the counts handle from the counts method.

Description

The cres_word property retrieves the word corresponding to the most recent call to cres_countnext.

Return Value

Returns a word.

See Also

- count
- count_close
- cres_countnext
- cres_wordcount

cres_wordcount property

Retrieves the word count.

```
Property cres_wordcount(counthandle As Long) As Long
read-only
```

Argument

counthandle	Value of the counts handle from the counts method.

Description

The cres_wordcount property retrieves the word count corresponding to the most recent call to cres_countnext.

Return Value

Returns the word count.

See Also

- count
- count_close
- cres_countnext
- cres_word

define_valtype function

Defines a value type for ranking purposes.

```
Function define_valtype(name As String, minval As Long, maxval
As Long) As Long
```

Argument

name	String that contains the name of the value type.
minval	Long with the minimum value of the value type.
maxval	Long with the maximum value of the value type.

Description

The define_valtype function lets you define your own value type which can be used to rank search results. For example, you may define the value type lines to count the number of lines per document. With the type name, you also must supply the lowest to the highest possible values. In your application, use a call to setrankval function to set a ranking value for each document in the index. To index the new type, call the addvalue function.

To use the value type to rank search results, use search_genrank function and pass the value type in the RankTerms argument, for example, lines. To have the search results ranked in reverse order (lowest value first), use -lines. .

To use the value type in a Boolean search, use the search function. In the BoolQuery argument, use the syntax: [valtype:range], for example, [lines:1-500].

The define_valtype function is an application-wide function and, therefore, affects all the indexes that are open. You must call define_valtype before you open the index. Call the release_valtypes function after the last call to close to release the resources associated with the value types.

Return Value

Returns 0 or an error code.

See Also

- addvalue
- setrankval

define_valtype_multiple function

Defines a value type for filtering on multiple values.

```
Function define_valtype_multiple(name As String, minval As
Long, maxval As Long, numvalues As
Long) As Long
```

Argument

Description

name	String that contains the name of the value type.
minval	Long with the minimum value of the value type.
maxval	Long with the maximum value of the value type.
numvalues	Maximum number of multiple filtering values.

Description

The define_valtype_multiple function lets you define your own value type to add a set of values to a document.

Subsequently, these values can be used to filter search results. With the type name, you also must supply the minimum to maximum range possible for values. The numvalues parameter determines the maximum number of multiple values allowed for the valtype for each document.

In your filter application, use multiple calls to the setrankval function to set a value for each document in the index.

To use the value type to filter search results, use search_genrank function and use the value type in the RankTerms argument. For example, [myval?(1,5)] filters the search results to those documents that contain a value of 1 or 5 in the myval valtype.

The define_valtype_multiple function is an application-wide function and, therefore, effects all the indexes that are open. You must call define_valtype_multiple before you open the index. Call the release_valtypes function after the last call to close to release the resources associated with the value types.

Return Value

Returns 0 or an error code.

See Also

- addvalue
- define_valtype
- setrankval

deletedoc function

Deletes a document with the specified document identifier.

```
Function deletedoc(docId As String) As Long
```

Argument

docId	Case-sensitive string that identifies the document (limited to 120 bytes).

Description

The deletedoc function marks for deletion all documents with a specified docid (if any exist). For the documents to actually be deleted you must call the makestable function. If you insert the call to makestable immediately after the deletedoc function, the deletion will occur immediately. To retrieve the number of documents deleted, call deletedoc_numdeleted.

Note: The docId string is case sensitive.

Return Value

Returns 0 or an error code.

See Also

- compact
- deletedoc_numdeleted
- makestable

deletedoc_numdeleted

Returns the number of deleted documents.

```
Property deletedoc_numdeleted As Long
read-only
```

Description

The deletedoc_numdeleted property returns the number of documents deleted by deletedoc. Use this property in conjunction with the function deletedoc

Return Value

Returns the number of documents which were deleted by deletedoc.

See Also

- deletedoc

enddoc function

Terminates a document added with startdoc.

```
Function enddoc ( ) As Long
```

Description

The enddoc function terminates a document created in the index by the startdoc function. Document contents arehadded to the index by a call to startdoc and terminated by the enddoc function.

Return Value

Returns 0 or an error code.

See Also

- startdoc

errmsg function

Converts an error code to a string.

```
Function errmsg(status As Long) As String
```

Argument

status	The error code returned from AvsIndex functions.

Description

The errmsg function returns a text message associated with an error status code.

Return Value

Returns an error message text.

getindexmode property

Retrieves the current index mode.

```
Property getindexmode As Long
read-only
```

Description

The getindexmode property returns 0 if the index is in query mode, or 1 if the index is in build mode. See Optimizing for speed to learn about build and query mode.

Return Value

Returns 0 or 1.

See Also

- buildmode
- compact

indexversion property

Retrieves the index version number.

```
Property indexversion As Long
read-only
```

Description

The indexversion property is used to return an integer value corresponding to the current version of the index. This value increases with each call to makestable, compact, or compact_minor.

Return Value

Returns index version number.

iopt_cache_threshold property

Determines the cache threshold.

```
Property iopt_cache_threshold As Long
```

Description

The iopt_cache_threshold property determines the maximum size of the index file that will be memory-mapped during an indexing process. The larger values will cause better performance but require larger amounts of virtual memory to be available. The default value is 500000 (0.5 MB).

Return Value

None, this is a write-only property.

See Also

- iopt_ignored_threshold

iopt_chars_before_wildcard property

Determines the number of characters required before a wildcard search.

```
Property iopt_chars_before_wildcard As Integer
```

Description

The iopt_chars_before_wildcard property provides the ability to change the number of characters before the wildcard (*) from the default of 3. The number can be zero (0) or greater.

Return Value

None, this is a write-only property.

iopt_charset property

Sets the character set for the index.

```
Property iopt_charset As Integer
```

Description

The iopt_charset property provides the ability to change the character set from the default set of ISO Latin 1 to UTF8 or ASCII 8 . The values of the respective character sets are:

ISO Latin 1	0
UTF8	1
ASCII 8	2

Return Value

None, this is a write-only property.

iopt_enable_rankbydate property

Enables or disables the rank-by-date feature.

```
Property iopt_enable_rankbydate As Integer
```

Description

The iopt_enable_rankbydate property enables the search method to rank the results using the date of the document.

By default, this property is enabled and effects all searches performed on the index. If the integer is zero (0), this feature is disabled.

Return Value

Returns nothing.

See Also

- iopt_enable_searchbydate

iopt_enable_searchbydate property

Enables or disables the search-by-date feature.

```
Property iopt_enable_searchbydate As Integer
```

Description

The iopt_enable_searchbydate property enables the search method to return documents that match the specified date of the search criteria. By default, this property is enabled and effects all searches performed on the index. If the integer is zero (0), the iopt_enable_searchbydate property is disabled.

Return Value

Returns nothing.

See Also

- sopt_rank_to_boolean

iopt_enable_searchsince property

Enables or disables the search-since feature.

```
Property iopt_enable_searchsince As Integer
```

Description

The iopt_enable_searchsince property enables the search method to return documents added since the last search operation. By default, this property is enabled and effects all searches performed on the index. If the integer is zero (0), the iopt_enable_searchsince property is disabled.

Return Value

Returns nothing.

See Also

- iopt_enable_rankbydate
- iopt_enable_searchbydate
- search
- search_genrank
- sopt_rank_to_boolean

iopt_ignored_threshold property

Sets the ignored-threshold feature.

```
Property iopt_ignored_threshold As Long
```

Description

The iopt_ignored_threshold property is the number given in one hundredths of a percent (for example, 1000 is 10 percent) that controls when a ranking term is to be ignored. Any ranking terms whose occurrences in the index account for a greater percentage of the total percentages than this number is ignored for ranking purposes. The default value is 1000 or 10 percent.

Return Value

Returns nothing.

See Also

- iopt_cache_threshold

iopt_indexformat property

Sets the current index format version number.

```
Property iopt_indexformat As Integer
```

Description

The iopt_indexformat property sets the value of the current index format as follows:

Value	Description
0	Default (now Version 2)
1	Version 1
2	Version 2

The Version 2 index format generates a slightly larger index (approximately 10% larger), but it is slightly faster to search than in index an Version 1 format.

Return Value

None.

iopt_nbuckets property

Sets the number of buckets used by the index.

```
Property iopt_nbuckets As Integer
```

Description

The iopt_nbuckets property determines the maximum number of buckets the index is allowed to use. Buckets are the hash modulus for splitting the index by word. The value of this property determines the number of the index files across which index entries are spread (by hashing). If this number is increased, the number of index files is increased but the size of the individual files should be smaller. The maximum number of buckets allowed is 500 and is set in the avs.h file.

The nbuckets and ntiers are index scaling properties that you can use to tune your system's memory configuration parameters.

Tiers are the sets of buckets the index is allowed to use. The tiers parameter value determines the maximum number of sets of buckets to which the index can grow during operations that add, delete, or update the index. Each call to avs_makestable creates a new tier in the index, and calls to avs_compact or avs_compact_minor are then used to reduce the tiers again.

When building a very large index, it is better to have a larger value as it reduces the number of compactions needed during building. However, query operations may take longer when more tiers are in use (more index files to examine for each index entry). The default values of tiers can be tuned with a nominal range of values from 4 - 40. Smaller values are appropriate for more searching, while the larger values are appropriate for more indexing.

Return Value

Returns nothing.

See Also

- iopt_ntiers

iopt_ntiers property

Sets the number of tiers used by the index.

```
Property iopt_ntiers As Integer
```

Description

The iopt_ntiers property determines the maximum number of sets of buckets to which the index can grow during operations that add, delete, or update the index. The nbuckets and ntiers are index scaling properties that you can use to tune your system's memory configuration parameters. Each call to avs_makestable creates a new tier in the index, and calls to avs_compact or avs_compact_minor are then used to reduce the tiers again. The default values can be tuned with a nominal range of values from 4 - 40. Smaller values are appropriate for more searching, while the larger values are appropriate for more indexing.

Buckets are the hash modulus for splitting the index by word. The value of this property determines the number of the index files across which index entries are spread (by hashing). If this number is increased, the number of index files is increased but the size of the individual files should be smaller. The maximum number of buckets allowed is 500 and is set in the avs.h file.

When building a very large index, it is better to have a larger value as it reduces the number of compactions needed during building. However, query operations may take longer when more tiers are in use (more index files to examine for each index entry).

Return Value

Returns nothing.

See Also

- iopt_nbuckets

iopt_parsesgml property

Parses SGML tags during indexing.

```
Property iopt_parsesgml as Integer
```

Description

The iopt_parsesgml property enables or disables the parsing of SGML tags during indexing. Currently, either zero (0) or 1 is the only valid value for this property. When the flag is set to 1, it enables the indexer to recognize SGML encoded entities as characters and add words containing those characters to the index. For example, é is the SGML encoding for e with the acute accent.

Return Value

Returns nothing.

iopt_unlimited_wild_words property

Sets the number of words returned by wildcard searches to unlimited.

```
Property iopt_unlimited_wild_words As Integer
```

Description

The iopt_unlimited_wild_words property sets the number of words returned by a wild card search to unlimited. The default value is 50 words matching the query. If more than 50 words match the query, the results are ranked by frequency. This does not effect the ranking done by a Boolean search.

Return Value

Returns nothing.

See Also

- iopt_chars_before_wildcard

lasterror property

Returns the last error from search, counts, or search_genrank.

```
Property lasterror As Long
read only
```

Description

The lasterror property returns the last error from the search, counts or search_genrank functions. This is usually an error related to an index, search, or counts handle. If the search or counts handle is -1, use this property to determine the error status.

Return Value

Returns an error status.

See Also

- addword

makestable function

Makes the index stable.

```
Function makestable( ) As Long
```

Description

The makestable function saves recent deletions or additions made to the index file. The documents and words added to the index are flushed to the disk, and all documents marked for deletion by a previous call to deletedoc, are deleted. This function should be called before closing the index and after every 500,000 or so words have been indexed.

The makestable function finishes the job of adding and deleting documents and makes the newly added documents searchable while removing the documents marked for deletion by the deletedoc function.

The application should also compact the index after a series of updates is completed, to improve subsequent query performance and to recover space from deleted entries.

Return Value

Returns 0 or an error code.

See Also

- compact
- compact_minor
- sopt_rank_to_boolean

open function

Opens an index.

```
Function open(path As String, mode As String) As Long
```

Argument

path	The full or relative path of the index directory on disk.
mode	Specifies whether the index should be opened as a read only (r) or a read write (rw) index.

Description

The open function opens the index pointed to by the path argument. The mode parameter can be either "r" for read only, "rw" for read-write, and "ro" for CD-ROM. When "rw" is set, the index is opened for reading and writing. Appropriate locks are set to interlock reads and updates in other processes.

Note: The directory specified by path must exist. If the directory specified by path is empty, a new index will be created with the correct permissions.

Return Value

Returns 0 or an error code.

See Also

- close

querymode

Sets an index to query mode.

```
Function querymode( ) As Long
```

Description

The querymode function optimizes an index for response to user queries. This allows users to get optimal response in querying the index when use of the query interface is high. The new mode takes effect immediately. To retain this state for the next time the index is opened, you must make a call to the makestable or compact functions.

When your program calls this function, it causes a full compaction of the index.

Return Value

Returns 0 or an error code.

See Also

- buildmode

release_valtypes function

Releases any value types have been defined.

```
Function release_valtypes( ) As Long
```

Description

The release_valtypes function releases any value types that have been defined. This function should only be called after your application's last call to close.

Return Value

Returns 0 or an error code.

See Also

- define_valtype

search function

Searches the index.

```
Function search(SimpleQuery As String, BoolQuery As String,
 SearchSince As String) As Long
```

Argument

SimpleQuery	Simple (or ranking) query expression. May be NULL.
BoolQuery	A boolean query expression. May be NULL.
SearchSince	Version string or NULL.

Description

The search function initializes a search for documents in an index file that match a simple query expression and the given search parameters. A simple query expression can contain words, phrases, the asterisk (*) wildcard character, and the + and - operators, which require or prohibit the presence of a word in the search results.

Given a simple query expression and other search parameters (like date ranges, boolean qualifier, and so forth) this function searches the index and returns a handle which can be used to retrieve:

- the number of documents found
- the number of documents actually returned for inspection
- specific search results corresponding to a document
- the matching terms from the query expression

It is possible to use this interface to perform consistently with either the AltaVista simple search or the AltaVista advanced search function (as seen at http://altavista.digital.com). You can also effectively combine both approaches in one call.

To Perform a Simple Search

Use the pQuery argument to point to the simple query string. Do not use the pBoolean argument. Set the sopt_ranktoboolean property to 1.

To Perform an Advanced Search

Use the pQuery argument to point to ranking terms. Use the pBoolean argument to point to the advanced query string. Set the sopt_ranktoboolean property to 0.

The Combined Approach -- Simple Query with Boolean Qualifiers

Use the pQuery argument to point to the simple query string and use the pBoolean argument to point to the advanced query string. Set the sopt_ranktoboolean property flag to 1. In this case, the behavior is that of the simple query, filtered by the boolean expression, that is, the results are the intersection of what would be produced by the simple query and the boolean query separately.

The search function returns the search handle as the value of the method or a -1 for an error. To get the error, call lasterror. For more information, see Searching the Index. In all cases, the results are ranked according to the relative weighted occurence of the (non-negative) terms in the simple query (or ranking) string. If there are no such terms, the result is unranked.

Return Value

Returns the search handle as the value of the method or -1.

See Also

- search_genrank
- search_getterms
- sres_day
- sres_docdata
- sres_docid
- sres_docsfound
- sres_docsreturned
- sres_month
- sres_relevance
- sres_searchversion
- sres_termcount
- sres_year

search_close function

Terminates a search.

```
Function search_close(searchhandle As Long) As Long
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The search_close function closes a search. This must be called when all calls to search_getresults are completed, to release the resources allocated for a search.

Return Value

Returns 0 or an error code.

See Also

- search
- search_genrank

search_genrank function

Searches an index and ranks the results.

```
Function search_genrank(BoolQuery As String, RankTerms As
String, SearchSince As String) As Long
```

Argument

BoolQuery	A Boolean query expression or NULL.
RankTerms	A string containing the ranking terms.
SearchSince	A search version string.

Description

The search_genrank function enables you search for the extended value types added by the addvalue function. This function lets you search for the boolean query expression and rank the results using the ranking expression. The RankTerms argument in the search_genrank function can be either a predefined term (for example, #date) or an application-defined value type (for example, lines). To rank the results in reverse order, that is, lowest value first, precede the term with a minus sign (-), for example, -#date or -lines.

The search_genrank function returns the search handle or -1 if there is an error. To get the error, call lasterror.

Return Value

Returns the search handle as the value of the method or -1.

See Also

- lasterror
- search
- search_close

search_getresults function

Gets the search results.

```
Function search_getresults(searchhandle As Long, resultnum As
Long) As Long
```

Argument

searchhandle	The search handle from search or search_genrank.
resultnum	The number of results.

Description

The search_getresults function is used to retrieve specific search results after calling the search or search_genrank functions. An ordinal value specifies which result to retrieve. This ordinal must be a value between 0 and the number of documents returned by sres_docsreturned property minus 1. This function retrieves various document attributes, such as, relevancy value, document date, document identifier, and document data and makes the attributes' value available through the search results handle.

Return Value

Returns the search results.

See Also

- search
- search_close
- search_genrank

search_getterms function

Retrieves matching search terms from the query.

```
Function search_getterms(searchhandle As Long, resultnum As
Long) As Long
```

Argument

searchhandle	The search handle from search or search_genrank.
resultnum	The number of results.

Description

The search_getterms function is used to retrieve the terms and term statistics for the specified search from search or search_genrank functions.

The term to retrieve is the 0-relative result number specified in the sres_numterms property. Call sres_terms to retrieve the term string. Use sres_termcount to determine the number of matches of the term string.

Return Value

Returns the search terms.

See Also

- search
- search_close
- search_genrank
- sres_numterms
- sres_termcount

setdocdatastr function

Sets the document data.

```
Function setdocdatastr(docData As String) As Long
```

Argument

docData	The string containing the document data.

Description

The setdocdatastr function sets the document's data (for example, the title of the document or other descriptive information) Call this function after analyzing the document's content and between startdoc and enddoc functions. The data is made available by a successful search. The maximum length of the document data string is 64,000 bytes.

Return Value

Returns 0 or an error code.

See Also

- enddoc
- startdoc

setdocdate function

Sets the document date.

```
Function setdocdate(year As Long, month As Long, day As Long)
As Long
```

Argument

year	The year greater than 0100 but less than 2148.
month	The month of the year from 1 to 12.
day	The day of the month from 1 to 31.

Description

The setdocdate function sets the date of the document and is called between the startdoc and enddoc functions.

The date is returned in the search results and can be retrieved by calling sres_day, sres_month, and sres_year on the search results handle. Dates are indexed and can be used to limit searches by adding a date range as an additional term in the boolean query string argument passed to search.

The date and time (see setdocdatetime)can also be used to order search results. For further information see search_genrank.

Return Value

Returns 0 or an error code.

See Also

- setdocdatetime

setdocdatetime function

Sets the document date and time.

```
Function setdocdatetime(year As Long, month As Long,
day As Long, hour As Long, minute As Long, second As Long)
As Long
```

Argument

year	The year greater than 0100 but less than 2148.
month	The month of the year from 1 to 12.
day	The day of the month from 1 to 31.
hour	The hour from 0 to 23.
minute	The minute from 0 to 59.
second	The seconds from 0 to 59.

Description

The setdocdatetime function sets the date and time of the document and is called between the startdoc and enddoc functions..

The date is returned in the search results and can be retrieved by calling sres_day, sres_month, and sres_year on the search results handle. Dates are indexed and can be used to limit searches by adding a date range as an additional term in the boolean query string argument passed to search. Dates can also be used to rank search results, see search_genrank for more details.

This function provides a higher degree of precision in the date assigned to a document (to 1 second) than setdocdate.

Return Value

Returns 0 or an error code.

See Also

- setdocdate

setrankval function

Adds a ranking value to the document.

```
Function setrankval(type_name As String, value As Long) As Long
```

Argument

type_name	Name of the value type to be added.
Value	Integer value for the type.

Description

The setrankval function adds a numeric value to a document index that can be used for ranking. A given value type should be used at most once in a document. To use the value as a ranking term in a search, see the search_genrank function

Return Value

Returns 0 or an error code.

See Also

- addvalue
- define_valtype

sopt_doclimit property

Sets the maximum number of documents to return from a search.

```
Property sopt_doclimit As Long
```

Description

The sopt_doclimit property is a write-only property which limits the number of documents that can be returned in a search or search_genrank result. The default value for the maximum number of documents returned is 32,000.

Return Value

Returns nothing.

See Also

- sopt_rank_to_boolean
- sres_docdata
- sres_docid

sopt_rank_to_boolean property

Sets the rank to boolean option.

```
Property sopt_rank_to_boolean As Integer
```

Description

The sopt_rank_to_boolean property when the value is non-zero directs the subsequent call to the search method to limit the search to documents which match the Boolean expression and have at least one of the ranking terms. By default, this call is enabled. Typically, this property reduces the number of documents found.

Return Value

Returns nothing.

See Also

- sopt_doclimit

sres_day property

Returns the day of the document date.

```
Property sres_day(searchhandle As Long) As Integer
read-only
```

Argument

searchhandle	Search handle for search or search_genrank.

Description

The sres_day function retrieves the day of the year value of a document's date. Call this function after calling search_getresults.

Return Value

Returns the day of the year value.

See Also

- sres_month
- sres_year

sres_docdata property

Retrieves the document data string from the search results.

```
Property sres_docdata(searchhandle As Long) As String
read-only
```

Argument

Searchhandle	The search handle from search or search_genrank

Description

The sres_docdata property retrieves the document data string from the search results.

Return Value

Returns the document data string.

See Also

- search_getresults
- sres_docid

sres_docid property

Retrieves the document identifier.

```
Property sres_docid(searchhandle As Long) As String
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_docid property retrieves the document identifier from the search result.

Return Value

Returns the document indentifier string.

See Also

- sres_docdata
- sopt_doclimit

sres_docsfound property

Retrieves the number of documents found from a search.

```
Property sres_docsfound(searchhandle As Long) As Long
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_docsfound property retrieves the number of documents found after a search.

Return Value

Returns the number of documents found.

See Also

- sres_docsreturned

sres_docsreturned property

Retrieves the number of documents returned.

```
Property sres_docsreturned(searchhandle As Long) As Long
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_docsreturned property retrieves the number of documents returned from a search.

Return Value

Returns the number of documents returned.

See Also

- sopt_doclimit
- sres_docsfound

sres_month property

Returns the month of the year associated with the document.

```
Property sres_month(searchhandle As Long) As Integer
read-only
```

Argument

searchhandle	Search handle from search or search_genrank.

Description

The sres_month function retrieves the month of the year value of a document's date. Call this function after calling search_getresults.

Return Value

Returns the month of the year value.

See Also

- search_getresults
- sres_day
- sres_year

sres_numterms property

Returns the number of matching terms from a search.

```
Property sres_numterms(searchhandle As Long) As Long
read-only
```

Argument

searchhandle	The search handle from a search.

Description

The sres_numterms property returns the number of matching terms found from a search. Call this property after a call to search. Note: This property is not valid for search_genrank.

Return Value

Returns the number of matching terms.

See Also

- search_close
- sres_term
- sres_termcount

sres_relevance property

Returns the relevance value of search result.

```
Property sres_relevance(searchhandle As Long) As Single
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_relevance property retrieves the relevancy value of a document returned with search_getresults.

Return Value

Returns the relevancy value.

See Also

- search_getresults

sres_searchversion property

Returns the version string of the current search.

```
Property sres_searchversion(searchhandle As Long) As String
read-only
```

Argument

searchhandle	Required. Always the Err object.

Description

The sres_searchversion property retrieves the version number of the current search. This version number can be stored for later use in the SearchSince argument of the search and search_genrank functions.

Return Value

Returns a version string.

See Also

- search_close
- search_genrank

sres_term property

Returns the nth matching term from the query.

```
Property sres_term(searchhandle As Long) As String
read-only
```

Argument

searchhandle	Search handle from a search.

Description

The sres_term property retrieves the nth matching term of the query.

Return Value

Returns a string.

See Also

- search_close
- search_genrank

sres_termcount property

Returns the term count of the nth matching term.

```
Property sres_termcount(searchhandle As Long) As Long
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_termcount property retrieves the number of matches of the nth matching term. This property should be used after a call to search, and in conjunction with search_getterms and sres_term to retrieve the terms and term counts.

Return Value

Returns the term count of the nth matching term.

See Also

- search_getterms
- sres_docsfound
- sres_docsreturned
- sres_term

sres_year property

Returns the year value of the document date.

```
Property sres_year(searchhandle As Long) As Integer
read-only
```

Argument

searchhandle	The search handle from search or search_genrank.

Description

The sres_year property retrieves the year value of a document's date. Call this function after calling sres_getsearchresults.

Return Value

Returns the year value.

See Also

- search_getterms
- sres_docsfound
- sres_day
- sres_docsreturned
- sres_month
- sres_term

startdoc function

Adds a document to the index.

```
Function startdoc(docid As String, flags As Long) As Long
```

Argument

docid	String that names the document (limited to 120 bytes).
Flags	Sets the conditions for creating a new document. The conditions can have the following flags: **0** Does not matter whether the document already exists. If it does not exist, create it. If it does exist, replace it. **1** New document. The docid must not already exist. **2** Replace an existing document. **4** Duplicate document IDs are allowed. If a document with the same ID already exists, another one can also be created.

Description

The startdoc function creates a new document in the index. This function must be used with a call to enddoc to bracket the beginning and end of the document to be added to the index.

The first location available in the index for the document is returned through a call to startdoc_startloc property. Use this in the first call to the addword function or similar kinds on functions. When you are finished adding document contents, call the enddoc function to terminate the document.

Return Value

Returns 0 or an error code.

See Also

- adddate
- addfield
- addliteral
- addvalue
- addword
- addword
- enddoc
- setdocdatastr
- setdocdate
- setrankval

startdoc_startloc property

Returns the starting location of the document in the index.

```
Property startdoc_startloc As Long
read-only
```

Description

The startdoc_startloc property returns the starting location of the document in the index. This property is used in conjunction with startdoc and addword.

Return Value

Returns the starting location of the document.

See Also

- addword
- search_getresults
- sres_day
- sres_month
- startdoc

AVSIndex Constants

Constants are meaningful names that take the place of a numbers or strings, and remain fixed. The following are constants or global variables for the AVSIndex Class:

Constant	Description
avs_adddoc_io_err	I/O error in ni2_index_adddoclist
avs_badargs_err	Invalid arguments (for example, null pointer)
avs_compact_io_err	I/O error in ni2_index_compact
avs_counts_err	Counts object has no counts context (VB, C++ API)
avs_cvt_err	Document converter error
avs_cvt_unsuptype	Unsupported document type conversion.
avs_date_err	Date out of range
avs_doc_exists	Document already exists
avs_doc_limit_err	<= 0 documents specified
avs_doc_notfound	Document not found (locate)
avs_docdata_err	Document data too long
avs_docid_err	Document identification too long
avs_doclist_err	Error creating doclist
avs_field_err	Field processing error
avs_fileio_err	Converter: file I/O error
avs_filter_err	Filter error
avs_getdata_err	Could not get data (ni2_index_getdata)
avs_index_err	Index object has no context (VB, C++ API)
avs_license_expired	Evaluation or beta license expired
avs_lock_err	Cannot acquire index locks
avs_malloc_err	Cannot allocate memory
avs_mkstable_io_err	I/O error in ni2_index_makestable
avs_mkvis_io_err	I/O error in ni2_index_makevisible
avs_nametoolong_err	Value type name too long

Constant	Description
avs_nomore_words	No more words to return from counts
avs_ok	Success
avs_open_err	Cannot open or create an index
avs_outofrange_err	Value out of range
avs_parse_err	Trouble parsing query
avs_rankterm_err	Unknown ranking term
avs_resultnum_err	Invalid result number specified
avs_search_err	Search object has no search context (VB,C++ api)
avs_startdoc_err	Startdoc/enddoc sequence error
avs_sync_err	Error synchronizing the read lock
avs_unk_exception_err	Unhandled exception error (C++ api)
avs_update_err	Index not open for update
avs_version_err	Caller/library version mismatch

Document Conversion API

A new document converter API that converts various document types to text is now available with this release. This API embodies document conversion technologies from Inso Corporation, Adobe Systems Inc., and Compaq Computer Corporation. Currently, conversion to HTML is only supported for PDF documents.

Class AvsDocument

The AvsDocument object the contains document converter methods and properties.

convert_file2html function

Converts a document to HTML.

```
Function convert_file2html(DocPath As String, TextPath As
String) As Long
```

Argument

DocPath	The path of the document to be converted.
TextPath	The path of the converted text document.

Description

The convert_file2html function converts the specified file to an HTML file. You must supply the path of the file to be converted and the path of the output file. Currently, only PDF files can be converted to HTML.

Return Value

Returns 0 or an error code.

See Also

- convert_file2text
- cvterrmsg

convert_file2text function

Converts a document to text.

```
Function convert_file2text(DocPath As String, TextPath As
String) As Long
```

Argument

DocPath	The path of the document to be converted.
TextPath	The path of the converted text document.

Description

The convert_file2text function converts a document to text. You must specify the pathname to the document to be converted as well as specify a pathname to the file which is to contain the converted text. The document contents are analyzed before conversion to determine the document type.

Note: The file containing the converted text may contain no line ending characters. For more information on the file types available for conversion, see the File Types Table in the C Reference Manual.

Return Value

Returns 0 or an error code.

See Also

- convert_file2html
- cvterrmsg

cvterrmsg function

Converts the document converter status code to a string.

```
Function cvterrmsg(status As Long) As String
```

Argument

status	The error status code from the document object.

Description

The cvterrmsg function converts the document converter status code to a string. Use the lastcvterr method to obtain the status code.

Return Value

Returns 0 or an error code.

See Also

- convert_file2text
- convert_file2html
- errmsg
- lastcvterr

errmsg function

Converts error status code to string.

```
Function errmsg(status As Long) As String
```

Argument

status	The error status code from the document object.

Description

The errmsg function converts the AVS error status code to a string. Use the lasterror method to obtain the status code.

Return Value

Returns 0 or an error code.

See Also

- convert_file2text
- convert_file2html
- cvterrmsg

lastcvterr property

Gets last document converter error.

```
Property lastcvterror As Long read-only
```

Description

The lastcvterror property retrieves the last document converter error. Use this property to get specific document converter error code if any of the document converter methods returns status=avs_cvterr.

Return Value

Returns 0 or an error code.

See Also

- cvterrmsg
- errmsg
- lasterror

lasterror property

Gets the last error code.

```
Property lasterror As Long
read-only
```

Description

The lasterror property retrieves the last error status code. Use this property to get the specific AVS error code if any of the document converter methods fails. This is useful when using the Visual Basic On Error statement.

Return Value

Returns 0 or an error code.

See Also

- cvterrmsg
- errmsg
- lastcvterr

opt_cvtpath property

Sets the document converter input pathname.

```
Property opt_cvtpath As String
read-only
```

Description

The opt_cvtpath property sets the document converter input pathname. The dictionary file, which is used by the PostScript filter, must exist in this directory.

Return Value

Returns 0 or an error code.

See Also

- cvterrmsg
- errmsg
- lastcvterr

AVSDocument Constants

Constant	Description
avs_cvter	Document converter error.
avs_dictionary_err	Unable to open postscript dictionary file.
avs_ok	Indicates success.