
AltaVista Search Developer's Kit 97

White Paper

Preface

This white paper is intended for application developers. It provides a quick, relatively detailed overview of the AltaVista Search Developer's Kit 97 capability set. Contact Bruce Webster, the AltaVista Search product manager for an evaluation kit. (bruce.webster@digital.com)

Check out the Partner Pavilion at <http://altavista.software.digital.com> for a list of AltaVista Search VARs and System Integrators.



Executive Summary

You know the information users need is in a database somewhere. Where is it? How can you provide easy access to it? AltaVista Search Developer's Kit 97 lets you build your own search and retrieval application or add AltaVista Search powered search and retrieval capabilities to database applications and file repositories. Your users can find what they need quickly and easily without special database training. Best of all, it is fast and has minimal impact on existing database applications.

Product Description

The AltaVista Search Developer's Kit 97 provides system integrators and software developers with the tools to integrate the AltaVista Search engine technology into customized search and retrieval applications. An AltaVista Search Developer's Kit 97 powered application controls the kinds of information users can search for as well as the way the software returns and displays results to users.

The AltaVista Search Developer's Kit 97 includes the AltaVista Search indexing, querying and results engines. It also includes application-programming interfaces, APIs, to access and manipulate them. Documentation and programming examples are also included for the developer's usage.

Features / Functions and Benefits

- Developers create or embed AltaVista powered search functionality into new or existing applications with the AltaVista Search Developer's Kit 97 API's.
 - API's, Application Programming Interfaces, allow you to create and manage the AltaVista Search NI2 index.
 - API's to find and track fielded information and documents within databases and large file repositories.
 - API's for submitting queries to your application's AltaVista Search NI2 index.
- Perform super fast searches on database records or documents, BLOBS, within databases
- Users do not have to use SQL. They use the familiar AltaVista Search querying syntax to formulate their queries. Developers can make it even easier by building a user interface that provides additional simplification.
- Every word and number within your database can be indexed and queried. This includes fielded information and binary large objects.
- Supports the familiar AltaVista Search query syntax. This includes Boolean, phrase, date and nested queries. Results can be ranked by date or by calculated relevance value.
- Utilizes small code footprint within the application address space. Memory is used as available to improve performance.
- The AltaVista Search Developer's Kit 97 is extremely scalable. The FBI is using the AltaVista Search Developer's Kit 97 with an Oracle database that contains over 40,000,000 records. Query performance was improved from 20 plus hours to less than 10 seconds.

Introduction

The AltaVista™ Search Developer's Kit 97 provides system integrators and software developers the tools they need to integrate the AltaVista Search engine technology into custom applications. These custom applications provide search and retrieval capabilities for data repositories not supported by standard AltaVista Search products. Typical data repository examples are non-web based structured and non structured databases.

- Structured data repositories contain fielded data. Database applications like Oracle, Sybase, Ingres, Microsoft Access, SQL and DB2 are examples of structured repositories. Document management systems are another example.
- Unstructured repositories contain discrete files. Shared folders and directories containing large numbers of documents on file servers are examples of unstructured repository.

You can also use the AltaVista Search Developer's Kit 97 to index Gopher sites.

Customization

You have the ability to customize the kind of information for which users can search, as well as, control the way the software returns and displays results.

Requirements

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

Hardware

Development systems requirements are:

- any Alpha system running either Microsoft® Windows NT® Version 4.0 or 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 2.51.

Application runtime requirements are:

- 64 MB of RAM.
- approximately 1 GB of disk space after installation for building and storing a moderately-sized index.

Software

- C language compiler and standard libraries.
- A web browser for viewing the documentation.

Developer's Kit Components

The AltaVista Search Developer's Kit includes these components.

- **Indexing engine** – This is the same indexing engine used by the AltaVista Search Intranet eXtension 97 and the AltaVista Public Search Service on the World Wide Web.
- **APIs** – This set of routines allow applications to access and manipulate the AltaVista Search index.
- **Documentation** – Covers descriptions of the APIs, API usage, along with project planning and project suggestions.
- **Multi-purpose-programming examples** – Every programming option is coded in a sample program. These coding examples can be copied and modified by developers. The sample program has been compiled for you and is available to run on Digital UNIX or Sun Solaris (avs_sample), and Microsoft Windows NT (avs_sample.exe).
- **Development license** – Allows developers to create, demonstrate and pilot their AltaVista Search powered solutions. (Runtime licenses are required to implement / sell the AltaVista Search powered solution.)

The AltaVista Search Developer's Kit 97 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 Developer's Kit 97 is not an add-on module or option to the AltaVista Search Intranet eXtension 97 software. These are separate products. The AltaVista Search Developer's Kit does not require any other AltaVista Search software to operate. Conversely, the index you create with the AltaVista Search Developer's Kit 97 is not compatible with the AltaVista Search Intranet eXtension 97 index. Although the same index structure is used in the two products, you cannot build an index with the AltaVista Search Developer's Kit 97 that is usable by the AltaVista Search Intranet eXtension 97. The way the index is created and the results are retrieved are internal to the AltaVista Search Intranet eXtension 97 product.

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.

Licensing

The licensing and pricing of AltaVista Search Developer's Kit 97 is based on the amount of data indexed. Data is measured in gigabytes. There are two types of licenses.

- The Developer's Kit 97 license allows the developer to build, test and demonstrate an AltaVista Search powered solution.
- The runtime licenses allow the developer to implement the AltaVista Search powered solution for production usage. Seven licensing and pricing tiers are available to provide maximum flexibility in finding the most cost-effective solution for your search and retrieval needs.

As with the AltaVista Search Intranet eXtension 97, more than one index can be created on a single server at no additional cost. If the AltaVista Search powered solution is installed on another server; another AltaVista Search Developer's Kit 97 runtime license must be purchased.

OEM licensing / business arrangements are also available. Contact Jerry Loew for details.
(jerry.loew@digital.com)

How to Use the AltaVista Search Toolkit

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 AltaVista Search index.
- 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 sections that follow describe some of these procedures and tasks in more detail. The AltaVista Search Developer's Kit 97 Programming Reference provides complete details on every procedure. You can get a copy of the AltaVista Search Developer's Kit 97 Programming Reference by downloading an evaluation copy of the AltaVista Search Developer's Kit 97.

Creating the Index

To index a document, your application calls the indexer with each word in the document, passing an integer location along with the word to indicate where the word is found. It then calls a procedure to give the indexer some data that is retrieved when a query matches the document in title, filename, URL, and so forth. The integer locations can be anything you want, but, normally, you would follow one of two methods:

- **Number the documents sequentially starting at one.** The integer location for each word is then just the number of the document in which it appears. All the words in the same document have the same location. This kind of index is usually about 10% of the size of the original text. It allows most types of query, but not phrase queries or fielded queries, for example, searching for a word in the title, because the index does not know where each word is within the documents.
- **Number the words sequentially.** The integer location for each word is then just its number. This kind of index takes about 30% of the size of the original document. It allows phrased queries and fielded queries. This is how the AltaVista Search Service on the World Wide Web uses the indexer.

You could also use a hybrid of the above two methods. For example, you can number the words within a title but put all words after the title at the same location. In this way you could use phrase queries on the title, but not in the body of the document. The indexer does not know what the locations mean as long as documents do not overlap in the location space. To learn more about locations, see "The Importance of Locations" in the AltaVista Search Developer's Kit 97 Programming Reference.

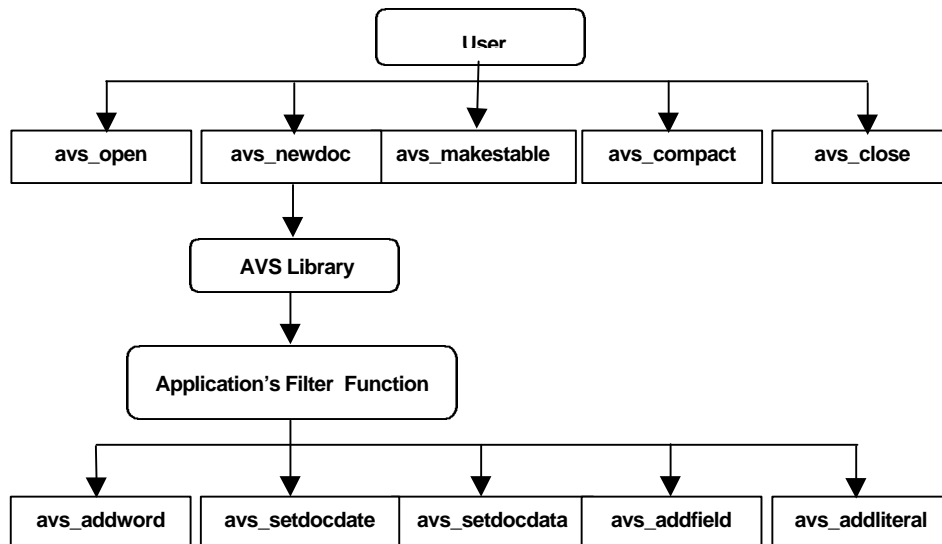
Your application should follow these basic steps to create an index:

1. Open the index with read/write capabilities using the `avs_open` function (if the index does not yet exist, this function creates it).
2. Add documents to the index using the `avs_newdoc` function.
3. Call `avs_makestable` to commit the new documents to disk. The newly added documents are not searchable until the call to `avs_makestable` occurs. This procedure also deletes those documents marked for deletion by the `avs_deletedocid` procedure.
4. Call `avs_close` to close the index.

Adding a New Document to an Index

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. Users can query the existing index while the additions and deletions take place.

This diagram illustrates the interaction of your application code and the AltaVista Search library during the addition of a new document to the index.

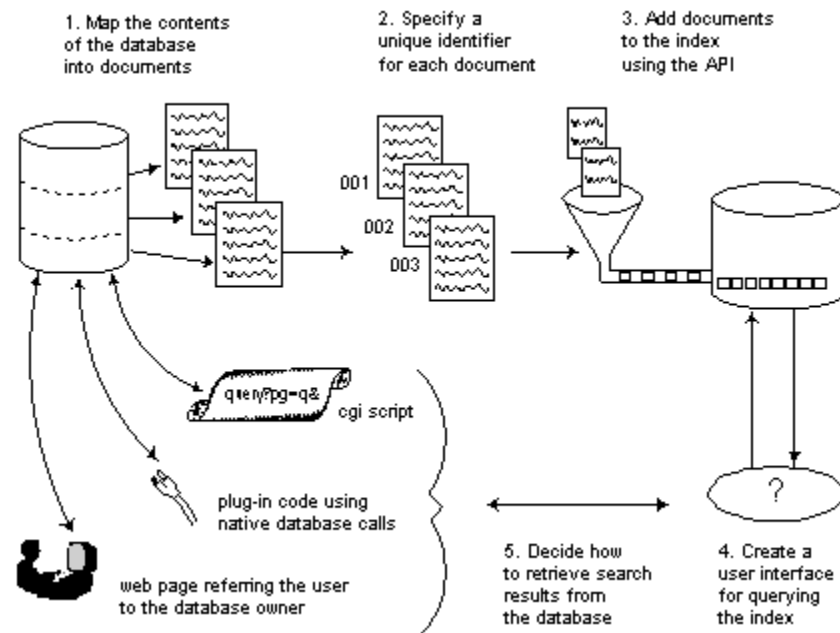


From your application, you make the appropriate calls to open the index you are populating. Use the filters to process or convert the documents you are adding to your index. Your application calls `avs_makestable` to write the index to disk, and closes the index. To access the newly built index, use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

Using the AltaVista Search SDK with Database Applications

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

1. Map the contents of the database, for example, tables, records, or reports, into documents.
2. For each document, specify a URL, an SQL instruction, and so forth, that can be used when a user's query gets a hit on the document.
3. Using the AltaVista Search Developer's Kit calls, place the document with the URL and SQL instructions into the index.
4. Decide how to access query results:
 - The URL can specify a method to retrieve the record through a cgi-bin style interface.
 - The URL can refer to a page whose contents refer to the person making the query to the "owner" of the database for more information.
 - The URL can load a plug-in which then uses native database calls from the client system into the database.



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 calls a filter, which is written or supplied by you, the application programmer.

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. Call the filter function using the following required arguments:

IN	<code>avshdl_t idx</code>	(index handle)
IN	<code>void *pFname</code>	(information identifying the document)
IN	<code>unsigned long startloc</code>	(starting location for adding words)
OUT	<code>unsigned long *pNumWords</code>	(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 and/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`, the filter can also perform the following functions.

- Set a date for the document (`avs_setdocdate`).
- Specify a data string to be returned as the result of a search (`avs_setdocdata`).
- Identify certain words to be indexed as fields (`avs_addfield`).

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 do the following. First, you would identify line 3 of each document as the "Subject:" field. Next, you would use the `avs_addfield` procedure to index it as such.

The Importance of Locations

The `avs_newdoc` procedure, in cooperation with the filter, keeps track of the starting and ending location of documents in the index. This prevents documents from overwriting other documents in the index.

When the `avs_newdoc` 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 increments 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 to `avs_newdoc` the total number of words that it added to the index. The indexing software uses this number to mark the end location of the document.

Location information is significant because without it, a group of documents in the index is actually just one long string of words. 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. Examples include.

- Searching for phrases
- Processing the NEAR advanced search operator
- Searching for certain words within a specified field

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, and 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.

Searching the Index

The AVS programming interface supports both simple and advanced searches.

You can perform simple or advanced (Boolean) searches by using the `avs_search` procedure. For performing a simple search, this procedure supports the basic operators `+` and `-` which indicate words or phrases that are required or prohibited in the search results. `avs_search` also allows advanced search capabilities that support the Boolean logic operators AND, OR, NOT, and NEAR, as well as the ability to specify ranking words that are different from the words in the search query.

Follow these basic steps to search the index:

1. Use the `avs_search` procedure to initialize a search.
2. Call `avs_getsearchresults` to retrieve specific documents that meet search criteria.
3. Optionally call any of the following procedures to retrieve additional information about search results:
 - `avs_search_getdatalen`
 - `avs_search_getdata`
 - `avs_search_getdate`
 - `avs_search_getdocid`
 - `avs_search_getrelevance`
4. Use `avs_search_close` to end the procedure and free the resources allocated for the search.

Both simple and advanced searches use an `avs_options` data structure, in which you can specify the maximum number of documents to return, and a date range within which you want to constrain the search. The options structure is defined in the `avs.h` header file. You can call the `avs_default_options` procedure to initialize its default values.

Understanding Simple and Advanced Search

Both the simple and advanced 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 (*) following three or more characters indicates a wildcard; the search engine will find all words that match the specified pattern.
- 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.

Using logical operators to refine a search

Both simple and advanced searches support the use of various operators that can help you refine the results of a search.

Simple Search Operators

Simple search supports two basic operators:

	Function
+	Includes only documents containing all specified words or phrases in the search results
-	excludes documents containing the specified word or phrase from the search results

Simple search operators must directly precede the word that the user wants to include or exclude, with no space between the operator and the word.

For example, the following simple query expression requests documents that must contain the word **results** and can also contain the phrase **year end**:

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

The following simple 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
```

Advanced Search Operators

The following are the advanced search operators and their meaning:

Keyword	Symbol	Action
AND	&	Finds only documents containing all of the specified words or phrases.
OR		Finds documents containing at least one of the specified words or phrases.
NOT	!	Excludes documents containing the specified word or phrase.
NEAR	~	Finds documents containing either specified words or phrases within 10 words of each other.

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)
```

Searching for Literal Entries Containing Special Characters

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 a word except the asterisk (*) which still works as a wildcard.

Using Dates in Your Application

When you index your documents, you can add dates through the **avs_setdocdate** procedure. A filter calls this procedure. 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 index is capable of storing dates from 01-01-1970 to 02-05-2036. It is limited by the 16-bit storage capabilities.

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.

Understanding Relevance Ranking

A feature of simple or advanced 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 one (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 user did not specify a ranking in his 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 little or no bearing on the ranking of a document. The most significant factor in determining ranking is the combined weight of words in the search query. In addition, 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.

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 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.

Each time you call the **`avs_makestable`** procedure, the newly added document information in memory is written to a new, separate file on disk. So after several `avs_makestable` calls, the on-disk index will actually consist of several files. You should periodically use the `avs_compact` procedure to merge the existing 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`**. 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. Note that compacting the index also frees the space occupied by deleted documents.

Analyzing Entries in the Index

The AltaVista Search programming interface provides a way to examine the contents of an index once it has been created. You can use the `avs_count` and related procedures as a diagnostic tool 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 expected from a query. You can also use the count procedures to obtain a general idea of the makeup of your index.

To enumerate entries in an index,

1. Use the `avs_open` procedure to open the index in read mode.
2. Use `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.
3. 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`.
4. Use `avs_count_getcount` to return the total number of words in the index that match the current search criteria.
5. Use `avs_count_getword` to retrieve the word associated with the current count.
6. End the procedure with `avs_count_close`.

Definition of Terms

Some basic terminology:

- **Document** -- A document is the highest level of aggregation recognized by the search engine. A context handle identifies it.
- **Filters** -- One or more procedures that process a document and reduce it to a series of indexable phrases (words).
- **Handle** -- The name of the index file you are creating or updating.
- **Index** -- The object that stores and processes text retrieval information. The index requires a disk directory where it stores data needed to process search requests. Search requests return a reference to a document.
- **Locations** -- Provide relative positioning within a document, phrase, or entire collection of documents.
- **Phrase** -- The lowest level of content indexing -- typically a single word.
- **Query** -- The act of searching for a unique word or words in an index and returning links to files that contain the word or words.
- **Word** -- A contiguous string of alphanumeric characters, bounded by non-alphanumeric characters (like spaces or special characters), as defined in the ISO Latin-1 standard

Summary

The AltaVista Search Developer's Kit 97 is being used by many organizations to improve ad hoc query capabilities into established database applications. Users receive improved speed and performance. Database administrators improve end user satisfaction without adding additional loads to their database application's infrastructure.

The FBI is using an AltaVista Search Developer's Kit 97 powered solution to provide ad hoc query capabilities to a 40,000,000 record Oracle database. This AltaVista Search powered solution reduced user query time from 24 hours to less than 14 seconds. Just as importantly, queries are no longer coded in SQL. Users submit queries by pushing UI buttons, selecting fields from dialog boxes and entering ad hoc information into simple fields.

You can order the AltaVista Search Developer's Kit 97 from Digital or from your Digital reseller. You can also try an evaluation copy by contacting the AltaVista Search product manager at bruce.webster@digital.com. The evaluation kit also contains a full documentation set.

Additional information can be obtained from the AltaVista Software web site at <http://altavista.software.digital.com>.

Return comments and suggestions to the AltaVista Search Marketing manager at bob.lehmenkuler@digital.com.

Notice

The information in the AltaVista Search Software Development Kit help system is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this system.

The software described in this white paper is furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph ©(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

AltaVista, Digital UNIX, and Alpha are trademarks of Digital Equipment Corporation.

All other trademarks and service marks are the property of their respective companies.

© Digital Equipment Corporation 1996, 1997. All Rights Reserved. Produced in the U.S.A.