

# CINDEX™

## Application Programming Interface

Copyright © Indexing Research 2002. All Rights Reserved

First Edition March 2002 (revised Nov 1 2002)

## Introduction

The Publishers' Edition of CINDEX for Windows provides an application programming interface (API) that allows other programs running under Windows to execute certain CINDEX operations and retrieve data from CINDEX while it is running.

The API is implemented through functions in a dynamic link library (DLL) called by the application program that wishes to communicate with CINDEX. The library is available from Indexing Research.

## Library Functions

The API provides four functions for use by the calling application. These are declared in the header file `cindexapi.h`. All functions return a result  $>0$  if they succeed, otherwise an error code (less than or equal to 0). `cinapi.h` contains a list of error return values.

### INT32 `cindex_connect(void)`

This function opens a connection to a running copy of CINDEX. It must be called before any other API operations can be undertaken.

### INT32 `cindex_disconnect(void)`

This function closes an open connection previously established with `cindex_connect()`. It should be called after all API operations are completed.

### INT32 `cindex_command(char * command, char * arg1, char * arg2)`

This function sends a command to be executed by CINDEX. Depending on the command, one or two arguments may be optional or required. `cindex_command()` is used to execute a CINDEX operation; it is not used to recover information from CINDEX.

A later section of this document describes the available commands and their arguments.

### INT32 `cindex_getdata(char * command, void *data)`

This command obtains information from CINDEX. The data are returned in the object pointed to by the pointer *data*.

A later section of this document describes the available commands and their arguments.

## Using the API

The API is supplied as the dynamic link library **`cindexapi.dll`**. This may be used by an application program either via load-time dynamic linking or run-time linking. For load-time linking an import library (**`cindexapi.lib`**) is supplied that can be linked into the application when it is built. The DLL can also be loaded at runtime through the Windows call `LoadLibrary`.

The first call to the API must be to `cindex_connect()`, and the last call to `cindex_disconnect()`. There are no restrictions on calls to `cindex_command()` or `cindex_getdata()`.

## Executing CINDEX Commands

The following paragraphs describe each of the commands that can be sent to CINDEX to execute some operation via `cindex_command()`.

INT32 `cindex_command(char * command, char * arg1, char * arg2)`

|                |  |
|----------------|--|
| <i>command</i> | A c string that names a command.           |
| <i>arg1</i>    | A pointer to an optional argument, or NULL |
| <i>arg2</i>    | A pointer to an optional argument, or NULL |

All permissible command names are defined by symbolic constants in the header file `cindexapi.h`. You should use these constants (rather than literal strings) when using the API.

**Return Value:** > 0 if success; <= 0 if error.

### **`cindex_command(CINAPI_CREATE, char *indexname, char *templatename)`**

Creates a new index *indexname* from (optional) template *templatename* if *templatename* is not NULL. Both *indexname* and *templatename* should specify a path and filename extension.

### **`cindex_command(CINAPI_OPEN, char *filename, char * filetype)`**

Opens the document *filename* of type *filetype*. *filename*, which must include a filename extension, will be opened from the CINDEX current directory unless a path is supplied. *filetype* may be one of the following symbolic constants (defined in `cindexapi.h`):

|                       |            |
|-----------------------|------------|
| CINAPI_OPENINDEX      | index      |
| CINAPI_OPENARCHIVE    | archive    |
| CINAPI_OPENSTYLESHEET | stylesheet |

If *filetype* is CINAPI\_OPENARCHIVE or CINAPI\_OPENSTYLESHEET the call will fail unless CINDEX already has an index is open. Records in an archive are read into the active index. A stylesheet is applied to the active index.

### **`cindex_command(CINAPI_CLOSE, NULL, NULL)`**

Closes the active index.

### **`cindex_command(CINAPI_CLOSE, NULL, NULL)`**

Closes the active index.

### **`cindex_command(CINAPI_QUIT, NULL, NULL)`**

Causes CINDEX to quit after a subsequent call to `cindex_disconnect()`.

### **`cindex_command(CINAPI_SAVEBACKUP, char *indexname, NULL)`**

Saves a backup copy of the active index as *indexname*. *indexname* should contain a full path, but not the filename extension “.cdx”.

**index\_command(CINAPI\_SAVEAS, char \*filename, char \*filetype)**

Saves the active index as *filename* of type *filetype*. *filename* should *not* include an extension. That is supplied automatically as appropriate for the file type being created. If *filename* does not include a path, the file is saved in CINDEX's current directory. *filetype* may be one of the following symbolic constants (defined in `indexapi.h`):

|                      |                  |
|----------------------|------------------|
| CINAPI_SAVEASINDEX   | index            |
| CINAPI_SAVEASARCHIVE | archive          |
| CINAPI_SAVEASRTF     | rich text format |

*filetype* may also be a text string that specifies the name of a CINDEX tag set. If you specify a tag set, you should not provide a path (CINDEX looks in a specified place for tag sets) but you should specify the name of the tagset with the extension “.ctg”.

**index\_command(CINAPI\_IMPORT, char \*filename, NULL)**

Imports the contents of file *filename* into the active index. *filename* should contain a path and filename extension. The file is assumed to be in the ‘.dat’ format produced by CINDEX for DOS.

**index\_command(CINAPI\_PRINT, NULL, NULL)**

Prints the active index on the default printer. CINDEX prints the whole index, the format dictated by the current document settings.

**index\_command(CINAPI\_SELECTALL, NULL, NULL)**

Selects all records in the current view of the active index.

**index\_command(CINAPI\_DELETE, NULL, NULL)**

Deletes the selected records from the active index (or restores the records if the first selected is already deleted).

**index\_command(CINAPI\_COMPRESS, NULL, NULL)**

Compresses the active index, removing deleted records, duplicate records, and empty records.

**index\_command(CINAPI\_SORT, char \*sorttype, NULL)**

Sorts the active index in the manner dictated by *sorttype*. *sorttype* may be one of the following symbolic constants:

|                  |   |
|------------------|---|
| CINAPI_SORTALPHA | sort alphabetically by the current sort settings. |
| CINAPI_SORTPAGE  | sort by locator order.                            |
| CINAPI_SORTON    | turn sorting on                                   |
| CINAPI_SORTOFF   | turn sorting off                                  |

**cindex\_command(CINAPI\_VIEW, char \*viewtype, NULL)**

Sets the current view of the active index to *viewtype*. *viewtype* may be one of the following symbolic constants:

|                      |             |
|----------------------|-------------|
| CINAPI_VIEWFORMATTED | formatted   |
| CINAPI_VIEWDRAFT     | draft       |
| CINAPI_VIEWALL       | all records |

**cindex\_command(CINAPI\_GOTO, char \*recordspec, NULL)**

Sets the display of the active index to show (and select) the record specified by *recordspec*. *recordspec* may be the beginning text of fields of a record (specified in the manner used by the CINDEX GoTo command) or a record number.

**cindex\_command(CINAPI\_SETSIZE, char \*recordsize, NULL)**

Sets the record size of the active index to *recordsize*. *recordsize* is a string representation of a decimal number.

**cindex\_command(CINAPI\_SHOWINDEX, char \*indexname, NULL)**

Makes index *indexname* the active index. The index must already be open in CINDEX. *indexname* must be fully-qualified with a path and a filename extension.

**cindex\_command(CINAPI\_FINDALL, CINDEXFINDREPLACE \* findrep, NULL)**

Finds, and displays as a temporary group, all records in the active view that meet the search criteria specified in the structure *findrep*. The structure CINDEXFINDREPLACE is defined in *cindexapi.h*. When passing the pointer to *findrep*, you should cast the pointer to (char\*).

**cindex\_command(CINAPI\_REPLACEALL, CINDEXFINDREPLACE \* findrep, NULL)**

Searches all records in the active view for the target text specified in the structure *findrep*, and replaces each occurrence with the specified replacement. The structure CINDEXFINDREPLACE is defined in *cindexapi.h*. When passing the pointer to *findrep*, you should cast the pointer to (char\*).

**cindex\_command(CINAPI\_ADDRECORD, char \*record, NULL)**

Adds a new record to the index. The record contents are contained in the c string *record* (terminated by a NULL character) that is encoded in CINDEX archive format. The archive format is described in the Appendix to this document.

**Obtaining data from CINDEX**

The following paragraphs describe each of the commands that can be sent to CINDEX to obtain data via *cindex\_getdata()*.

INT32 *cindex\_getdata*(char \* *command*, void \* *data*)

## CINDEX Application Programming Interface

*command*      A c string that specifies a command.

*data*            A pointer to the data to be filled in by CINDEX

**Return Value:** the size of the object, or  $\leq 0$  on error.

All permissible command names are defined by symbolic constants in the header file `cindexapi.h`. You should use these constants (rather than literal strings) when using the API.

Each of the available commands assumes data of a specified type (defined in `cindexapi.h`). In making the call to `cindex_getdata()` you should pass a pointer to that data object, so that CINDEX can fill the object. It is your responsibility to ensure that you have allocated sufficient memory to contain the data. If you call `cindex_getdata()` with *data* set to NULL, CINDEX will not fill the object but will return the required size in bytes.

### **`cindex_getdata(CINAPI_GETRECORDINFO, CINDEXRECORDINFO * rinfo)`**

Returns information about the structure of records in the index. The structure CINDEXRECORDINFO is defined in `cindexapi.h`.

### **`cindex_getdata(CINAPI_GETSORTINFO, CINDEXSORTINFO* sinfo)`**

Returns information about the manner in which the index is sorted. The structure CINDEXSORTINFO is defined in `cindexapi.h`.

### **`cindex_getdata(CINAPI_GETFONTINFO, CINDEXFONTINFO* finfo)`**

Returns information about the manner in which the index is sorted. The structure CINDEXFONTINFO is defined in `cindexapi.h`.

### **`cindex_getdata(CINAPI_GETSELECTEDRECORD, char *record)`**

Returns the contents of the selected record (or the first of multiple selected records) in the active view of the index . The record contents are contained in a c string (terminated by a NULL character) that is encoded in CINDEX archive format. The archive format is described in the Appendix to this document. To recover succeeding records in the index, use the command CINAPI\_GETNEXTRECORD (below).

### **`cindex_getdata(CINAPI_GETNEXTRECORD, char *record)`**

Returns the contents of the next record in the active view of the index . To retrieve the contents of a series of records, first call CINAPI\_GETSELECTEDRECORD (above) then call CINAPI\_GETNEXTRECORD to advance the selection and obtain the contents of the next record.

## Appendix: Format of archived records

In archive format the text of a record is contained in a c string, with each field of the record separated from the next by a 'tab' character. The record must contain at least two fields, a main heading and a field containing locators (page references). A record may contain up to sixteen fields, depending on the number permitted in the index that contains the record. The last field is always the locator field.

A record string may contain additional information appended to the last (locator) field, but before the terminating NULL character. This 'extended' information encodes the following:

The record status (labeled, generated or deleted).

The time at which the record was last modified (in seconds since Jan 1 1970).

The four-character identifier of the user who last modified the record.

The extended information begins with the character 0x13, and has the following format:

0x13#*time\_in\_seconds* *userid*

# is a single character that encodes record status:

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2     | Bit 1   | Bit 0   |
|-------|-------|-------|-------|-------|-----------|---------|---------|
|       | TRUE  |       |       |       | generated | labeled | deleted |

If any status bit is set, Bit 6 is TRUE and # resolves to an uppercase letter. Otherwise # is a space (0x20).

*time\_in\_seconds* is an ASCII decimal number representing the time (in seconds since Jan 1 1970) at which the record was last modified. *time\_in\_seconds* follows \_ without any additional space.

*userid* is a four-character user identifier. It is separated from *time\_in\_seconds* by a single intervening space (0x20).

### Text Styles and Fonts

Beyond characters from the ANSI character set, a record can contain, within its text fields, codes that identify type styles and fonts. A code is identified by two characters, the first of which always has the value 0x1A. The second character contains the style or font code (never both concurrently). If the character identifies a style or styles it has the following format.

| Bit 7    | Bit 6 | Bit 5     | Bit 4       | Bit 3     | Bit 2     | Bit 1   | Bit 0    |
|----------|-------|-----------|-------------|-----------|-----------|---------|----------|
| Turn off |       | subscript | superscript | smallcaps | underline | italics | boldface |

Bits 0 through 5 identify the style(s) to be turned on or turned off. The state of Bit 7 determines whether the style(s) is to be turned on or off. When TRUE, the styles are turned off. Bit 6 is always FALSE.

If the character identifies a font it has the following format.

## CINDEX Application Programming Interface

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
|       | Font  |       | id 4  | id 3  | id 2  | id 1  | id 0  |

Bits 7 and 5 are always FALSE and Bit 6 is always TRUE. Bits 0 through 4 encode one of 32 possible font identifiers. Font 0 is always the CINDEX default font; Font 1 is always the Symbol font. Fonts 2 through 31 can represent any fonts. The mapping of font identifiers to particular fonts is handled through the CINDEX font table, to which you have access via the API call CINAPI\_GETFONTINFO.